Contents

Documentation Accessibility .......................................................... 21

Part I. Preliminary Information ......................................................... 23

Chapter 1. Introduction to the Oracle Essbase API Reference .................... 25
  Essbase API Overview ................................................................... 25
  What's in This Document .......................................................... 26
  What You Should Know Before You Start ...................................... 26
  Conventions Used in this Document ............................................. 26
  Using Function Reference Entries .............................................. 27

Chapter 2. Building the Program ....................................................... 29
  Supported Compilers .............................................................. 29
  Supported Platforms .............................................................. 30
  Naming Conventions ............................................................... 31
  Including API Files in Your Program ......................................... 32
  Building a Program on UNIX .................................................... 33

Chapter 3. Integrating Essbase With Your Product .................................. 37
  Essbase Directory Structure ..................................................... 37
  Customizing the Run-Time Environment .................................... 38
    Customizing the Path to the Essbase CLIENT Directory .......... 39
    Customizing the Path to the Message Database ...................... 39
    Customizing the Path to the Essbase Login Help File ............. 40
    Creating Your Own On-line Help for AutoLogin ..................... 41
  API Files You Need to Ship .................................................... 41
  API Libraries ......................................................................... 42
  Installing Your Application Program ....................................... 43
  Optimizing TCP/IP Networking for API Clients ......................... 44

Chapter 4. Building a Simple API Program ........................................... 47
  Introduction ........................................................................... 47
  Design Environment Setup Issues ............................................. 47
C Main API Message Handling ........................................... 85
   How the Essbase C Main API Handles Messages ......................... 85
   Defining a Custom Message Function in C Programs ....................... 86
Choosing a Network Protocol ............................................ 88
Calling C Main API Functions ............................................ 88
Typical C Main API Task Sequence ........................................ 89
Initializing the C Main API .............................................. 90
Logging in to an Essbase Server ........................................... 90
Selecting an Active Application and Database ................................. 90
Retrieving and Updating Data ............................................ 91
Recalculating the Database .............................................. 92
Logging Out from the Essbase Server and Terminating the C Main API .............. 93
C Main API Common Problems and Solutions ................................ 93

Chapter 7. C Main API Declarations ........................................ 95
   Standard C Language Types .............................................. 95
      Simple Data Types (C) .............................................. 95
      Other Data Types (C) ............................................... 96
      Bitmask Data Types (C) ............................................. 96
      Pointer Types (C) .................................................. 98
      Miscellaneous Types (C) ........................................... 99
      Array Types (C) ................................................... 99
      API Definitions (C) ................................................ 100
   Constant Definitions (C) ............................................... 100
      Attributes Constants (C) ........................................... 100
      Dimension Tag Constants (C) ..................................... 103
      Implied Share Setting (C) ......................................... 103
      Information Flag Constants (C) .................................... 104
      List Option Constants (C) ........................................ 105
      Maximum String Lengths (C) ....................................... 105
      Request Type Constants (C) ....................................... 106
      Size Flag Constants (C) ........................................... 106
      Unicode Mode Constants (C) ...................................... 107
   LRO Constant and Structure Definitions (C) ................................ 107
      Constants for LROs (C) ............................................. 107
      ESS_CELLADDR_API_T ........................................... 108
      ESS_LRODESC_API_T ............................................. 108
      ESS_LROHANDLE_API_T .......................................... 109
      ESS_LROINFO_API_T ............................................. 109
Constant and Structure Definitions for Partitions (C) ..................................... 109
Drill-Through Constant and Structure Definitions ............................................. 110
  C Main API Drill-Through Constants and Structures (essdt.dll) ...................... 110
  C Main Drill-Through Constants and Structures (essdtapi.dll) ....................... 112
C Main API Structures .................................................................................. 114
  ESS_APPDB_T ......................................................................................... 114
  ESS_APPINFO_T ...................................................................................... 115
  ESS_APPINFOEX_T .................................................................................. 116
  ESS_APPSTATE_T ..................................................................................... 117
  ESS_ATTRIBUTEINFO_T ............................................................................ 118
  ESS_ATTRIBUTEVALUE_T ........................................................................... 118
  ESS_ATTRSPECS_T .................................................................................... 119
  ESS_BLDDL_STATE_T ................................................................................. 121
  ESS_CONNECTINFO_T ............................................................................... 122
  ESS_CONNECTINFOEX_T .......................................................................... 122
  ESS_DBFILEINFO_T .................................................................................. 123
  ESS_DBINFO_T ......................................................................................... 124
  ESS_DBREQINFO_T ................................................................................... 126
  ESS_DBSTATE_T ......................................................................................... 127
  ESS_DBSTATS_T ......................................................................................... 130
  ESS_DIMENSIONINFO_T ............................................................................ 132
  ESS_DIMSTATS_T ....................................................................................... 133
  ESS_DTAPICOLUMN_T ............................................................................... 134
  ESS_DTAPIDATA_T ..................................................................................... 134
  ESS_DTAPIHEADER_T .................................................................................. 135
  ESS_DTAPIINFO_T ..................................................................................... 135
  ESS_DTAPIREPORT_T .................................................................................. 136
  ESS_DTBUFFER_T ....................................................................................... 136
  ESS_DTDATA_T ......................................................................................... 136
  ESS_DTHEADER_T ...................................................................................... 137
  ESS_DISKVOLUME_REPLACE_T .................................................................... 137
  ESS_DURLINFO_T ....................................................................................... 138
  ESS_EXTUSERINFO_T ............................................................................... 138
  ESS_GENLEVELNAMEEX_T ........................................................................... 139
  ESS_GLOBAL_T ......................................................................................... 140
  ESS_INIT_T .............................................................................................. 141
  ESS_LOAD_BUFFER_T .................................................................................. 145
  ESS_LOCKINFO_T ....................................................................................... 146
  ESS_LOCKINFOEX_T ................................................................................... 146
Chapter 8. C Main API Functions

C Main API Function Categories .................................................. 199
C Main API Alias Table Functions ........................................... 199
C Main API Application Functions ........................................... 200
C Main API Attributes Functions ............................................. 201
C Main API Database Functions ............................................. 201
C Main API Database Member Functions ................................ 202
C Main API Drill-through Functions ........................................ 203
C Main API File Functions .................................................... 204
C Main API Group Administration Functions .............................. 205
C Main API Initialization and Login Functions .......................... 206
C Main API LRO Functions .................................................... 207
C Main API Location Aliases Functions ................................ 207
C Main API Memory Allocation Functions ................................ 207
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>C Main API Miscellaneous Functions</td>
<td>208</td>
</tr>
<tr>
<td>C Main API Object Functions</td>
<td>208</td>
</tr>
<tr>
<td>C Main API Partition Functions</td>
<td>208</td>
</tr>
<tr>
<td>C Main API Performance Stats Functions</td>
<td>210</td>
</tr>
<tr>
<td>C Main API Reporting, Updating, and Calculation Functions</td>
<td>210</td>
</tr>
<tr>
<td>C Main API Security Filter Functions</td>
<td>211</td>
</tr>
<tr>
<td>C Main API Substitution Variables Functions</td>
<td>211</td>
</tr>
<tr>
<td>C Main API User Administration Functions</td>
<td>212</td>
</tr>
<tr>
<td>C Main API User and Group Identity Functions</td>
<td>212</td>
</tr>
<tr>
<td>C Main API Shared Services Functions</td>
<td>214</td>
</tr>
<tr>
<td>C Main API Unicode Mode Functions</td>
<td>215</td>
</tr>
<tr>
<td>C Main API Function Reference</td>
<td>215</td>
</tr>
</tbody>
</table>

**Part III. C Outline API** ................................................................ 677

**Chapter 9. Using the C Outline API** ............................................ 679

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>C Outline API Overview</td>
<td>679</td>
</tr>
<tr>
<td>C Outline API Error Handling</td>
<td>679</td>
</tr>
<tr>
<td>C Outline API Server Outline Queries</td>
<td>680</td>
</tr>
<tr>
<td>C Outline API Outline Verification</td>
<td>680</td>
</tr>
<tr>
<td>C Outline API Memory Allocation</td>
<td>681</td>
</tr>
<tr>
<td>C Outline API Security Requirements</td>
<td>681</td>
</tr>
<tr>
<td>C Outline API Function Call Sequence</td>
<td>682</td>
</tr>
<tr>
<td>Typical C Outline API Task Sequence</td>
<td>682</td>
</tr>
</tbody>
</table>

**Chapter 10. C Outline API Declarations** ..................................... 685

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>C Outline API Error Return Values</td>
<td>685</td>
</tr>
<tr>
<td>C Outline API DTS Member Structures</td>
<td>691</td>
</tr>
<tr>
<td>C Outline API Symbolic Constant Definitions</td>
<td>692</td>
</tr>
<tr>
<td>Account Member Currency Conversion Category Values</td>
<td>693</td>
</tr>
<tr>
<td>Account Member Time Balance Skip Values</td>
<td>693</td>
</tr>
<tr>
<td>Account Member Time Balance Values</td>
<td>693</td>
</tr>
<tr>
<td>Dimension Categories</td>
<td>693</td>
</tr>
<tr>
<td>Dimension Categories (Tags)</td>
<td>694</td>
</tr>
<tr>
<td>Member Types</td>
<td>694</td>
</tr>
<tr>
<td>Generation and Level Options</td>
<td>695</td>
</tr>
<tr>
<td>Query Options</td>
<td>695</td>
</tr>
<tr>
<td>Query Types</td>
<td>695</td>
</tr>
<tr>
<td>Restructure Values</td>
<td>696</td>
</tr>
<tr>
<td>Share Constants</td>
<td>696</td>
</tr>
</tbody>
</table>
Chapter 11. C Outline API Functions ........................................... 713
C Outline API Function Categories ............................................ 713
C Outline API Alias Table Functions ........................................ 713
C Outline API Attributes Functions ....................................... 714
C Outline API Dynamic Time Series Functions ...................... 715
C Outline API Generation Name Functions ............................ 715
C Outline API Level Name Functions .................................... 715
C Outline API Member Administration Functions ...................... 715
C Outline API Member Alias Functions ................................ 716
C Outline API Member Formula Functions ............................... 716
C Outline API Member Traversal Functions ............................. 717
C Outline API Outline Administration Functions ..................... 717
C Outline API Outline Query Functions ................................. 718
C Outline API Setup and Cleanup Functions ........................... 718
C Outline API Unicode Mode Functions ................................ 718
C Outline API User-Defined Attributes Functions ................... 719
C Outline API User-Defined View Selection Functions ............. 719
C Outline API Varying Attributes Functions .......................... 719
C Outline API Function Reference .......................................... 721

Chapter 12. C Outline API Examples ........................................ 981
Example of Traversing an Outline ........................................ 981
Extended Member Query Code Example ................................. 982
## Part IV. C Grid API

### Chapter 13. Using the C Grid API
- General Information on the C Grid API................................. 991
- Overview of C Grid API Architecture.................................. 992
- C Grid API Supported Platforms and Compilers...................... 992
- Files to Include in C Grid API Programs.............................. 993
- C Grid API Initialization and Setup.................................. 993
- C Grid API Memory Management...................................... 993
- C Grid API Versioning................................................ 993
- Using the C Grid API Functions...................................... 994
- Using the C Main API Functions...................................... 994
- C Grid API Coordinate Systems..................................... 995

### Chapter 14. C Grid API Declarations
- C Grid API Constants................................................. 997
  - Grid Perspective Types........................................... 1003
  - Text List (SmartList) Types.................................... 1004
  - Unicode Mode Types............................................. 1004
- C Grid API Data Types............................................... 1004
- C Grid API Structures................................................ 1006
  - ESSG_CONNECTINFO_T............................................. 1006
  - ESSG_DATA_T..................................................... 1006
  - ESSG_DRILLDATA_T.............................................. 1008
  - ESSG_DTDATA_T................................................ 1009
  - ESSG_DTHEADER_T.............................................. 1009
  - ESSG_DTINFO_T................................................ 1010
  - ESSG_DTREPORT_T............................................. 1010
  - ESSG_INIT_T.................................................. 1011
  - ESSG_LRODESC_T.............................................. 1011
  - ESSG_LROINFO_T............................................... 1012
  - ESSG_RANGE_T................................................ 1012

### Chapter 15. C Grid API Function Reference
- EssGBeginConditionalRetrieve....................................... 1015
- EssGBeginConditionalZoomIn........................................ 1017
- EssGBeginCreateLRO................................................ 1020
- EssGBeginDataPoint................................................ 1021
- EssGBeginDeleteLROs............................................... 1023
- EssGBeginDrillAcross.............................................. 1023
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>EssGBeginDrillOrLink</td>
<td>1024</td>
</tr>
<tr>
<td>EssGBeginKeepOnly</td>
<td>1025</td>
</tr>
<tr>
<td>EssGBeginLock</td>
<td>1027</td>
</tr>
<tr>
<td>EssGBeginPivot</td>
<td>1028</td>
</tr>
<tr>
<td>EssGBeginRemoveOnly</td>
<td>1030</td>
</tr>
<tr>
<td>EssGBeginReport</td>
<td>1033</td>
</tr>
<tr>
<td>EssGBeginReportFile</td>
<td>1034</td>
</tr>
<tr>
<td>EssGBeginRetrieve</td>
<td>1037</td>
</tr>
<tr>
<td>EssGBeginSamplingZoomIn</td>
<td>1038</td>
</tr>
<tr>
<td>EssGBeginUpdate</td>
<td>1040</td>
</tr>
<tr>
<td>EssGBeginZoomIn</td>
<td>1041</td>
</tr>
<tr>
<td>EssGBeginZoomOut</td>
<td>1044</td>
</tr>
<tr>
<td>EssGC CancelOperation</td>
<td>1046</td>
</tr>
<tr>
<td>EssGCell</td>
<td>1047</td>
</tr>
<tr>
<td>EssGCreateMemberwKeyStr</td>
<td>1049</td>
</tr>
<tr>
<td>EssGConnect</td>
<td>1052</td>
</tr>
<tr>
<td>EssGConnectEx</td>
<td>1053</td>
</tr>
<tr>
<td>EssGDeleteLRO</td>
<td>1054</td>
</tr>
<tr>
<td>EssGDestroyGrid</td>
<td>1055</td>
</tr>
<tr>
<td>EssGDisconnect</td>
<td>1056</td>
</tr>
<tr>
<td>EssGDTBeginDrillThrough</td>
<td>1057</td>
</tr>
<tr>
<td>EssGDTConnect</td>
<td>1058</td>
</tr>
<tr>
<td>EssGDTEndDrillThrough</td>
<td>1058</td>
</tr>
<tr>
<td>EssGDTExecuteReport</td>
<td>1059</td>
</tr>
<tr>
<td>EssGDTGetData</td>
<td>1060</td>
</tr>
<tr>
<td>EssGDTGetHeader</td>
<td>1060</td>
</tr>
<tr>
<td>EssGDTGetInfo</td>
<td>1061</td>
</tr>
<tr>
<td>EssGDTGetReportData</td>
<td>1062</td>
</tr>
<tr>
<td>EssGDTListReports</td>
<td>1062</td>
</tr>
<tr>
<td>EssGDTReportCount</td>
<td>1063</td>
</tr>
<tr>
<td>EssGDTRequestDrillThrough</td>
<td>1064</td>
</tr>
<tr>
<td>EssGDTSetInfo</td>
<td>1065</td>
</tr>
<tr>
<td>EssGEndOperation</td>
<td>1065</td>
</tr>
<tr>
<td>EssGFreeCellLinkResults</td>
<td>1066</td>
</tr>
<tr>
<td>EssGFreeMemberInfo</td>
<td>1067</td>
</tr>
<tr>
<td>EssGFreeMemberwKeyStr</td>
<td>1067</td>
</tr>
<tr>
<td>EssGFreeRows</td>
<td>1070</td>
</tr>
<tr>
<td>EssGGetAPIContext</td>
<td>1071</td>
</tr>
<tr>
<td>EssGGetAPIInstance</td>
<td>1072</td>
</tr>
<tr>
<td>Function Name</td>
<td>Page</td>
</tr>
<tr>
<td>---------------------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>EssGGetCellLinkResults</td>
<td>1073</td>
</tr>
<tr>
<td>EssGGetDataPointResults</td>
<td>1074</td>
</tr>
<tr>
<td>EssGGetFormattedValue</td>
<td>1075</td>
</tr>
<tr>
<td>EssGGetFromMemberwKey</td>
<td>1076</td>
</tr>
<tr>
<td>EssGGetGridOption</td>
<td>1079</td>
</tr>
<tr>
<td>EssGGetGridPerspective</td>
<td>1080</td>
</tr>
<tr>
<td>EssGGetIsCellDrillable</td>
<td>1080</td>
</tr>
<tr>
<td>EssGGetLinkedPartitionDesc</td>
<td>1081</td>
</tr>
<tr>
<td>EssGGetLRO</td>
<td>1082</td>
</tr>
<tr>
<td>EssGGetLRODesc</td>
<td>1083</td>
</tr>
<tr>
<td>EssGGetMemberInfo</td>
<td>1084</td>
</tr>
<tr>
<td>EssGGetResults</td>
<td>1085</td>
</tr>
<tr>
<td>EssGGetRows</td>
<td>1086</td>
</tr>
<tr>
<td>EssGGetSmartlistforCell</td>
<td>1087</td>
</tr>
<tr>
<td>EssGInit</td>
<td>1088</td>
</tr>
<tr>
<td>EssGLoginSetPass</td>
<td>1089</td>
</tr>
<tr>
<td>EssGNewGrid</td>
<td>1090</td>
</tr>
<tr>
<td>EssGPerformOperation</td>
<td>1092</td>
</tr>
<tr>
<td>EssGSendRows</td>
<td>1092</td>
</tr>
<tr>
<td>EssGSetGridOption</td>
<td>1093</td>
</tr>
<tr>
<td>EssGSetGridPerspective</td>
<td>1095</td>
</tr>
<tr>
<td>EssGSetPath</td>
<td>1096</td>
</tr>
<tr>
<td>EssGTerm</td>
<td>1097</td>
</tr>
<tr>
<td>EssGUnlock</td>
<td>1098</td>
</tr>
<tr>
<td>EssGUpdateLRO</td>
<td>1099</td>
</tr>
<tr>
<td>EssGVersion</td>
<td>1099</td>
</tr>
</tbody>
</table>

**Chapter 16. C Grid API Examples**

<table>
<thead>
<tr>
<th>Example Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>C Grid API Example</td>
<td>1101</td>
</tr>
<tr>
<td>C Grid API Drill-Through Example</td>
<td>1105</td>
</tr>
<tr>
<td>ESSG_OP_MEMBERANDUNIQUENAME Example</td>
<td>1107</td>
</tr>
<tr>
<td>ESSG_DT_MEMBERwKEY Example</td>
<td>1109</td>
</tr>
<tr>
<td>BuildTable Example Function</td>
<td>1111</td>
</tr>
<tr>
<td>DisplayOutput Example Function</td>
<td>1112</td>
</tr>
<tr>
<td>FreeTwoDim Example Function</td>
<td>1114</td>
</tr>
</tbody>
</table>
**Chapter 17. C Grid API Error Codes** ................................................................. 1115

**Part V. Visual Basic Main API** ................................................................. 1117

**Chapter 18. Using the Visual Basic Main API** ........................................... 1119
Visual Basic Main API Conventions .......................................................... 1119
Understanding Visual Basic API Declarations ............................................ 1119
Visual Basic Functions for Excel ............................................................... 1120
Visual Basic API Handles ........................................................................ 1120
Instance Handles ...................................................................................... 1120
Context Handles ...................................................................................... 1121
Visual Basic API File Objects ................................................................... 1122
Visual Basic API Message Handling ......................................................... 1123
How the Essbase API Handles Messages ................................................. 1123
Using Message Handling in Visual Basic Programs ................................. 1124
Visual Basic API Function Calls ............................................................... 1125
Visual Basic API Function Call Sequence ................................................ 1126
Typical VB API Task Sequence ................................................................. 1127
Initializing the Visual Basic API ............................................................... 1127
Logging In to an Essbase Server .............................................................. 1128
Selecting an Active Application and Database ........................................ 1128
Retrieving and Updating Data .................................................................. 1129
Recalculating the Database ..................................................................... 1130
Logging Out from the Essbase Server and Terminating the API ............... 1131
Visual Basic Main API Common Problems and Solutions ......................... 1132
Drill-through Visual Basic API Example .................................................. 1132

**Chapter 19. Visual Basic Main API Declarations** ....................................... 1153
Constant Definitions ............................................................................... 1153
Maximum String Lengths ......................................................................... 1153
Information Flag Constants ..................................................................... 1154
Size Flag Constants .................................................................................. 1154
Dimension Tag Constants ........................................................................ 1154
Constant and Structure Definitions for Linked Objects ............................ 1155
Constants for LROs .................................................................................. 1155
ESB_CELLADDR_API_T ................................................................. 1156
ESB_LRODESC_API_T ......................................................................... 1156
ESB_LROHANDLE_API_T .................................................................... 1156
ESB_LROINFO_API_T ........................................................................ 1157
Constant and Structure Definitions for Partitions ................................... 1157
<table>
<thead>
<tr>
<th>Type Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard C Language Types</td>
<td>1157</td>
</tr>
<tr>
<td>Standard Visual Basic Language Types</td>
<td>1159</td>
</tr>
<tr>
<td>Visual Basic API Attributes Terminology</td>
<td>1164</td>
</tr>
<tr>
<td>Visual Basic Main API Structures</td>
<td>1165</td>
</tr>
<tr>
<td>ESB_APPDB_T</td>
<td>1165</td>
</tr>
<tr>
<td>ESB_APPINFO_T</td>
<td>1165</td>
</tr>
<tr>
<td>ESB_APPINFOEX_T</td>
<td>1166</td>
</tr>
<tr>
<td>ESB_APPSTATE_T</td>
<td>1167</td>
</tr>
<tr>
<td>ESB_ATTRIBUTEINFO_T</td>
<td>1168</td>
</tr>
<tr>
<td>ESB_ATTRSPECS_T</td>
<td>1169</td>
</tr>
<tr>
<td>ESB_DBFILEINFO_T</td>
<td>1170</td>
</tr>
<tr>
<td>ESB_DBINFO_T</td>
<td>1171</td>
</tr>
<tr>
<td>ESB_DBREQINFO_T</td>
<td>1173</td>
</tr>
<tr>
<td>ESB_DBSTATE_T</td>
<td>1173</td>
</tr>
<tr>
<td>ESB_DBSTATS_T</td>
<td>1176</td>
</tr>
<tr>
<td>ESB_DIMENSIONINFO_T</td>
<td>1177</td>
</tr>
<tr>
<td>ESB_DIMSTATS_T</td>
<td>1178</td>
</tr>
<tr>
<td>ESB_DURLINFO_T</td>
<td>1179</td>
</tr>
<tr>
<td>ESB_GLOBAL_T</td>
<td>1179</td>
</tr>
<tr>
<td>ESB_INIT_T</td>
<td>1180</td>
</tr>
<tr>
<td>ESB_LOCKINFO_T</td>
<td>1181</td>
</tr>
<tr>
<td>ESB_MBRALT_T</td>
<td>1181</td>
</tr>
<tr>
<td>ESB_MBRUSER_T</td>
<td>1182</td>
</tr>
<tr>
<td>ESB_MEMBERINFO_T</td>
<td>1182</td>
</tr>
<tr>
<td>ESB_OBJDEF_T</td>
<td>1184</td>
</tr>
<tr>
<td>ESB_OBJINFO_T</td>
<td>1184</td>
</tr>
<tr>
<td>ESB_PART_CONNECT_INFO_T</td>
<td>1185</td>
</tr>
<tr>
<td>ESB_PART_DEFINED_T</td>
<td>1186</td>
</tr>
<tr>
<td>ESB_PART_INFO_T</td>
<td>1186</td>
</tr>
<tr>
<td>ESB_PART_REPL_T</td>
<td>1188</td>
</tr>
<tr>
<td>ESB_PARTOTL_QRY_FILTER_T</td>
<td>1188</td>
</tr>
<tr>
<td>ESB_PARTOTL_QUERY_T</td>
<td>1189</td>
</tr>
<tr>
<td>ESB_PARTSLCT_T</td>
<td>1189</td>
</tr>
<tr>
<td>ESB_PROCSTATE_T</td>
<td>1190</td>
</tr>
<tr>
<td>ESB_RATEINFO_T</td>
<td>1191</td>
</tr>
<tr>
<td>ESB_TIMERECORD_T</td>
<td>1191</td>
</tr>
<tr>
<td>ESB_USERAPP_T, ESB_GROUPAPP_T</td>
<td>1192</td>
</tr>
<tr>
<td>ESB_USERDB_T, ESB_GROUPDB_T</td>
<td>1192</td>
</tr>
<tr>
<td>ESB_USERINFO_T, ESB_GROUPINFO_T</td>
<td>1193</td>
</tr>
</tbody>
</table>
Chapter 20. Visual Basic Main API Functions ............................................ 1199
Visual Basic Main API Function Categories ............................................ 1199
VB Main API Alias Table Functions ............................................ 1199
VB Main API Application Functions ............................................ 1200
VB Main API Attributes Functions ............................................ 1201
VB Main API Database Functions ............................................ 1201
VB Main API Database Member Functions ............................................ 1202
VB Main API Drill-through Functions ............................................ 1202
VB Main API File Functions ............................................ 1203
VB Main API Group Administration Functions ............................................ 1203
VB Main API Initialization and Login Functions ............................................ 1204
VB Main API LRO Functions ............................................ 1204
VB Main API Location Aliases Functions ............................................ 1205
VB Main API Miscellaneous Functions ............................................ 1205
VB Main API Object Functions ............................................ 1206
VB Main API Reporting, Updating, and Calculation Functions ............................................ 1206
VB Main API Security Filter Functions ............................................ 1207
VB Main API Substitution Variables Functions ............................................ 1208
VB Main API User Administration Functions ............................................ 1208
Visual Basic Main API Function Reference ............................................ 1209

Part VI. Visual Basic Outline API ..................................................... 1443
Chapter 21. Using the Visual Basic Outline API ........................................... 1445
About the Visual Basic Outline API ............................................ 1445
Visual Basic Outline API Error Handling ............................................ 1445
Visual Basic Server Outline Queries ............................................ 1446
Visual Basic Outline API Outline Verification ............................................ 1446
Visual Basic Outline API Memory Allocation ............................................ 1447
Visual Basic Outline API Security Requirements ............................................ 1447
Visual Basic Outline API Function Call Sequence ............................................ 1448
Typical Visual Basic Outline API Task Sequence ............................................ 1448

Chapter 22. Visual Basic Outline API Declarations ........................................... 1451
VB Outline Error Return Values ............................................ 1451
VB Outline Symbolic Constant Definitions ............................................ 1454
ESB_ATTRIBUTEQUERY_T ............................................ 1458
ESB_GENLEVELNAME_T ............................................ 1459
Chapter 23. Visual Basic Outline API Functions

Visual Basic Outline API Function Categories

- VB Outline API Alias Table Functions
- VB Outline API Attributes Functions
- VB Outline API Dynamic Time Series Functions
- VB Outline API Generation Name Functions
- VB Outline API Level Name Functions
- VB Outline API Member Administration Functions
- VB Outline API Member Alias Functions
- VB Outline API Member Formula Functions
- VB Outline API Member Traversal Functions
- VB Outline API Outline Administration Functions
- VB Outline API Outline Query Functions
- VB Outline API Setup and Cleanup Functions
- VB Outline API User Attribute Functions

Visual Basic Outline API Function Reference

Chapter 24. Example of Traversing an Outline (VB)

Part VII. Other APIs

Chapter 25. Java API Reference

Chapter 26. MDX Provider API

MDX Provider API General Information

MDX Provider API Reference

- MDX Provider Declarations
- EssMdxExecuteQuery
- EssMdxFreeQuery
- EssMdxGetAxes
- EssMdxGetAxisInfo
- EssMdxGetAxisMembers
- EssMdxGetCellAtIndices
- EssMdxGetCellAtOffset
- EssMdxGetCellInfo
Chapter 27. Welcome to XMLA Reference ................................................................. 1619

Chapter 28. Working with XMLA ................................................................. 1621

Key Features ........................................................................................................ 1621
Methods ............................................................................................................... 1621
Discover ............................................................................................................. 1622
Execute ............................................................................................................. 1625
XMLA Rowsets ................................................................................................. 1627
CATALOGS Rowset ......................................................................................... 1627
MDSCHEMA_CUBES Rowset .......................................................................... 1629
MDSCHEMA_DIMENSIONS Rowset ............................................................... 1630
MDSCHEMA_FUNCTIONS Rowset ................................................................. 1633
MDSCHEMA_HIERARCHIES Rowset ............................................................... 1635
MDSCHEMA_MEASURES Rowset .................................................................. 1638
Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible, with good usability, to the disabled community. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Accessibility standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For more information, visit the Oracle Accessibility Program Web site at http://www.oracle.com/accessibility/.

Accessibility of Code Examples in Documentation

Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation

This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

Access to Oracle Support for Hearing-Impaired Customers

Oracle customers have access to electronic support through My Oracle Support or by calling Oracle Support at 1.800.223.1711. Hearing-impaired customers in the U.S. who wish to speak to an Oracle Support representative may use a telecommunications relay service (TRS). Information about the TRS is available at http://www.fcc.gov/cgb/consumerfacts/trs.html/, and a list of telephone numbers is available at http://www.fcc.gov/cgb/dro/trspnbeq.html. International hearing-impaired customers should use the TRS at +1.605.224.1837. An Oracle Support engineer will respond to technical issues according to the standard service request process.
In Preliminary Information:

- Introduction to the *Oracle Essbase API Reference*
- Building the Program
- Integrating Essbase With Your Product
- Building a Simple API Program
- Unicode Issues in Essbase API Programs
Essbase API Overview

Oracle Essbase provides a business performance management solution that satisfies the complex calculation requirements of end-user analysts across the enterprise in various departments, including finance, accounting, and marketing. Essbase operates in a client-server computing environment on a local area network (LAN), enabling multiple users to retrieve and analyze centralized data.

Essbase client tools provide access to centralized data through a variety of interfaces, including:

- Spreadsheet interfaces.
- Application and data management facilities.
- Custom programs you can develop using the Essbase Application Programming Interface (API).

The Essbase API provides a range of powerful and sophisticated features, including:

- Transparent client-server access
- Data manipulation, consolidation and reporting
- Encapsulated server login procedure
- Remote file management
- Application and database administration
- User and group administration
- Transparent, built-in security
- Customized memory and message handling
- Multiple platform support
• Function library that allows direct creation, manipulation, and maintenance of database outlines from a C or Visual Basic program

• For a list of new features, see the Oracle Essbase New Features.

The API is an interface between your custom client program and Essbase and manages the transfer of data between client and server. Your program makes calls to functions within the API, and data is returned from the Essbase servers you connect to.

You can also run custom programs on the server machine, using the same API functions as on the client. You don’t have to be concerned about where the Essbase Server computer is located on the network when writing a custom API program. Locating the server and transferring data is handled by the API.

Before you write programs for the API, use this document to become familiar with some of the concepts and conventions it uses.

The API functionality is contained in header files you include in the source code of your program and a set of libraries that you link to your program.

What's in This Document

This document is designed for programmers who develop custom front-end programs that access the Essbase Server.

The Oracle Essbase API Reference is a comprehensive reference to the functions and libraries you can use to develop custom front-end programs that access Essbase application servers. The document provides:

• General information about installing and using the API
• Specific reference material for programmers in C, Java, and Visual Basic development environments

What You Should Know Before You Start

To use this document, you need the following:

• A working knowledge of the operating system your server and clients use.
• An understanding of Essbase.
• Knowledge of programming in Windows or UNIX.
• Familiarity with C, Visual Basic, or Java.

Conventions Used in this Document

Table 1 lists the conventions that the Oracle Essbase API Reference uses to make code and examples easier to understand.
Table 1  Syntax and Text Formatting Conventions

<table>
<thead>
<tr>
<th>Convention</th>
<th>Purpose</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>monospace font</td>
<td>Function, structure, file, directory, and environment variables names in text</td>
<td>ESS_STS_T, ESSAPIW.LIB</td>
</tr>
<tr>
<td>* italic</td>
<td>Anything you replace with a value in syntax</td>
<td>EsbOtiCloseOutline (hOutline)</td>
</tr>
<tr>
<td>&quot; &quot;</td>
<td>Double quotes enclose text parameters or parameters that include a space</td>
<td>&quot;appName&quot;</td>
</tr>
<tr>
<td>()</td>
<td>Parentheses enclose function parameters, show order of execution for operations</td>
<td>EsbOtiDeleteMember (hOutline, hMember);</td>
</tr>
<tr>
<td>//</td>
<td>Comment marker indicates text from // to end of line should be ignored in processing</td>
<td>// Gets results</td>
</tr>
<tr>
<td>;</td>
<td>Statement terminator marks end of command</td>
<td>EXIT;</td>
</tr>
</tbody>
</table>

Using Function Reference Entries

Table 2 lists the information supplied by API function entries.

Table 2  API Function Entries

<table>
<thead>
<tr>
<th>Function Entry</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Brief description of the function.</td>
</tr>
<tr>
<td>Parameters</td>
<td>Definitions of the parameters of the function.</td>
</tr>
<tr>
<td>Return Value</td>
<td>Value returned by the function.</td>
</tr>
<tr>
<td>Notes</td>
<td>Notes on using the function.</td>
</tr>
<tr>
<td>Access</td>
<td>Level of security or other access required to use the function.</td>
</tr>
<tr>
<td>Example</td>
<td>How to use the function.</td>
</tr>
<tr>
<td>See Also</td>
<td>Related functions.</td>
</tr>
</tbody>
</table>
In This Chapter

- Supported Compilers ...................................................................................... 29
- Supported Platforms....................................................................................... 30
- Naming Conventions....................................................................................... 31
- Including API Files in Your Program ............................................................... 32
- Building a Program on UNIX .............................................................................. 33

Supported Compilers

Table 3 lists the compilers that the current release of the Essbase API supports.

<table>
<thead>
<tr>
<th>Platform</th>
<th>Compiler</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows 2000/XP/2003</td>
<td>Microsoft Visual Basic 6.0</td>
</tr>
<tr>
<td></td>
<td>Microsoft Visual C++ .NET 2003</td>
</tr>
<tr>
<td></td>
<td>Microsoft Visual Studio 2005</td>
</tr>
<tr>
<td></td>
<td>Microsoft Visual Studio 2008</td>
</tr>
<tr>
<td>HP-UX</td>
<td>Hewlett Packard C compiler for HP-UX Version 11.01 or later</td>
</tr>
<tr>
<td>AIX</td>
<td>IBM C compiler for AIX Version 4.0 or later</td>
</tr>
<tr>
<td>Solaris</td>
<td>Solaris C/C++ compiler Version 5.3 or later</td>
</tr>
<tr>
<td>Red Hat Linux</td>
<td>GCC compiler included with Red Hat Linux AS 4.0.</td>
</tr>
</tbody>
</table>

Note: The Essbase API does not support VB.NET

Sample Windows Make File

The following is a sample make file for 32-bit Windows. See also Support on 64-Bit Platforms and “Building a Program on UNIX” on page 33.

```make
# Common settings for nti

UTF8 = 1
```
# Essbase include and library path
#============================================
ESSINCDIR = /I$(<Location of API>)\api\include
ESSLIBDIR = /LIBPATH:$(<Location of API>)\api\lib

# MSDEV compiler options
#============================================
CP = cp
MKDIR = mkdir
RM = rm
MAKE = nmake
CC = cl
CPPC = cl
LINK = link
SVRLINK = link
CFLAGS = /nologo /MLd /c /w /Zp1 /DAD_UTF8
CPPFLAGS = /nologo /MLd /c /w /Zp1 /DAD_UTF8
LFLAGS = /nologo /DEBUG /MACHINE:I386
LPPFLAGS = /nologo /DEBUG /MACHINE:I386
LIBFLAGS = /nologo /MACHINE:I386 /DEBUG

STDLIBS = kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib advapi32.lib
          shell32.lib ole32.lib oleaut32.lib uuid.lib odbccp32.lib kernel32.lib
          user32.lib gdi32.lib winspool.lib comdlg32.lib advapi32.lib shell32.lib ole32.lib
          oleaut32.lib uuid.lib odbccp32.lib odbccp32.lib

!IF "$(UTF8)" == "0"
ESSLIBS = essapin.lib essgapin.lib essotln.lib
!ELSE
ESSLIBS = essapinu.lib essgapinu.lib essotlnu.lib
!ENDIF

Supported Platforms
For a list of platforms the current release of the Essbase API supports, see the Oracle Hyperion

Support on 64-Bit Platforms
- Client programs developed using the Essbase C API or Visual Basic API can be run on 32-bit platforms connecting to either 32-bit or 64-bit Essbase servers.
- Precompiled client programs developed using the 32-bit Essbase Visual Basic API can be run on the 64-bit Windows platform connecting to 64-bit Essbase servers, as long as the 32-bit runtime environment is set up correctly.
- Client programs developed using the Essbase C API can be run on 64-bit platforms connecting to 64-bit Essbase servers.
- When running the precompiled 32-bit client program on the 64-bit machine, run it from a command prompt or other shell window where ESSBASEPATH is set to the installation
directory of the 32-bit runtime client, and PATH is set to include the `bin` subdirectory under the ESSBASEPATH directory.

- To build 64-bit objects on Windows, use the following compiler and linker flags:
  - Compiler:
    - `DBIT64 -DWIN64`
  - Linker (Itanium based processor):
    - `/MACHINE:IA64`
  - Linker (Intel and AMD based processor):
    - `/MACHINE:AMD64`

### Naming Conventions

The API uses its own naming conventions for functions, constants, and data types. To ensure compatibility with future releases of the API, use these constants and data type declarations in your program:

- **Function names**—Describe the action the function performs. A name is made of a prefix that represents the interface, followed by one or more words or fragments that describes the action and its object. The parts of the name are not separated by spaces but are capitalized for easier interpretation. Names follow this format:

<table>
<thead>
<tr>
<th>Format and Parts of Name</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>InterfaceVerbObject</code></td>
<td><code>Interface</code></td>
</tr>
<tr>
<td>Programming interface</td>
<td>EssCreateGroup</td>
</tr>
<tr>
<td>Ess = C API</td>
<td>EssUpdate = C API</td>
</tr>
<tr>
<td>Esb = Visual Basic API</td>
<td>EsbUpdate = Visual Basic API</td>
</tr>
<tr>
<td>EssOtl = C Outline API</td>
<td>EssOtlOpenOutline = C Outline API</td>
</tr>
<tr>
<td>EsbOtl = Visual Basic Outline API</td>
<td>EsbOtlOpenOutline = Visual Basic Outline API</td>
</tr>
<tr>
<td>EssG = C Grid API</td>
<td>EssGSetGridOption = C Grid API</td>
</tr>
</tbody>
</table>

- **Data structure names**—Begin with a prefix that represents the interface, includes a word or fragment that describes the structure, and ends with a suffix indicating either typedef definition or macro. Underscore characters separate the parts of the name. Names follow this format:

<table>
<thead>
<tr>
<th>Format and Parts of Name</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Interface_Name_Type</code></td>
<td><code>Interface</code></td>
</tr>
</tbody>
</table>
Including API Files in Your Program

To use the Essbase API in your program, you must include the file that contains API definitions. This topic describes the files you need for the C or Visual Basic API.

Header Files

If your program is using the Main API, essapi.h should be included. If it is using the Outline API, essotl.h should be included. If it is using the Grid API, essgapi.h should be included.

API Files for C Programs

To use the Main API in a C program, you must include the API header definitions file (ESSAPI.H) in the appropriate source modules. Always include this file after any C run-time library header files. If you are programming in the Windows environment, place ESSAPI.H after the Windows include file WINDOWS.H.

C Compiler Options (32–Bit Windows Only)

If you are using an encapsulated C development environment, such as Microsoft Visual C++, you should check the compiler and linker options carefully to ensure that the API will work correctly. In particular, you must ensure that structure fields are byte-aligned, and that the correct libraries are used. Make sure to include the appropriate API library in your link process (see “API Libraries” on page 42).

The following program statements will ensure byte alignment and should be placed in the INCLUDE section of the program:

```c
#ifdef WINNT
#pragma pack (1)
#endif
#include
#include
```
API Files for Visual Basic Program (Windows Only)

To use the API in a Visual Basic program, you must include the ESB32.BAS file for 32-bit programs. This file contains the constant definitions and declarations for all Essbase functions. You can use the file as shipped, or customize it to meet the needs of your application.

To use ESB32.BAS:

1. Open a project.
2. Choose File, then Add File, and specify `%ESSBASEPATH%\API\INCLUDE\ESB32.BAS`.

Building a Program on UNIX

The Essbase API is supported on the same platforms Essbase supports: HP-UX, AIX, Solaris, and Linux. The Essbase API supports the same CPU architectures (with regard to 32- and 64-bit) that are supported by Essbase. See the Oracle Hyperion Enterprise Performance Management System Installation and Configuration Guide.

This topic provides the information needed to compile an application program using the API on UNIX.

Memory Allocation

The Essbase API for UNIX uses the standard C library memory allocation functions, malloc(), realloc(), and free(), as the default memory functions. You use the default memory functions if you pass NULLs in the AllocFunc, ReallocFunc, and FreeFunc fields of the ESS_INIT_T initialization structure. See “Using Memory in C Programs” on page 84 for more information.

UNIX Support

EssAutoLogin() is not supported in the UNIX versions of the Essbase API.

Be sure to follow UNIX file-naming conventions when using UNIX versions of the Essbase API.

HP-UX Information

- HP-UX-supplied files—For a listing of files supplied with Essbase API for HP-UX, see “API Libraries” on page 42.

  Use the -L flag to tell the linker where to locate the shared libraries:

  $(CC) file1.o file2.o -L /essbase/lib -lessapi \ $(LIBS) -o

  All libess*.sl files are linked with the +s flag which allows you to use the SHLIB_PATH search path to locate the shared library when the linked program is run. For further information about SHLIB_PATH, please check HP-UX programming documentation.
## Linking programs on HP-UX

With the Essbase 6.0 release, you must use aCC to link your program to maintain compatibility with the third party libraries used with Essbase. If you are using an earlier version, you should use the ld compiler for linking.

## HP-UX Make File example

The following sample shows a make file for HP-UX.

```bash
# Compiler Flags
CC=cc
CFLAGS = -I$(<Location of API>/api/include -g

# Library files;
LIBS = -L$(<Location of API>/api/lib -lessapinu -lessotlnu -lessgapinu

main: main.o
   $(CC) -o $@ $^ $(LIBS)

main.o: main.c
   $(CC) $(CFLAGS) $< -c -o $@

Modify this sample make file to reflect the directories where you have installed the API files and add other compiling options you want to use.

Even though the link line only specifies three libraries to link against, all of the .sl files must be available at runtime.

## HP-UX 64-bit Make File example

On 64-bit HP-UX, use the compiler flag +DD64. No linker flag is needed.

```bash
# Compiler Flags
CC=cc
CFLAGS = +DD64 -I$(<Location of API>/api/include -g

# Library files;
LIBS = -L$(<Location of API>/api/lib -lessapinu -lessotlnu -lessgapinu

main: main.o
   $(CC) -o $@ $^ $(LIBS)

main.o: main.c
   $(CC) $(CFLAGS) $< -c -o $@

Modify this sample make file to reflect the directories where you have installed the API files and add other compiling options you want to use.

Even though the link line only specifies three Essbase libraries to link against, all of the .so files must be available at runtime.

## AIX Information

### AIX-supplied files

For a listing of files supplied with Essbase API for AIX, see “API Libraries” on page 42.

### AIX Make File example

The following sample shows a make file for AIX.

```bash
# Compiler Flags
CC=cc_r
```
CFLAGS = -qcpluscmt -I$(<Location of API>)/api/include -g

# Library files;
LIBS = -L$(<Location of API>)/api/lib -lessapinu -lessotlnu -lessgapinu
main: main.o
   $(CC) -o $@ $^ $(LIBS)

main.o: main.c
   $(CC) $(CFLAGS) $< -c -o $@

Modify this sample make file to reflect the directories where you have installed the API files and add other compiling options you want to use.

For 64-bit AIX, use the -q64 -DAIX64 -DBIT64 compiler and -b64 linker flags.

Solaris Information

- **Solaris-supplied files**—For a listing of files supplied with Essbase API for Solaris, see “API Libraries” on page 42.

- **Solaris Make File example**—The following sample shows a make file for Solaris.


```
# Compiler Flags
CC=cc
CFLAGS = -I$(<Location of API>)/api/include -g

# Library files;
LIBS = -L$(<Location of API>)/api/lib -lessapinu -lessotlnu -lessgapinu
main: main.o
   $(CC) -o $@ $^ $(LIBS)

main.o: main.c
   $(CC) $(CFLAGS) $< -c -o $@

Modify this sample make file to reflect the directories where you have installed the API files and add other compiling options you want to use.

For 64-bit Solaris, use the -xarch=generic64 -DBIT64 compiler and -xarch=generic64 linker flags.

Red Hat Linux Information

- **Red Hat Linux-supplied files**—For a listing of files supplied with Essbase API for Red Hat Linux, see “API Libraries” on page 42.

- **Red Hat Linux Make File example**—The following listing shows a sample make file to compile and link a Red Hat Linux API program using the GCC compiler:

```
# Compiler Flags
CC=gcc
CFLAGS = -I$(<Location of API>)/api/include -g

# Library files;
LIBS = -L$(<Location of API>)/api/lib -lessapinu -lessotlnu -lessgapinu
```

Building a Program on UNIX
main: main.o
    $(CC) -o $@ $^ $(LIBS)

main.o: main.c
    $(CC) $(CFLAGS) $< -c -o $@

**Linux 64-bit Make File example**—On 64-bit Linux, use the compiler flag `-DBIT64`.

```bash
# Compiler Flags
CC=gcc
CFLAGS = -I$(<Location of API>)/api/include -g -DBIT64

# Library files;
LIBS = -L$(<Location of API>)/api/lib -lessapinu -lessotlnu -lessgapinu

main: main.o
    $(CC) -o $@ $^ $(LIBS)

main.o: main.c
    $(CC) $(CFLAGS) $< -c -o $@
```

Modify the sample make file to reflect the directories where you have installed the API files and add other compiling options you want to use.
Essbase Directory Structure

An computer that has the Essbase client programs installed uses a predefined directory structure, described in Table 4. The exact name of the root directory depends on the name selected during user installation, but the structure under the root directory is always the same.

Table 4 Predefined Directory Structure for Essbase Installations

<table>
<thead>
<tr>
<th>Directory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\root</td>
<td>Root directory: All Essbase files</td>
</tr>
<tr>
<td>\root\bin</td>
<td>Binary directory: executables, shared libraries, and other program files</td>
</tr>
<tr>
<td>\root\client</td>
<td>Client directory: Client application and database files</td>
</tr>
<tr>
<td>\root\client\appname</td>
<td>Files relating to the application appName (one for each application)</td>
</tr>
<tr>
<td>\root\client\appname\dbname</td>
<td>Files relating to the database dbName in the application appName (one for each database in the application)</td>
</tr>
</tbody>
</table>

The root directory can have any name the user chooses at installation time.

Note: The root directory name cannot include spaces.
<table>
<thead>
<tr>
<th>Component</th>
<th>Directory</th>
</tr>
</thead>
</table>
| Run-Time Client, 32-bit          | `%EPM_ORACLE_HOME%/common\EssbaseRTC\releaseNumber`  
  For example, `%EPM_ORACLE_HOME%/common\EssbaseRTC\11.1.2.0` |
| Run-Time Client, 64-bit          | `%EPM_ORACLE_HOME%/common\EssbaseRTC-64\releaseNumber`  
  For example, `%EPM_ORACLE_HOME%/common\EssbaseRTC-64\11.1.2.0` |
| API, 32-bit                      | `%EPM_ORACLE_HOME%/products\Essbase\EssbaseClient\api` |
| API, 32-bit (installed on 64-bit) | `%EPM_ORACLE_HOME%/products\Essbase\EssbaseClient-32\api` |
| API, 64-bit                      | `%EPM_ORACLE_HOME%/products\Essbase\EssbaseClient\api` |
| Oracle runtime files, 32-bit     | `%EPM_ORACLE_HOME%/bin-32`                     |
| Oracle runtime files, 64-bit     | `%EPM_ORACLE_HOME%/bin`                        |

**Customizing the Run-Time Environment**

The Essbase API allows you to customize access to some of the API features, so you can integrate these features with your programs. Besides customizing the memory management and message handling, you can customize the items described in the these topics:

- “Customizing the Path to the Essbase CLIENT Directory” on page 39
- “Customizing the Path to the Message Database” on page 39
- “Customizing the Path to the Essbase Login Help File” on page 40
- “Creating Your Own On-line Help for AutoLogin” on page 41

You can change each of these paths by passing an entry into the appropriate field of the Essbase API initialization structure when you call `EsxInit()`. Because you can change these paths, you can install these directories and files anywhere you like and rename them if you desire.

You might want to place the files associated with your program in a specific directory. If this is the case, you should set these paths explicitly in `ESX_INIT_T`.

An alternative to setting the paths explicitly is to rely on the user's ESSBASEPATH and ARBORMSGPATH environment variables. When you call `EsxInit()`, the API can define the paths in the initialization structure based on the root directory of any pre-existing Essbase files (ESSBASEPATH) or on ARBORMSGPATH.

**Note:** All settings in the initialization structure apply only to the calling program's instance of the API library. Custom settings within your program do not affect any other programs using the API library.
Customizing the Path to the Essbase CLIENT Directory

The API uses the CLIENT directory to store any local application or database related files (such as database outlines or report scripts). The directory structure within the CLIENT directory mirrors that of the \App\directory on the Essbase Server. Each application has its own sub-directory, and within each application sub-directory, each database in that application has a separate sub-directory. The list of applications and databases need not match that of any particular server.

Although the structure of the application and database sub-directories is fixed, you can customize the client directory under which the application directories are created.

Setting the Local Path Field of the Initialization Structure

The primary way to set the client directory path is to explicitly set the LocalPath field in the API initialization structure to point to a string indicating the full path name of the CLIENT directory. This setting causes the API to look in this directory for all client application and database related files. For example, to set the CLIENT directory to D:\PRODUCT\CLIENT, make the following change to the initialization structure:

```csharp
ESS_INIT_T InitStruct;Initstruct.LocalPath = "D:\PRODUCT";
```

For Visual Basic,

```vbnet
Dim pInit as ESB_INIT_T
pInit.LocalPath="D:\PRODUCT"
```

A secondary way to set the client directory path is to set LocalPath to NULL. By default, Essbase then uses the ESSBASEPATH environment variable to determine the path to the CLIENT directory.

Customizing the Path to the Message Database

Essbase uses a message database file called, by default, ESSBASE.MDB. The API enables you to store the message database file with any file name and in any directory path you choose. You must use the ESSBASE.MDB file, but you can rename it. Using the MessageFile field of EX_INIT_T, you can explicitly set the location and name of the message database.

Setting the MessageFile Field of the Initialization Structure

You can change the message database file name and directory path by setting the MessageFile field in the initialization structure to point to a string indicating the full path and file name of the message database. This causes the Essbase message system to look for the path and file name specified whenever it needs to reference the text of an Essbase system message. For example, if you wanted to call the message database file PRODUCT.MDB, and install it in the C:\PRODUCT\MESSAGE directory, you would make the following change to the initialization structure:

```csharp
ESS_INIT_T InitStruct;Initstruct.MessageFile = "C:\PRODUCT\MESSAGE\PRODUCT.MDB";
```

For Visual Basic,

```vbnet
Dim pInit as ESB_INIT_T
pInit.MessageFile="C:\PRODUCT\MESSAGE\PRODUCT.MDB"
```
If you don't want to set the name and location explicitly, you can set the MessageFile field to NULL. By default, the API looks for a fully qualified file name in the ARBORMSGPATH environment variable on the user's machine. If this variable is not set, the API uses the ESSBASEPATH environment variable, appends \BIN to it, and uses that directory name to look for ESSBASE.MDB.

Setting the ARBORMSGPATH Variable

If you want to use the ARBORMSGPATH environment variable, place an ARBORMSGPATH statement in your AUTOEXEC.BAT file if you are programming on a Windows platform. Under UNIX, you set this variable in the environment script corresponding to your shell. See the Installation Notes topic for more information. To set the path and file name to C:\PRODUCT\MESSAGE\PRODUCT.MDB you would use the following statement:

```
ARBORMSGPATH = C:\PRODUCT\MESSAGE\PRODUCT.MDB
```

If you intend to use the ARBORMSGPATH or the ESSBASEPATH environment variable, set the MessageFile field in EX_INIT_T to NULL.

How Essbase Finds the Message Database

Essbase performs the following priority search to find the message database:

1. Essbase uses the directory path and file name specified in the MessageFile field of the initialization structure.
2. If the MessageFile field is set to NULL, Essbase uses the complete file and directory path specified in the ARBORMSGPATH environment variable.
3. If no ARBORMSGPATH variable is defined, Essbase uses the file name ESSBASE.MDB in the directory path specified in the ESSBASEPATH environment variable, in its BIN sub-directory.
4. If no ESSBASEPATH variable is defined, Essbase displays an error message.

Customizing the Path to the Essbase Login Help File

In Windows environments, the EsxAutoLogin() call displays a dialog box that contains a Help button. It also provides access to other dialog boxes with their own Help buttons. Clicking the Help button displays the Essbase System Login help topic (or the file specified in EX_INIT_T), which is shipped with the Oracle Essbase Spreadsheet Add-in User's Guide online help.

If you don't write your own Help file, you can simply supply the default help to your users with the product installation.

Setting the HelpFile Field of the Initialization Structure

You can specify the API help file by setting the HelpFile field in the initialization structure to a string indicating the full path and file name of the API help file. The API looks for the help file whenever the user invokes a help screen.

For example, if the API help screens are included in a file called PRODUCT.HLP in the C:\PRODUCT\HELP directory set the initialization structure to the following path:
For Visual Basic, set the path this way:
Dim pInit as ESB_INIT_T
pInit.HelpFile="C:\PRODUCT\HELP\PRODUCT.HLP"

Creating Your Own On-line Help for AutoLogin

In Windows environments, the EsxAutoLogin() call displays a dialog box that contains a Help button. It also provides access to other dialog boxes with their own Help buttons. Clicking the Help button displays the Essbase System Login help topic (or the file specified in ESX_INIT_T), which is shipped with the Oracle Essbase Spreadsheet Add-in User’s Guide online help.

You can either use the default or define your own help file. If you create your own help file, specify its path and file name in ESX_INIT_T so that the correct file appears when the user chooses the Help button.

If you plan to use EsxAutoLogin() with your own help file, then you need to include ESSHELP.H in your help project file as follows:

```
[MAP]
  #include <ESSHELP.H>
```

ESSHELP.H defines the help IDs for the dialog boxes displayed by the API. When you include ESSHELP.H, you need to create topics in your help source files with context strings corresponding to the strings in the header file. For example, you need to create a topic with a context string IDH_SYSTEM_LOGIN_DB for the Login dialog box. See ESSHELP.H for a list of context strings you should include.

If you have other context-sensitive help areas in your program, then add additional lines to the MAP section for your additional header files as follows:

```
[MAP]
  #include <ESSHELP.H>
  #include <MYHELP.H>
```

API Files You Need to Ship

For your program to work with Essbase, each client machine that runs your program must have access to the required Essbase Run-Time Client files. If the Spreadsheet Add-in or the Essbase client are already installed on a client machine, the Essbase Run-Time Client files are already available in the ESSBASEPATH\bin directory. Otherwise, you must install them as part of your product’s own installation process.

**Note:** Ensure that ESSBASEPATH is set to EPM_ORACLE_HOME\common\EssbaseRTC\11.1.2.0 (for 32 bit), or EPM_ORACLE_HOME\common\EssbaseRTC-64\11.1.2.0 (for 64 bit).
For several platforms, you need to distribute additional Oracle run-time libraries, beyond what is included in the Essbase Run-Time client directory (under \EPM\_ORACLE\_HOME\). The following platforms require access to the additional libraries:

- 32-bit Windows
- 64-bit Windows
- 32-bit Linux
- 64-bit Linux
- HP-UX Itanium 64
- Solaris x86 64
- Solaris SPARC64
- AIX 64

The additional Oracle run-time libraries are located in \%EPM\_ORACLE\_HOME\%\bin (for Windows) and \$EPM\_ORACLE\_HOME\/lib (for UNIX). In some installations, these run-time libraries may instead be located in \%ORACLE\_HOME\%\bin (for Windows) or \$ORACLE\_HOME\/lib (for UNIX).

On UNIX platforms, ensure that you preserve the symbolic links when distributing libraries.

**General Description of Files**

The Essbase API libraries can exist anywhere on the client machine or on an accessible network file server.

To ensure that the operating system can find the libraries at run time, they should either be in the same directory as your executable files, or in one or multiple directories included in the user's PATH variable (for Windows), LIBPATH (for AIX), SHLIB_PATH (for HP-UX), or LD_LIBRARY_PATH (for Solaris and Linux). See “Essbase Directory Structure” on page 37 for more information.

The ESSBASEPATH variable needs to be set so that your program can find the .mdb files. Optionally set ARBORPATH on the client side.

**Platform-by-Platform File Lists**

Users of applications programs can install the Essbase Client in order to avoid downloading specific files. Refer to the Oracle Hyperion Enterprise Performance Management System Installation and Configuration Guide for information on installing the Essbase Client.

See “API Libraries” on page 42 for list of linking library files. A full list of files is not provided, as it may be subject to change.

**API Libraries**

The files needed to link the main, outline, and grid APIs for each supported platform are listed below.
Installing Your Application Program

When you create an installation for your Essbase API program, you may wish to include the API support files as part of the installation for your application. Alternately, you can install the Essbase runtime client on the target machine and accept all the environment update options. That process installs all the files needed by the API and sets the PATH variable.

If you decide to include the Essbase API environment setup as part of the installation of your product, you must construct your installation process to install the files required by the Essbase API. The exact steps required depend on your program and on the target operating system. The following steps illustrate a typical installation process:

**Windows API Libraries (32 Bit and 64 Bit)**
- ESSAPINU.LIB
- ESSOTLNU.LIB
- ESSGAPINU.LIB

**AIX API Libraries (32 Bit and 64 Bit)**
- libessapinuS.a
- libessgapinuS.a
- libessotlnuS.a

**HP-UX API Libraries (32 Bit)**
- libessapinu.s1
- libessotlnu.s1
- libessgapinu.s1

**HP-UX API Libraries (64 Bit)**
- libessapinu.so
- libessgapinu.so
- libessotlnu.so

**Solaris API Libraries (32 Bit and 64 Bit)**
- libessapinu.so
- libessgapinu.so
- libessotlnu.so

**Red Hat Linux API Libraries**
- libessapinu.so
- libessgapinu.so
- libessotlnu.so
1. Prompt the user for the root installation drive and directory name, where root represents the name of the installation drive and directory; for example, C: \Hyperion\products \Essbase.

2. Create the \root and \root\bin directories.

3. Copy the product executable files to the \root\bin directory.

4. Copy any other product files to the \root directory or any sub-directories.

5. Prompt the user to choose a network protocol.

6. Copy or rename the appropriate Essbase network driver library to the \root\bin directory.

7. Copy the remaining library files to the \root\bin directory.

8. Copy the message database to the \root\bin directory.

9. In your operating system environment, define the ESSBASEPATH environment variable, and make it equivalent to \root\. This step is necessary only if you didn’t explicitly set the client directory path in the ESX_INIT_T structure.

10. In your operating system environment, define the ARBORMSGPATH environment variable, and make it equivalent to \root\bin\filename. This specifies the custom directory path and file name for the message database. This step is necessary only if you didn’t explicitly set the message database path in the ESX_INIT_T structure or if the message database cannot be found by using the ESSBASEPATH environment variable.

Note: These instructions are appropriate for Windows client machines. Installing on other operating systems requires slightly different steps.

Installing API Programs on Different Platforms

If you install your program on different operating system platforms, be aware that each operating system has slightly different procedures for setting the environment variables, such as PATH, ESSBASEPATH, and ARBORMSGPATH.

On Windows, the environment variables are set in the environment section of the Windows System Properties. Access the system variables through the Start > Settings > Control Panel > System > Environment tab. Adding the %ESSBASEPATH%\Bin path declaration to the path variable on Windows is equivalent to editing the PATH statement in the AUTOEXEC.BAT file on earlier Windows machines.

On UNIX systems, environment variables are typically set using login scripts for individual users. The standard practice for setting these variables on UNIX is to provide a script with your installation that sets the appropriate variables and can be included in a user’s login script by the system administrator. For more information on setting environment variables, see the Oracle Hyperion Enterprise Performance Management System Installation and Configuration Guide.

Optimizing TCP/IP Networking for API Clients

All Essbase C-API based clients communicate with the Essbase Server by means of a network layer. A request from a client C-API based application to Essbase involves opening a TCP/IP
socket at the start of the request and closing it at the end of the request. A socket is a resource managed by the operating system, and there are a fixed number of such resources - the number of which is operating-system specific. When a socket is closed, it enters a dormant state, referred to as TIME_WAIT state, for a duration that is operating-system specific and configurable. At the end of that time period, the socket can be reaped by the operating system for reuse. Whether a call to open a socket succeeds or not is a function of how rapidly the operating system is able to reap the closed sockets for reuse.

Problems may occur in cases where an API client is designed to make and conclude so many connections, so rapidly, that the fixed number of available ports (about 64,000) is at or near exhaustion. Because the used ports are still resting in TIME_WAIT state, available ports cannot be harvested fast enough by the operating system, and the result is that connections are denied or the program runs sluggishly. If a deployment expects highly concurrent processing and these symptoms are occurring, we recommend decreasing the TIME_WAIT delay for the operating system. For example, a significant performance improvement can result from decreasing the delay from four minutes to 30 seconds.

This situation can be detected by running the command `netstat` on the command prompt. The output shows the number of sockets that are in TIME_WAIT state. The higher the number, the larger the probability that certain subsequent API requests will fail.

To get around this situation, consider reducing the TIME_WAIT value on your operating system.

On Windows, the TIME_WAIT value is found in the Windows registry.

On UNIX, system tools such as `ndd` (Solaris & HPUX), `no` (AIX), and `echo` (Linux) are used to manipulate kernel parameters.

To view and adjust the TIME_WAIT value on Solaris and HP-UX,

```bash
ndd -get /dev/tcp tcp_time_wait_interval
ndd -set /dev/tcp tcp_time_wait_interval 30000
```

On AIX, the following command gives a value for all parameters:

```bash
/usr/sbin/no -a
```

Issue the following command on AIX to set TCP_TIMEWAIT state to 30 seconds (but do not adjust it if it is already below 30):

```bash
/usr/sbin/no -o tcp_timewait =2
/usr/sbin/no -o tcp_ephemeral_low = 32768
/usr/sbin/no -o tcp_ephemeral_high = 65535
```

On Linux, issue the following command to set the timeout_timewait parameter to 30 seconds:

```bash
echo 30 > /proc/sys/net/ipv4.tcp_fin_timeout
```
Introduction

This topic details implementing a simple Essbase API application, including hints and tips that might not be apparent to a new API user.

Essbase API functions are prefixed with "Ess" for the C API or "Esb" for the Visual Basic API. This discussion uses the function prefix "Esx" when discussing the operation of an API function that is available in both languages. For example, EsxLogin() refers to either EssLogin(), EsbLogin(), or both. Similarly, the prefix "ESX" indicates "ESS" or "ESB" as a prefix for a data type or constant, for example, ESX_NULL.

This tutorial refers to functional sample programs that are delivered with the API documentation. To find the sample programs, look in /Docs/Api/Samples. The C programs are in /Samples/Cexecs. The Visual Basic programs are in /Samples/VBexecs. Both the compilable source and the compiled executables are included. See Appendix A, “API Sample Programs.”

Design Environment Setup Issues

Before you can build Essbase API programs you must set a few configuration options in your design environment. This discussion focuses on Microsoft Visual C++ version 6 and Visual Basic version 6. The configuration settings within a specific development environment are set in different ways, but here are a couple of hints to assist in building an API program:

- Use byte-alignment for all API program structures. Note that byte-alignment is NOT the default setting for most C compilers!
- Alternately, if you need to link your code with code that uses another alignment setting (for example when you are using another external API), use the following #pragma directive (only when compiling with the Microsoft C/C++ compiler):
Always compile using the large memory model (for X86 platforms).

Include the header file essapi.h in all program files that use the API, and the header file essotl.h in all files that use the outline API.

Include the appropriate link library in your link process. Add the following library to your project: ESSAPIN.LIB for Windows. Add the Outline API library if your program uses the Outline API (ESSOTLN.LIB for Windows). For more information on the Essbase API libraries, refer to “API Libraries” on page 42.

**Basic Requirements**

All API programs are required to perform some core operations, such as logging in. These sections describe in detail the process of writing the shell of an application, and is meant for programmers who are new to the Essbase API:

- “The Nested Program Model” on page 48
- “Using Function Return Codes” on page 49
- “Calling API Functions” on page 50
- “Initializing the API” on page 51
- “Logging On to a Server” on page 51
- “Connecting to a Database” on page 53
- “Logging Out” on page 54
- “Terminating the API” on page 54

**The Nested Program Model**

When programming using the API, your code should adopt the nested programming model. In the nested programming model the code has calls to an initial function and a corresponding final function. The calls are arranged as a sandwich, with the code to perform some action in between as the filling. Consider the following example:

```plaintext
begin action 1
    begin action 2
        begin action 3
            perform action 3
        end action 3
    begin action 4
        perform action 4
    end action 4
end action 2
end action 1
```

The implication of this arrangement is that you should ensure that you end every action or operation that you begin. Here is a more concrete example that uses real API actions:
Initialize the API
  Login to a server
    Connect to a database
      Open a database outline
        Browse the outline
      Close the outline
    Open a report
      Modify & save the report
    Close the report
  Disconnect from a database
Logout from the server
Terminate the API

The example above illustrates the basic structure of any code that accesses the Essbase API.

**Using Function Return Codes**

One of the first things you need to know is how to handle the status codes returned by API functions. In general, a zero return code indicates successful completion and a non-zero return code indicates an error. In the latter case, the program should abort the operation in progress and return to the default state, only calling those API functions that are needed to clean up. Every time a program makes a call to the API, it should check the return code and handle it properly.

The API provides a type declaration for status return codes (ESS_STS_T) and a constant declaration (ESS_STS_NOERR). The constant declaration can be used to test the status return codes from API functions in an implementation-independent way.

```c
/* C Example of checking return value from an API function */
ESS_STS_T       sts;
if ((sts = EssSomeFunction (.....)) == ESS_STS_NOERR)
{
    do something else;
}
else
{
    process error;
}
```

```vb
' VB Example of checking return value from an API function */
Dim     sts as ESB_STS_T
if ((sts = EsbSomeFunction (.....)) == ESB_STS_NOERR)
    do something else
else
    process error
endif
```

The nested programming model is good for releasing resources if an Essbase function fails and returns an error return value. Consider the following example:

```plaintext
allocate resource 1
begin action 1
    allocate resource 2
    begin action 2
        action 2
    end action 2
```
Calling API Functions

Each API function has the prefix Ess (for C) or Esb (for Visual Basic) followed by a verb-object naming convention, for example, \texttt{EssGetDatabaseInfo()}. Some functions that relate to a specific area of the product have an additional prefix to indicate that relationship. For example, all the Outline API functions have EssOtl or EsbOtl prefixes.

All API functions take a series of arguments. The arguments are different for every function, and follow a logical sequence. The first argument to most functions is typically a handle, either an instance handle, a context handle, an outline handle, or a member handle. The term "handle" refers to an identifier used by the API to keep track of different objects in the system (just like a file handle). Different handles are returned by certain functions. Handles should then be stored in your program and passed to other API functions when required.

Handles are different in C and Visual Basic. For more information on the different types of API handles and their uses, refer to Chapter 6, “Using the C Main API” and Chapter 18, “Using the Visual Basic Main API.”

If there are any arguments to be passed in to a function, they typically come next in the sequence. Finally, if the function returns any values, the variables to store those returned values are passed in at the end of the argument list.

In the following examples, the first argument is a context handle (\texttt{hCtx}). The next two arguments (the application and database names, Sample and Basic), are passed in and the argument to be returned (the database information structure, \texttt{ESX\_DBINFO\_T}) is passed in at the end:

\begin{verbatim}
/* C Example of passing arguments to an API function */
ESS_STS_T       sts;
ESS_HCTX_T      hCtx;
ESS_PDBINFO_T   pDbInfo;
st = EssGetDatabaseInfo (hCtx, "Sample", "Basic", &pDbInfo);
if (sts == ESS_STS_NOERR)
{
    do something;
}
\end{verbatim}

\begin{verbatim}
' VB Example of passing arguments to an API function
Dim      sts as ESB_STS_T
Dim      hCtx as ESB_HCTX_T
Dim      DbInfo as ESB_DBINFO_T
sts = EsbGetDatabaseInfo (hCtx, "Sample", "Basic", DbInfo)
if (sts = ESB_STS_NOERR)
    do something
endif
\end{verbatim}

Note that in the C example, the returned argument (\texttt{pDbInfo}) is passed to the function as a double indirection (a pointer to a pointer) by passing the address of a declared structure pointer variable (using the \& operator). This variable is then assigned the address of a database information structure that is allocated internally by the API function.
In the Visual Basic example, the caller first allocates the structure (DbInfo) and passes it to the API function (implicitly by reference).

**Initializing the API**

All application programs must initialize the API (with EsxInit()) before using any other Essbase functions. The program should perform the initialization only once, preferably during the program’s startup sequence.

```c
/* C Example of initializing the API */
ESS_STS_T     sts;
ESS_INIT_T    InitStruct;
ESS_HINST_T   hInst;
/* first clear the init structure (use API defaults) */
memset (&InitStruct, 0, sizeof (ESS_INIT_T));
sts = EssInit (&InitStruct, &hInst);
```

' VB Example of initializing the API

```vb
Dim     sts as ESB_STS_T
Dim     InitStruct as ESB_INIT_T
Dim     hInst as ESB_HINST_T
sts = EsbInit (InitStruct, hInst)
```

The API default settings are appropriate for most application programs. If you need to change the settings, refer to the EsxInit and/or EsbInit for more information on setting the individual fields of the API initialization structure (“ESS_INIT_T” on page 141 and “ESB_INIT_T” on page 1180) in your program.

The instance handle (hInst) that is returned from EsxInit() should be saved within your program for subsequent API calls. This instance handle uniquely identifies your program and its associated resources to the API.

See Appendix A, “API Sample Programs.”

**Logging On to a Server**

After the API is initialized, a program must log in to an Essbase Server in order to perform any actions on that server. Generally, a login only needs to be performed when a specific action is requested by the user (typically a database connect operation). Note that a login to a server does not necessarily imply a connection to a specific application or database on that server; some administration operations do not require a connection to a particular database, and some do not even require connection to a server.

A login can be performed using EsxLogin(). For Microsoft Windows only, an encapsulated login dialog function, EsxAutoLogin(), is available. The dialog box displayed by this function is similar to the one used by the Administration Services Console and Spreadsheet Add-in interfaces. Optionally, the user can use the dialog box to select an application and a database to connect to (see “Connecting to a Database” on page 53). The user can also perform other operations, such as changing a password.

```c
/* C Example of a login using the EssLogin function */
ESS_STS_T     sts;
```
VB Example of a login using the EsbLogin function

Dim sts as ESB_STS_T
Dim hInst as ESB_HINST_T
Dim Server as ESB_SVRNAME_T
Dim Username as ESB_USERNAME_T
Dim Password as ESB_PASSWORD_T
Dim Access as ESB_ACCESS_T
Dim hCtx as ESB_HCTX_T
hInst = ESS_INVALID_HCTX
Server = "Larch"
Username = "Joe User"
Password = "secret"
hCtx = ESB_INVALID_HCTX
sts = EsbLogin (hInst, Server, Username, Password, Access, hCtx)

VB Example of a login using the EsbAutoLogin function

Dim sts as ESB_STS_T
Dim hInst as ESB_HINST_T
Dim Access as ESB_ACCESS_T
Dim hCtx as ESB_HCTX_T
hCtx = ESB_INVALID_HCTX
sts = EsbAutoLogin (hInst, ESB_NULL, ESB_NULL, ESB_NULL, ESB_NULL,
ESB_NULL, ESB_AUTO_DEFAULT, Access, hCtx)

See EssLogin, EsbLogin, EssAutoLogin, and EsbAutoLogin.

Note that, if string variables, instead of ESX_NULL, are passed to the function as the user-entered parameters, on return from the function those variables contain the values entered into the login dialog box by the user.

Your program should normally login once (at the start of a user session). However, if tying up unused server ports is a big issue, consider logging in at the start of each operation, and logging out at the end of each operation (see “Logging Out” on page 54). Note, however, that this process can slow down user response time significantly.
When using either `EsxLogin()` or `EsxAutoLogin()`, the returned login context handle (`hCtx`) should be saved within your program for subsequent API calls. The login context handle uniquely identifies that particular login to the API.

**Using Local Context Handles**

If you are performing API administrative operations (such as file operations) on the client machine, you can use a dummy login context handle to represent a local login to the API. The dummy handle can be used like a server context handle, except that most server-specific and database-specific operations cannot be performed. Use `EsxCreateLocalContext()` to create a local context handle. Consider the following example:

/* C Example of creating a local context handle */
ESS_STS_T       sts;
ESS_HINST_T     hInst;
ESS_HCTX_T      hLocalCtx = ESS_INVALID_HCTX;
sts = EssCreateLocalContext (hInst, ESS_NULL, ESS_NULL, &hLocalCtx);

' VB Example of creating a local context handle
Dim     sts as ESB_STS_T
Dim     hInst as ESB_HINST_T
Dim     hLocalCtx as ESB_HCTX_T
hLocalCtx = ESB_INVALID_HCTX
sts = EsbCreateLocalContext (hInst, ESB_NULL, ESB_NULL, hLocalCtx)

**Connecting to a Database**

Many Essbase API functions (such as server administration, security, and outline maintenance) can be performed after the program has logged in. However, many database-related functions (for example, reporting or performing calculations) require that the program connect to a specific application and database. Use `EsxSetActive()` to identify a specific Essbase database. Logging in with `EsxAutoLogin()` also allows the identification of a specific database.

Note that the user must have sufficient privileges to access the database. A list of all applications and databases to which a particular user has access is returned by `EsxLogin()`, and can be obtained using `EsxListDatabases()`.

If you connect to a database that is not running, Essbase automatically starts the database. It is not necessary to disconnect from a database. However, using the same login context handle to connect to another database will disconnect you from the original database. If you really need to be connected to two or more databases at once, your program needs to login multiple times (and manage each context handle independently).

/* C Example of connecting to a database */
ESS_STS_T       sts;
ESS_HCTX_T      hCtx;
ESS_APPNAME_T   AppName = "Sample";
ESS_DBNAME_T    DbName = "Basic";
ESS_ACCESS_T    Access;
sts = EssSetActive (hCtx, AppName, DbName, &Access);

' VB Example of connecting to a database
Dim     sts as ESB_STS_T
The user's access level to the selected database is returned by `EsbSetActive` (and by `EsxAutoLogin()`). This access level can be checked by using the security constant definitions that allow the application program to alter user options, by graying out menus, and so on.

### Logging Out

After the user completes one or more database operations and finishes with Essbase, your program should log out from the server. Logging out can be done either as a result of an explicit user request or automatically (for example, after a specific sequence of actions is complete). All active connections should also be logged out before the program terminates and exits.

It is not always necessary for the program to log out after each data access operation. Whether to log out (and so release Essbase Server ports) or remain logged in (giving faster response to successive user requests) is a design judgment call.

```c
/* C Example of logging a user out */
ESS_STS_T     sts;
ESS_HCTX_T    hCtx;
sts = EssLogout (hCtx);
hCtx = ESS_INVALID_HCTX;
```

```vbnet
' VB Example of logging a user out
Dim   sts as ESB_STS_T
Dim   hCtx as ESB_HCTX_T
sts = EsbLogout (hCtx)
hCtx = ESB_INVALID_HCTX
```

After logging out, do not use that same context handle. That will probably crash your program.

If you want to dispose of a local context handle, use `EsxDeleteLocalContext()`:

```c
/* C Example of deleting a local context handle */
ESS_STS_T     sts;
ESS_HCTX_T    hLocalCtx;
sts = EssDeleteLocalContext (&hLocalCtx);
```

```vbnet
' VB Example of deleting a local context handle
Dim   sts as ESB_STS_T
Dim   hLocalCtx as ESB_HCTX_T
sts = EsbDeleteLocalContext (hLocalCtx)
```

### Terminating the API

At the very end of its execution, your program should terminate the Essbase API by calling `EsxTerm()`, to ensure the proper release of all API resources. This function also logs out all active server connections (if they are not already explicitly logged out by your program).
/ C Example of terminating the API */
ESS_STS_T     sts;
ESS_HINST_T    hInst;
sts = EssTerm (hInst);
hInst = ESS_INVALID_HINST;

' VB Example of terminating the API
Dim     sts as ESB_STS_T
Dim     hInst as ESB_HCTX_T
sts = EsbTerm (hInst)
hInst = ESB_INVALID_HINST

After terminating the API, do not attempt to make any more calls to API functions. If you make more calls your program will probably crash.

Assembling a Program

So far in this discussion we have addressed those aspects of the API that are common to all programs. We have not addressed the operations that the program will be designed to accomplish. All programs require that you understand the nested programming model, pass arguments to and from the API functions in a consistent way, interpret the function's return codes, initialize the API, log in to a server, connect to a database, log out, and terminate. Now we need to address the real point of the program; the program needs to perform an operation of some kind.

This discussion covers the main functional groups of the C Main API. Some sections have references to the sample programs, but the sample programs do not include all areas of the API. The sample program loads data, reports the contents of the database, performs an update and a calculation, and then reports the new status of the data. Comments in the code show places where functions could be added in the future to perform additional operations.

To get some idea of the types of operations that the API can perform, take a look at the “C Main API Function Categories” on page 199 and the “Visual Basic Main API Function Categories” on page 1199. There are almost 200 functions in the C Main API divided into 20 functional groups. That means there is a wide variety of operations that the API can perform. The C Outline API (78 functions) and the Grid API (59 functions) represent additional possible complexity for an API program. The sample programs need to stay as simple as possible, so they only use a small number of functions from the C Main API, and they do not use the Outline API or the Grid API at all.

The sample programs use the Sample Basic database that is supplied with Essbase. The database is delivered empty and needs to be loaded with data. The data is delivered in a text file named CALCDAT.TXT. The sample program uses a prebuilt calc script and a prebuilt report script. The login information used by these programs (server name, application name, database name, user name, and password) are hardcoded into the program. The program displays the Login dialog box, but all the fields are filled in. The user needs only to click Okay in response to the dialog box. The server name is "LocalHost". The application name is "Sample". The database name is "Basic". The user name is "admin" and the password is "password".

Topics that discuss how to assemble a program:
Building Dimensions

Dimensions are the building blocks of the database. They define the database's structure (commonly referred to as the outline or metadata). Build the database by first assembling the necessary dimensions and each dimension's associated members. Then add the data. The outline can be developed from scratch or an existing database can be altered by adding and subtracting dimensions and members. The Sample Basic application/database is delivered with a complete outline, so it is not necessary to build the outline to run the sample programs. But it is necessary to load the data either through Oracle Essbase Administration Services, MaxL, or by running the sample program.

The API can automate the process of rebuilding dimensions dynamically from a data file or SQL source. To automate the process you must first create rules files by using Administration Services Console and then use the rules files to build the dimensions by calling EssBuildDimension or EsbBuildDimension.

These functions take the rules and data file object definitions as arguments and dynamically modify the outline on the server according to the parameters set in the rules file. They also cause any data in the database to be restructured to correspond to the new dimension structures in the outline.

The API can alter an existing database by adding and subtracting dimensions and members (using the Outline API) until the needed structure is in place. After the outline is finished load the data into the database using EssImport or EsbImport.

Editing the Outline

The database outline can be navigated and modified, using the outline API functions. These functions allow movement through the outline hierarchy, modification of member information and properties, addition and deletion of members, and so on.
Control Flow of the Outline API Functions

To begin using an outline, call `EsxOtlOpenOutline()`. If you intend to edit the outline, you should set both of the `fLock` and `fKeepTrans` arguments passed to `EsxOtlOpenOutline()` to `TRUE`. The `fLock` flag locks the outline to prevent anyone else from updating it (but not from viewing it). The `fKeepTrans` flag saves all transactions performed during the edit of the outline, for when the outline is subsequently restructured.

To start navigating the outline from the first dimension member, call `EsxOtlGetFirstMember()`. Alternately, you can locate a member by name by using `EsxOtlFindMember()` or `EsxOtlFindAlias()`. In either case, the function returns a member handle that can then be used to get or set information about that member or to get the member handles of adjacent members in the outline hierarchy.

To get information about the current member, use `EsxOtlGetMemberInfo()`, `EsxOtlGetMemberAlias()` and `EsxOtlGetMemberFormula()`. To set information for the current member, use the corresponding Set functions.

To get the parent of a member, call `EsxOtlGetParent()`. To get the first child of a member, call `EsxOtlGetChild()`. To get the siblings of a member, call `EsxOtlGetNextSibling()` or the `EsxOtlGetPrevSibling()`. To locate the next shared occurrence of a member, call `EsxOtlGetNextSharedMember()`.

To add or delete dimensions in an outline, use `EsxOtlAddDimension()` or `EsxOtlDeleteDimension()`.

To modify members in the outline hierarchy, use `EsxOtlAddMember()`, `EsxOtlDeleteMember()`, or `EsxOtlMoveMember()`.

After an outline is modified, it can be verified using `EsxOtlVerifyOutline()`, saved using `EsxOtlWriteOutline()`, and then closed using `EsxOtlCloseOutline()`.

Before any changes made to a server outline can take effect, the database must be restructured by calling `EsxOtlRestructure()`. This function applies the edits made to the outline against the old version of the outline and restructures both the outline and the associated data.
For detailed descriptions of these functions see `EssOtlOpenOutline`, `EssOtlGetMemberInfo`, `EsbOtlOpenOutline`, `EsbOtlGetMemberInfo` and each function's associated See Also lists.

**Loading Data**

After the outline dimensions are built, data can be loaded into the database through the API. The data load can be done by using a data file or a SQL source together with a rules file, by loading a free-form data file, or by loading free-form data a record at a time.

To load by using a rule with either a data file or an SQL source, use `EsxImport()`. Pass valid rules and data file object definitions as arguments. To load a free-form data file without a rules file, simply pass a NULL rules file object definition.

To load data a record at a time, call `EsxBeginUpdate()` with the Unlock argument set to FALSE, and then call `EsxSendString()` with each record of data to be loaded. This method avoids the need to lock the blocks being updated. This mechanism should be used only for batch data loading. Do not use this mechanism in multi-user situations. The lack of locking can compromise data integrity.

Note also that each record sent to the server by this method must have a terminating newline character at the end of each row.

For detailed descriptions of all these functions, see `EssImport`, `EssBeginUpdate`, `EsbImport`, and `EsbBeginUpdate`.

See “API Libraries” on page 42.

**Reporting**

Reporting in the Essbase API requires the use of a report script. The report script is sent through the API to the Essbase Server and is executed. The results are sent back through the API to the caller. The resulting output data can be displayed, printed, sent to a file, and so on. It can also be parsed and stored in an array data structure within your program.

Topics that discuss reporting:

- “Creating a Report Script” on page 58
- “Executing a Report Script” on page 61
- “Parsing the Report Output” on page 62
- “Using Report Output as a Script” on page 63
- “Using Report Output to Perform Zoom Operations” on page 64
- “Creating Tabular Format Report Output” on page 64

**Creating a Report Script**

A report script is a text string that contains the data extraction and data formatting commands required to generate output from the Essbase Server. See the *Oracle Essbase Technical*
Reference for a full description of the Report Writer language. The following principal elements generally need to be included in a report script for an API application:

- **{TABDELIMIT} command**—Include at the beginning of any report script sent to the API. It causes the output data to be returned in a format useful for parsing within a program. This command suppresses all unnecessary formatting (for example the commas used as thousand separators in numbers) and returns each member name or data value as a tab-separated token, that can be parsed and divided into cells.

- **{DECIMALS n} command**—Specifies the decimal precision of the returned numeric data (all numbers are stored internally as floating point numbers with 15 digits of precision). For example, {DECIMALS 2} gives two digits of decimal precision.

- **{INDENTGEN n} command**—Allows a program the option of indenting either parent members or child members in the rows of the report output. A negative value of \( n \) indents parent members by \( n \) spaces relative to their children. A positive value of \( n \) indents the child members by \( n \) spaces relative to their parents. A zero value of \( n \) turns off all indenting. For example, {INDENTGEN -2} indents parent members by two spaces per level (the default):

  100-10  47  41  50  138  
  100-20  44  38  49  131  
  100-30  21  14  20  55   
  100   112  93  119  324  
  200-10  25  19  23  67   
  200-20  18  14  18  50   
  200-30  17   9  14  40   
  200   60  42  55  157   
  Product 287 217 290 324  

- **{MISSINGTEXT string} command**—Converts #Missing values in the output data to a string specified by the program. For example {MISSINGTEXT "N/A"} converts any #Missing values to the string "N/A".

- **{SUPMISSING} and {SUPZERO} commands**—Eliminates unnecessary rows in the report output. The {SUPMISSING} command suppresses the output of all data rows that contain only #Missing values (that is, no actual data), and the {SUPZERO} command suppresses the output of rows that contain only zero values.

  Also useful are the {SUPBLANK} command, that suppresses both zero and #Missing values, and the {SUPALL} command, that suppresses a range of report output parameters.

- **{MISSINGTEXT string} command**—Converts #Missing values in the output data to a string specified by the program. For example {MISSINGTEXT "N/A"} converts any #Missing values to the string "N/A".

- **{OUTALTNAMES} or {OUTMBRNAMES} commands**—{OUTALTNAMES} enables you to use alias names instead of member names in the output. To revert to member names, use {OUTMBRNAMES} (the default).

- **<PAGE, <COL and <ROW commands**—Specify how the different dimensions are oriented in a report. <PAGE specifies which dimensions are in the page header (at the top of the report), and <COL and <ROW specify that dimensions are in the columns and rows, respectively. For example, <ROW(Market, Product) forces the members of the Market and Product dimensions to be displayed in that order in the rows of the report.

  Any member from any dimension can be specified in <PAGE, <COL, and <ROW. Each dimension should appear in only one of these commands, otherwise the last command takes precedence, and all dimensions should be specified (or the report layout will be unpredictable).
- **List of member names** (including any macro commands)—To extract the data required in the report by the simplest method, list the members concerned. For example, "Actual Sales Ohio Jan Feb Mar Product" produces the following report output:

<table>
<thead>
<tr>
<th></th>
<th>Actual Sales Ohio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jan</td>
<td>287</td>
</tr>
<tr>
<td>Feb</td>
<td>217</td>
</tr>
<tr>
<td>Mar</td>
<td>290</td>
</tr>
</tbody>
</table>

Alternately, you can use macro commands to specify a range of members from a dimension. Consider the following example:

- `<CHILDREN` / `<ICHILDREN`
- `<DESCENDANTS` / `<DESCENDANTS`
- `<DIMBOTTOM`
- `<ALLINSAMEDIMENSION`
- `<ONSAMELEVELAS`
- `<PARENT`
- `<ANCESTORS`

**Note:** All the above macro commands can be abbreviated, for example, `<DESC`, `<ICHILD`, and `<PAR.`

The most commonly used of the above macro commands are `<CHILD` (or `<ICHILD`) to perform a single level drill-down; `<DESC` (or `<DESCENDANTS`) to perform multilevel drill-downs, and `<DIMBOTTOM` to drill down to the lowest level members of a dimension.

For example, "Actual Sales Ohio `<ICHILD Qtr1 `<DESC Product" produces the following report output:

<table>
<thead>
<tr>
<th></th>
<th>Actual Sales Ohio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jan</td>
<td>100–10 47 41 50 138</td>
</tr>
<tr>
<td>Feb</td>
<td>100–20 44 38 49 131</td>
</tr>
<tr>
<td>Mar</td>
<td>100–30 21 14 20 55</td>
</tr>
<tr>
<td></td>
<td>100 112 93 119 324</td>
</tr>
<tr>
<td></td>
<td>200–10 25 19 23 67</td>
</tr>
<tr>
<td></td>
<td>200–20 18 14 18 50</td>
</tr>
<tr>
<td></td>
<td>200–30 17 9 14 40</td>
</tr>
<tr>
<td></td>
<td>200 60 42 55 157</td>
</tr>
<tr>
<td></td>
<td>300–10 30 19 32 81</td>
</tr>
<tr>
<td></td>
<td>300–20 24 16 25 65</td>
</tr>
<tr>
<td></td>
<td>300–30 12 7 11 30</td>
</tr>
<tr>
<td></td>
<td>300 66 42 68 176</td>
</tr>
<tr>
<td></td>
<td>400–10 30 27 32 89</td>
</tr>
<tr>
<td></td>
<td>400–20 14 10 12 36</td>
</tr>
<tr>
<td></td>
<td>400–30 5 3 4 12</td>
</tr>
<tr>
<td></td>
<td>400 49 40 48 137</td>
</tr>
<tr>
<td></td>
<td>100–20 44 38 49 131</td>
</tr>
<tr>
<td></td>
<td>200–20 18 14 18 50</td>
</tr>
<tr>
<td></td>
<td>300–30 12 7 11 30</td>
</tr>
<tr>
<td></td>
<td>Diet 74 59 78 211</td>
</tr>
</tbody>
</table>
Because member names can be numbers (for example, "100") and can contain embedded spaces (for example, "New York"), it is always a good practice to surround member names with double quotation marks when sending a report script to the API. In Release 4.0 and above, you can force member names to be output in this format by using the [QUOTEMBRNAMES] command.

- **Bang (!)**—The final element of a report script must always be a bang (!), the exclamation point character. Each script must have one (at least one) bang to cause data to be generated. If a report script appears to be executing correctly but no data is output, check to make sure that you are appending a bang to the report script.

Many of these elements are typical user-configurable parameters that are set up in advance by the user, either globally or per-report (or both).

See “API Libraries” on page 42.

## Executing a Report Script

A report script can be executed in one of three ways:

- By passing it as a string to EsxReport()
- By passing a set of strings with EsxBeginReport(), EsxSendString(), and EsxEndReport()
- By specifying a report script file with EsxReportFile()

All of these methods send the report specification to the server for processing. The output from the server is then returned to the client, and you must read all the output from that report before calling other API functions with the same context handle.

### Control Flow of the Reporting Functions

To execute a report, you can call EsxReport() and pass the report script as a single string. Set the Output argument to TRUE and the Lock argument to FALSE unless you are performing a lock and send operation.
Alternately, call `EsxBeginReport()` (setting the Output and Lock arguments as above), and then call `EsxSendString()` to send the report script a string at a time. Finally, terminate the report sequence with a call to `EsxEndReport()`.

To execute a report script from a file, call `EsxReportFile()`.

To get the report output, call `EsxGetString()` repeatedly to read the returned strings, until a null value is returned (in C, this means a null pointer value, in Visual Basic, an empty buffer is returned).


See “API Libraries” on page 42.

### Parsing the Report Output

To parse the data returned from a report, you first need to understand the report's format. If you included the `{TABDELIMIT}` command in the report script, the data comes back in the following format:

```
<token><tab><token><tab><token><tab>..........<token><newline>
<token><tab><token><tab><token><tab>..........<token><newline>
.....
<token><tab><token><tab><token><tab>..........<token><null>
```

For example, consider the following report script:

```
{SSFORMAT}{DECIMAL 0} <COL(Year) <ROW(Market) Budget Sales Cola <CHILD Qtr1 <ICHILD Market
```

This report script would normally output data that looks like the following:

<table>
<thead>
<tr>
<th></th>
<th>Budget</th>
<th>Sales</th>
<th>Cola</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jan</td>
<td>5200</td>
<td>5000</td>
<td>5300</td>
</tr>
<tr>
<td>Feb</td>
<td>5600</td>
<td>5350</td>
<td>5700</td>
</tr>
<tr>
<td>Mar</td>
<td>4250</td>
<td>4050</td>
<td>4400</td>
</tr>
<tr>
<td></td>
<td>3800</td>
<td>3450</td>
<td>3800</td>
</tr>
<tr>
<td>Market</td>
<td>18850</td>
<td>17850</td>
<td>19200</td>
</tr>
</tbody>
</table>

When you include the `{TABDELIMIT}` command, the report script outputs the data as follows:

```
Budget	Sales	Cola
Jan	Feb	Mar
East	5200	5000	5300
West	5600	5350	5700
Central	4250	4050	4400
South	3800	3450	3800
Market	18850	17850	19200
```

To parse data in this format, scan the returned string for a tab, a newline, or a null, each of which define the end of a token. The token can be one of four types:

- A member name (must begin with an alphanumeric character)
- A data value (must always begin with a number or a negative sign)
A special value, such as #Missing (must begin with a # character)

An empty cell (none of the above)

If the report is stored in an internal data structure, such as a grid or array, and the report shrinks in the number of rows or the number of columns (for example, after a zoom out operation), you might need to adjust the bounds of the new report.

The possible conflict between numeric values and numeric member names can usually be resolved by scanning any tokens that begin with a number and validating that they conform to the parameters (for example, decimal precision) of a number value. Any token that does not conform should be treated as a member name.

A more reliable method is to use the positioning of the token in the report to determine whether it is a member name or a data value. The first \( x \) rows of the report can be only member names (where \( x \) is the number of column dimensions + 1 row for the page header), and the first \( y \) columns can only be member names (where \( y \) is the number of row dimensions). If the coordinates of the token are greater than both \( x \) and \( y \), then the token is either a special value (begins with a # character), or it is a number value.

It is possible to force double quotation marks around all member names (and so avoid the identification issue) by using the <QUOTEMBRNAMES command. When you use this command, you can recognize member names by the leading double quotation marks.

It is often useful to parse the returned report output tokens into Page, Column, Row and Data areas, so they can be easily re-used in subsequent reports (see Using Report Output as a Script, below).

**Using Report Output as a Script**

The output from a report script can be used as the input to another report. The report output contains only member names and data, so you need to preface the new report with the header commands (as described above). Then append the member names output by the previous report onto the report header (not including the returned data, to avoid sending unnecessary information to the server), and execute that as a script. For example, if you first execute the following:

```
<COL("Year") <ROW("Market")
"Actual" "Sales" "Cola" <CHILD "Qtr1" <CHILD "East" !
```

The resulting report output might look something like the following:

<table>
<thead>
<tr>
<th></th>
<th>Actual</th>
<th>Sales</th>
<th>Cola</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jan</td>
<td>36</td>
<td>32</td>
<td>39</td>
</tr>
<tr>
<td>Feb</td>
<td>24</td>
<td>09</td>
<td>14</td>
</tr>
<tr>
<td>Mar</td>
<td>37</td>
<td>29</td>
<td>37</td>
</tr>
<tr>
<td>New York</td>
<td>0</td>
<td>5</td>
<td>11</td>
</tr>
<tr>
<td>New Hampshire</td>
<td>12</td>
<td>10</td>
<td>11</td>
</tr>
</tbody>
</table>
Now if you send the header from the previous report (that is, the first two lines of format commands), strip out all data from the report output, surround all member names with double quotation marks, and append a bang (!) character, you should get the following report script:

```
{TABDELIMIT}{DECIMALS 0} <PAGE("Scenario", "Measures", "Product") <COL("Year")
<ROW("Market")
"Actual" "Sales" "Cola" "Jan" "Feb" "Mar" "New York" "Massachusetts" "Florida"
"Connecticut" "New Hampshire"
!
```

This script now generates the same report that the first script generated. This method is useful when performing a series of ad-hoc operations, such as drill-downs, on a view.

Essbase inserts spaces before certain member names. What is inserted depends on the <INDENTGEN report setting. Leading spaces must be removed if the members are subsequently used as part of a report script.

**Using Report Output to Perform Zoom Operations**

To perform a simple (one-level) zoom in on a member in a view, send the output from the report that created the view as a script with the `<CHILD` (or `<ICHILD`) command before the member to be zoomed on. To perform a multilevel zoom in, use the `<DESC` or `<IDESC` commands. To perform a zoom out, use the `<PARENT` (or possibly the `<ANCESTORS`) command.

For example, consider the following report output:

```
Actual  Sales  Cola
    Jan    Feb   Mar
East   109    85   112
```

If the user chooses to drill down on East, the report script might be as follows:

```
{SSFORMAT}{DECIMALS 0} <PAGE(Scenario, Measures, Product) <COL(Year) <ROW(Market)
Actual  Sales  Cola Jan Feb Mar <ICHILD East
!
```

This script generates the following report output:

```
Actual  Sales  Cola
    Jan    Feb   Mar
New York  36    32    39
Massachusetts  24    09    14
Florida    37    29    37
Connecticut  0     5     11
New Hampshire  12    10    11
East      109    85   112
```

**Creating Tabular Format Report Output**

It is possible to force the output of a report to be in a tabular format that resembles a relational database query. The Report Writer commands to achieve this format are as follows:

- `{ROWREPEAT}` command—Causes the full list of member names to be output on each row of the report, even when there are nested groups. In the following example, Ohio is repeated on each row:

```
``
Actual Sales
Jan    Feb    Mar
Ohio    100-10 130     121     134
Ohio    100-20 118     104     123
Ohio    100-30  77      65      81

- **[SUPCOLHEADING]** command—Adding this command to the report suppresses the column headings in the report output.

- **[SUPHEADING]** command—Adding this command also suppresses the page headings in the report output. As shown in the following example:

Ohio    100-10 130     121     134
Ohio    100-20 118     104     123
Ohio    100-30  77      65      81

Also, all of the dimensions (or all but one) need to be included in the `<ROW` command in the report, to ensure that the data is returned in a fully normalized form.

### Updating Data

Updating data is the process of changing data in a view, and sending the data back to the server. When the update is in progress the user must lock the blocks that relate to the view. This ensures that no other user can change the data between the time the program retrieves that data and the time the data is written back to the database.

The sequence of actions for an update is as follows:

1. Execute a report script to lock the relevant blocks and retrieve the data to be updated
2. Change some or all of the data in the view
3. Send the data back to the server and unlock the blocks
Control Flow of the Update Functions

Lock the blocks with EsxReport() or EsxBeginReport(). Make sure to set the Lock argument passed to these functions to TRUE, locking all the blocks relating to the retrieved data. These functions can either lock the blocks and retrieve the data or just lock the blocks (if the data is either new or already current). The functions lock the blocks without retrieval by changing the value of the Output argument passed to them to TRUE or FALSE, as appropriate.

Next, allow the user to edit the data cells in the view (using whatever mechanism your product provides).

Finally, call EsxUpdate() and pass it the entire contents of the view (including the updated data values), or call EsxBeginUpdate(), and send the entire view to the server a string at a time by calling EsxSendString().

Each string sent to the server must have a newline terminating each line of the update specification.

To execute an update from a file, first lock the blocks as described above and then call EsxUpdateFile().

For more detailed descriptions see EssUpdate, EssSendString, EssBeginUpdate, EsbUpdate, EsbSendString, and EsbBeginUpdate.

See “API Libraries” on page 42.

Calculating Data

To calculate data in Essbase means to consolidate part or all of the database by using either the hierarchies and formulas defined in the database outline (the default calculation), or the formulas contained in a calc script.
Control Flow of the Calculation Functions

The default calculation is stored in the database and is executed by calling \texttt{EsxDefaultCalc}(). To get and set the script used for a default calculation, use \texttt{EsxGetDefaultCalc}() and either \texttt{EsxSetDefaultCalc}() or \texttt{EsxSetDefaultCalcFile}().

Like reports, calculations can be executed in one of three ways:

- By passing the calc specification as a string to \texttt{EsxCalc}()
- By passing a set of strings with \texttt{EsxBeginCalc()}, \texttt{EsxSendString}(), and \texttt{EsxEndCalc}()
- By specifying a calc script file with \texttt{EsxCalcFile}()

Calculations in Essbase are asynchronous operations, meaning that when the appropriate calc function is called, the API returns to the caller immediately without waiting for the calc to finish (unlike executing a report, for example). Essbase uses asynchronous calculations because a calculation can take a significant amount of time to complete (several hours is not uncommon). So, after the calculation starts, the program must check (by calling \texttt{EsxGetProcessState}()) at intervals to see if the calculation is complete.

The simplest way to check is to set up a system timer to wake up a process at short intervals (say 5-10 seconds), checking the status of the calculation. While the calculation is running you can perform any other operations within your program, but you can not make function calls to the Essbase API using the same context handle.

For detailed descriptions of all these functions see \texttt{EssCalc}, \texttt{EssBeginCalc}, \texttt{EssCalcFile}, \texttt{EsbCalc}, \texttt{EsbBeginCalc}, and \texttt{EsbCalcFile}.

See “API Libraries” on page 42.

Using Security

All the capabilities provided by Administration Services for administering security are available through the Essbase API. To fully understand the workings of the security system, refer to the Oracle Essbase Database Administrator’s Guide.

Many of the functions that use the security system require certain privileges to be available to the logged in user and return errors if an attempt is made to change security information without the correct authority. Typically, the logged in user should have Administrator or Application or Database Manager privileges, but you should be aware of possible problems if you are using the security functions and should plan for such errors, particularly during your initial testing.

To create or delete users or groups in Essbase, use \texttt{EsxCreateUser}() and \texttt{EsxDeleteUser}(). To set a user’s password, use \texttt{EsxSetPassword}(). To get a list of users on a server, use \texttt{EsxListUsers}().

To get and set a user’s or a group’s security information, call \texttt{EsxGetUser}() and \texttt{EsxSetUser}().

To get and set the list of users that are members of a group (or the list of groups to which a member belongs), call \texttt{EsxGetGroupList}() and \texttt{EsxSetGroupList}().

To get user access privileges to an application, call \texttt{EsxGetApplicationAccess}().
The security functions can return the names of all the users who have access to a named application, all the applications to which a named user has access, or the access level of a specific application-user combination. A similar function exists for databases, and corresponding Set functions exist for setting application and database access.

To get the contents of a named security filter, first call `EsxGetFilter()` then repeat calls to `EsxGetFilterRow()` (to get each row description in the filter) until a NULL string is returned. To set the contents of a filter, first call `EsxSetFilter()`, and then repeat calls to `EsxSetFilterRow()` until all rows have been sent (send a NULL row pointer to terminate the sequence).

To get a list of the named filters in a database, call `EsxListFilters()`. To get a list of users who are assigned a named filter, use `EsxGetFilterList()`.

For detailed descriptions of the security-related functions, see “C Main API Security Filter Functions” on page 211 and “VB Main API Security Filter Functions” on page 1207.

### Maintaining Applications and Databases

Apart from maintaining database outlines, there are some other administrative functions that can be performed with the API.

To get information about an application, use `EsxGetApplicationInfo()`. To get modifiable application state parameters, call `EsxGetApplicationState()` (a corresponding Set function also exists to update these parameters). Similar administrative functions exist for databases.

When using any of the application or database Set functions, call the corresponding Get function first to initialize the structure fields.

To get an application log file, call `EsxGetLogFile()`.

To get a selection of database run-time statistics, call `EsxGetDatabaseStats()`. To get or set a database note (a text string that can be viewed from the default login dialog box), use `EsxGetDatabaseNote()` and `EsxSetDatabaseNote()`.

To export part or all of a database into a text file format that can be reloaded into databases, use `EsxExport()`.

To move Essbase file objects (outlines, calc scripts, rules files, and so on) between applications or databases, use `EsxCopyObject()`. To move objects between the client and server for editing, use `EsxGetObject()` and `EsxPutObject()`.

To create an object, call `EsxCreateObject()`. To rename an object, call `EsxRenameObject()`. To delete an object, call `EsxDeleteObject()`. To list all objects of a particular type within an application or database, call `EsxListObjects()`.

For detailed descriptions of using the administration functions for database and application, see “C Main API Database Functions” on page 201, “C Main API Application Functions” on page 200, “VB Main API Database Functions” on page 1201, and “VB Main API Application Functions” on page 1200.

See “API Libraries” on page 42.
Handling Messages

The API includes a mechanism for intercepting error messages and other messages generated at the server and for displaying the appropriate messages automatically on the client program’s screen. This mechanism, although generally useful, can be turned off if desired. The API allows your program to prevent those messages from appearing and to trap them for processing within your program. You can choose which messages to display and then display the messages in a way that is consistent with your program’s internal message and error handling. This mechanism provides seamless integration of Essbase with your program.

The default message processing in Essbase is platform-dependent, but typically generates a dialog box with the log information (application and database name, username, timestamp, and so on) and the message text.

Every Essbase message has a unique identification number, a message level number, and an associated text string (that is not necessarily unique). By default, Essbase displays error messages only for serious errors, not for warnings and not for information messages.

Message Handling in C

In the C API, you can define a Custom Message Handling function and pass a pointer to that function during the initialization call, *EssInit()*. This custom function is then called when the API receives a message from the server. The custom function can examine the function return code either to process the message internally or to pass the message back to the API for default message processing. For more details see, “C Main API Message Handling” on page 85.

An example of a message handling function for Windows and C is given below:

```c
/* C Example of a message handling function */
ESS_FUNC_M ErrorHandler (ESS_PVOID_T    myCtx,
    ESS_LONG_T      MsgNum,
    ESS_USHORT_T    Level,
    ESS_STR_T       LogStr,
    ESS_STR_T       MsgStr)
{
    ESS_STS_T       sts = 0;
    ESS_STR_T       ErrorStr;
    ESS_USHORT_T    len;
    HANDLE          hMem;
    /* Only display messages of level ERROR or above */
    if (Level >= ESS_LEVEL_ERROR)
    {
        /* Calculate combined length of Log and Message strings */
        len = 3;                /* allow for end of line characters + null */
        if (LogStr != NULL)
            len += strlen (LogStr);
        if (MsgStr != NULL)
            len += strlen (MsgStr);
        /* Concatenate the strings */
        if ((hMem = GlobalAlloc (GPTR, len)) != 0)
        {
            ErrorStr = GlobalLock (hMem);
            sprintf (ErrorStr, "%s\n%s", LogStr, MsgStr);
            /* Display message in a Windows message box */
            MessageBox ((HWND)NULL, ErrorStr, "Essbase Error",
                MB_OK);
        }
    }
}
```

Assembling a Program 69
Message Handling in Visual Basic

In the Visual Basic API the message handling mechanism is slightly different. Again, you pass a parameter to the API during the initialization call, `EsbInit()`. The call initiates custom message processing (suppressing the Essbase default processing) and sets up a message stack. Then, when an error occurs in your program (indicated by a non-zero return value from an API function call), you should call an internal error handling function. That function should in turn call `EsbGetMessage()` to retrieve any messages from the stack and then display the messages in whichever way you choose. For more details, see “Visual Basic API Message Handling” on page 1123.

An example of a message handling function in Visual Basic is given below:

```vbnet
' VB Example of message handler
Dim hInst As Long
Dim hCtx As Long
Dim sts As Long
Dim Server As String * ESB_SVRNAMELEN
Dim User As String * ESB_USERNAMELEN
Dim Password As String * ESB_PASSWORDLEN
Dim Appname As String * ESB_APPNAMELEN
Dim Dbname As String * ESB_DBNAMELEN
Dim Access As Integer
Dim Init As ESB_INIT_T
' GetMessage Variables
Dim Count As Integer
Dim TestApp As String
Dim TestDb As String
Dim TestFtrName As String
Dim ErrMsg As String * 256
Dim ErrNum As Long
Dim ErrLev As Integer
ESB_TRUE = Chr$(1)
ESB_FALSE = Chr$(0)
Init.Maxhandles = 10
Init.ClientError = ESB_TRUE
Init.ErrorStack = 100
sts = EsbInit(Init, hInst)
sts = EsbAutoLogin(hInst, Server, User, Password, Appname, Dbname,
    ESB_AUTO_NOSELECT, Access, hCtx)
If sts <> 0 Then
    sts = EsbGetMessage(hInst, ErrLev, ErrNum, ErrMsg, 256)
    MsgBox ErrMsg & Chr(13) & "Program Ending"
End If
TestApp = "Sample"
TestDb = "Basic"
TestFtrName = "Anything"
' This function call should return an error and then be picked up by EsbGetMessage
sts = EsbGetFilterList(hCtx, TestApp, TestDb, TestFtrName, Count)
If sts <> 0 Then
```

Building a Simple API Program
sts = EsbGetMessage(hInst, ErrLev, ErrNum,ErrMsg, 256)
MsgBox "Program Ending" & Chr(13) & Chr(13) &ErrMsg
End If
sts = EsbLogout(hCtx)
sts = EsbTerm(hInst)
End

## Managing Memory

In the C API only, it is possible to define custom memory management functions for use within the API itself, so that you do not have any conflict between your internal memory management scheme and the memory management scheme of the API. Again, custom functions provide integration of the API into your program.

First, you need to write three functions within your code:

- A memory allocation function.
- A memory freeing function.
- A memory reallocation function.

Next, you need to pass pointers to these three functions to the API during the initialization call, EssInit(). The functions are then used within the API whenever the API needs to allocate, free, or reallocate a memory buffer. Any items that are allocated within the API and returned to your program are guaranteed to have used these functions, so you can reallocate or free them without any possibility of a memory corruption or violation.

For more information on using custom memory management with the API, see “Using Memory in C Programs” on page 84.
General Programming Considerations

Only Unicode mode clients can fully work with Unicode mode applications. In general, writing Essbase applications programs for Unicode must take into account the mode of the client and of the server. The assumption made for this discussion is that the Essbase Server is fully Unicode enabled, that is, the Essbase Server is the latest version.

There are three basic scenarios depicting three types of client communicating with the Unicode enabled Essbase Server. The three client types:

- Non-Unicode client program communicating with the Unicode server
- Unicode-enabled client in non-Unicode mode communicating with the Unicode server.
- Unicode-enabled client communicating with the Unicode server.

Unicode enabled programs in non-Unicode mode can access all data on Unicode servers, but can not change the database outlines for Unicode mode applications. Only Unicode enabled client programs operating in Unicode mode have full access to both the data and the database outlines on Unicode enabled servers.

Defining Unicode Mode Client Programs

Only Unicode-mode client programs can communicate with the server using UTF-8 encoded data. To initialize a Unicode mode client program, use the usApiType field of the ESS_INIT_T structure passed to EssInit(). This field has two possible values: ESS_API_NONUNICODE and ESS_API_UTF8.

This API initialization function is the only place to specify the mode of an application program. This topic contains the following sections:
Non-Unicode Clients

The non-unicode clients are the older clients were built to work with previous version of the Essbase API. These clients deal entirely in short strings and non-unicode encoding. These older clients cannot deal with the longer strings and are, therefore, restricted to dealing with non-Unicode-enabled applications.

This type of client can not edit the outlines or rules files on a Unicode mode server.

A Unicode-enabled server can communicate in non-Unicode mode with non-Unicode clients.

The non-Unicode clients can edit outlines and rules files while not connected to a server. However, encoding can be an issue for non-Unicode clients editing rules files and for Unicode clients editing rules files and outlines.

When editing rules files or outlines a Unicode mode server, the user can select the format of the output file or let it default to being the same as the input file. The permissible output file formats are:

- Non-Unicode format - short strings and non-Unicode encoding
- Unicode format - long strings and UTF-8 encoding

The files are edited internally in non-Unicode encoding by non-Unicode clients and in Unicode encoding by Unicode clients.

If the input file is to be converted from Unicode format to non-Unicode format, but cannot be converted because it contains strings that are too long, then the conversion is aborted and a diagnostic is returned to the user.

Unicode-enabled Clients in Non-Unicode Mode

Unicode-enabled clients are built with the include files and DLLs of Unicode-enabled Essbase, but communicate with the API in native encoding. The API does not support placing client API DLLs from Unicode-enabled Essbase onto a client that is built with include files from non-Unicode Essbase. Clients must be built with the Unicode-enabled Essbase include files in order to run with the new DLLs.

Unicode-enabled clients in non-Unicode mode cannot edit the outlines or rules files on a Unicode mode server.

To work with the Unicode-enabled include files and DLLs, a client must support the longer maximum string lengths. For some clients, this may be as simple as recompiling with the Unicode-enabled Essbase include files that define the longer maximum lengths.
For other clients, supporting the longer maximum lengths may require code changes. For instance, the Binary Spreadsheet Table (BST) used by Spreadsheet Add-in uses a single byte for storing the member name byte length. Because one byte is not enough to hold the new maximum byte length for member names (320 bytes), the design of the BST must be changed to allow the spreadsheet client to support the longer maximum lengths.

**Unicode-enabled Clients in Unicode Mode**

Unicode clients are built with Unicode-enabled Essbase and communicate with the API in UTF-8.

To run as a Unicode client, a client must handle long maximum string lengths, as described in the previous subsection on new native clients. In addition, the client must communicate with the API in UTF-8.

If a client is written in Java, the conversion may be easier than if the client is written in another language. However, in either case, the changes are likely to be substantial. For instance, the client code must communicate with the operation system in non-Unicode encoding while communicating with the Essbase API in Unicode mode.

**Specifying Unicode Mode**

The initialization structure, ESS_INIT_T, is the only place that you can specify the Unicode-related mode of the client program. If nothing is specified, the program operates in non-Unicode mode. Use the `usApiType` field to specify the mode.

**Specifying the Byte Order Encoding**

Unicode clients using the C Main API to communicate with Unicode-enabled Essbase applications must send the UTF-8 encoded Unicode byte order mark (BOM) in the text stream immediately after calling any of the following functions:

- `EssBeginReport`
- `EssBeginUpdate`
- `EssBeginDataload`
- `EssBeginDataloadASO`
- `EssBeginDataloadEx`
- `EssBeginStreamBuildDim`
- `EssBeginCalc`

To send the BOM, use `EssSendString`, as shown in the following examples:

```c
void ESS_BeginUpdate()
{
    ESS_STS_T sts = ESS_STS_NOERR;
```
ESS_BOOL_T Store;
ESS_BOOL_T Unlock;
ESS_STR_T query = "";
/* Begin Update */
Store = ESS_TRUE;
Unlock = ESS_FALSE;
sts = EssBeginUpdate (hCtx, Store, Unlock);
printf("EssBeginUpdate sts: %ld\n",sts);
/* Send update specification */
//String with BOM characters
query = "\xEF\xBB\xBF 'marché' 'New York' 'Actual' 'Sales' '100-10' 5";
if(!sts)
    sts = EssSendString(hCtx, query);
/* End Update */
if(!sts)
    sts = EssEndUpdate(hCtx);
}
void ESS_BeginReport()
{
    ESS_STS_T sts = ESS_STS_NOERR;
    ESS_STR_T rString = ESS_NULL;
    ESS_STR_T query = ESS_NULL;
    sts = EssBeginReport (hCtx, ESS_TRUE, ESS_FALSE);
    printf("EssBeginReport sts: %ld\n",sts);
    if(!sts)
    {
        //String with BOM characters
        query = "\xEF\xBB\xBF 'New York' 'Actual' 'Sales' '100-10' 'marché' 'Jan' !";
        sts = EssSendString(hCtx, query);
    }
    if(!sts)
        sts = EssEndReport (hCtx);
    if(!sts)
        sts = EssGetString(hCtx,&rString);
    while (!(!sts) && (rString != NULL))
    {
        printf("%s", rString);
        EssFree (hInst, rString);
        sts = EssGetString (hCtx, &rString);
    }
    printf("\n");
}

**Unicode Mode and Essbase Server**

Essbase Server allows the creation of Unicode mode applications, or migration of non-Unicode mode applications to Unicode mode, only when it is in Unicode mode.

For more information, see “C Main API Unicode Mode Functions” on page 215.
Unicode Outlines

For functions related to working with Unicode mode outlines, see “C Outline API Unicode Mode Functions” on page 718.

Grid API

To initialize a Unicode mode client program utilizing the Grid API, use the usApiType field of the ESSG_INIT_T structure which is passed to EssGInit(). Additionally, ESSG_DATA_T has additional values for the usType field to work in Unicode mode.
Part II

C Main API

In C Main API:

- Using the C Main API
- C Main API Declarations
- C Main API Functions
Using the Instance Handle in an Application

An instance handle (similar in concept to a file handle) represents a program’s access to the API, and distinguishes the program-specific resources and settings used within the API. This identification is necessary for dynamic shared libraries, which may be accessed by several different programs simultaneously. When a program initializes the API by calling EssInit(), an instance handle is returned.

Instance handles may be passed to other programs, child processes, or threads, which can then log in independently of the original using the same API resources and settings. Make sure that
all programs, processes or threads using the same instance handle log out before they can terminate the API.

**Note:** A thread may require its own instance handle (\textit{phInstance}) to avoid overwriting another thread's networking status information.

### C Main API Context Handles

A context handle represents a single, valid login by a user onto the system. A successful call to \texttt{EssLogin()} returns a context handle, which can be passed to other API calls which require a context handle as an argument:

- **Using context handles in an application**—Context handles are defined as type `ESS_HCTX_T` in C programs. In general, a context handle is valid for as long as the user remains logged in to that server (that is, until after a successful \texttt{EssLogout()} call). However, in case such as a server shutdown, a context handle can become invalid. Your program should therefore provide some way for the user to log back in during a session (for example, through a menu option or function key).

**Note:** A context handle is specific to an instance of the API, and contains an implied reference to the resources and settings for the appropriate instance.

- **Multiple context handles**—A single instance of an API program may make multiple calls to \texttt{EssLogin()}, using the same user name or different user names on one or more Essbase servers. Each call to \texttt{EssLogin()} returns a unique context handle, and your program must keep track of each context handle returned. You may have up to 255 context handles per client application in use simultaneously, but if a program performs all its processing on a single server, in general it is easier to use only one context handle and to switch between different applications and/or databases as required, using either the \texttt{EssSetActive()} function or the \texttt{EssAutoLogin()} function.

- **Sharing context handles**—In general, it is not advisable to share context handles between multiple programs, processes, or threads, unless such use is guaranteed to be exclusive. A better approach is to use the same instance handle and log in each process or thread separately. Essbase ensures that multiple logins using the same user name on the same server will only occupy one port on that server.

- **Local context handles**—Operations on local objects and files (on the client) can use a local context handle (see Using Local Context Handles).

See also Local Contexts.

### C Main API File Objects

An Essbase object is simply a file (in 8 by 3 alphanumeric character format) Essbase uses, such as a database outline, a calc script, or other data. Essbase has an object system which allows you to refer to such files through the API simply by the name, the file type, and the application and...
database with which they are associated. This allows objects to be manipulated independently of the underlying file system (which may vary between different platforms and implementations of Essbase).

Objects can reside on any Essbase Server or client, and can be copied between them. A locking mechanism on the server controls access to objects, so that users with sufficient privilege can lock server objects and copy them to the client (using the `EssGetObject()` function), edit them, and save them back to the server (using the `EssPutObject()` function). Server objects can also be opened without locking for read-only access, but then cannot be saved back to the server. A user can also create or edit objects on client workstations for their personal use, or save them to the server for other users to share.

### Accessing Objects

When you access objects through the API, the object name refers to the file name of the object (without an extension). The object types are declared in the API header file in the form `ESS_OBJTYPE_`xxx (where xxx represents the specific type, as in `ESS_OBJTYPE_REPORT`). Most objects are associated with an application and database, but some objects such as calc scripts and rules files can be stored at the application level and used with any database within the application.

Database outline files are different from other objects, and cannot be deleted, renamed, copied, or saved using the API.

Server object files are physically located in the corresponding application or database sub-directory. However, it is not generally advisable to manipulate server object files directly. Always use the appropriate API functions to copy the files locally.

Client object files are also stored by default in application and database sub-directories of the directory specified by the `LocalPath` setting of `ESS_INIT_T`. You can freely manipulate and edit these files, but you should ensure your program is well-behaved when locking and unlocking server objects which are being edited on the client (always lock an object before editing and unlock it afterwards, whether or not changes are saved).

You can bypass the client object system and go directly to the file system by setting the application and database to NULL. This makes the object field the entire path.

### Local Contexts

If you intend to access file objects on a client machine through the API, you need to create a local context handle for the API object functions to use. To create a local context, use the `EssCreateLocalContext` function, which returns a context handle. This handle can be passed to any of the object API functions instead of a login context handle, and causes the API to perform the requested operation on the local client object system instead of the server. You only need to create a local context once, immediately after your program first initializes the API.

If you create a local context, your program should clean up by calling the `EssDeleteLocalContext` function before terminating the API.
Using Memory in C Programs

All programs perform some form of memory allocation. The Essbase API allocates memory internally, some of which is returned in the form of pointers to the calling program. The calling program can also allocate memory, which is passed as pointers to the API. To avoid potential conflicts between different memory management schemes, the API provides two mechanisms for integrating the memory management in your application:

- Use the API's memory management scheme in your application
- Customize the API to use your application's memory management scheme internally

Using the C API's Memory Management Scheme

The API provides a set of memory management functions, `EssAlloc()`, `EssRealloc()`, and `EssFree()`. These functions (plus all internal API memory allocations) call memory allocation routines pointed to by the `AllocFunc`, `ReallocFunc`, and `FreeFunc` fields of the `ESS_INIT_T` initialization structure. If you pass NULLs into these fields, you use the default allocation routines supplied with the API, which use native memory application routines appropriate to the target platform.

The native memory allocation routines called by all platforms call the C standard library calls `malloc()`, `realloc()`, and `free()`. The C standard library calls accommodate the operation of the Outline API, which uses many small allocations of memory during normal usage. Unlike `GlobalRealloc()`, `realloc()` does not initialize new buffer areas to NULLs.

**Note:** If you are using a compiler for an Intel X86-based Microsoft Windows platform, remember that the API exclusively uses the large memory model.

Customizing the Memory Management Scheme

If you do not want to call the API's memory management functions, or you want to ensure that the same allocation scheme is used consistently throughout your application, you can define your own set of memory management functions for the API to use. To do this, you can write your own custom functions to allocate, reallocate, and free memory, and make your functions available to the API. Usually these functions internally call the corresponding memory management functions used within your application.

Defining Custom Memory Management Functions in C Programs

To define your own custom memory management functions in a program, you write the functions and set the `AllocFunc`, `ReallocFunc`, and `FreeFunc` fields in the API initialization structure to point to your custom function before calling `EssInit()`. You can use any names you wish for these functions and their arguments, but you must use the following form to declare them:

```c
ESS_FUNC_M CustomAlloc (ESS_SIZE_T BufSize, ESS_PPVOID_T ppBuffer);
ESS_FUNC_M CustomRealloc (ESS_SIZE_T BufSize, ESS_PPVOID_T ppBuffer);
ESS_FUNC_M CustomFree (ESS_PVOID_T pBuffer);
```

In this code, the fields are defined as follows:
The **BufSize** argument is the minimum size of memory buffer to allocate or reallocate.

The **ppBuffer** argument is the address of a memory pointer to receive the allocated or reallocated buffer's address.

The **pBuffer** argument is the address of a memory buffer to free. These functions return zero (0) for success and non-zero for failure.

Pointers to these three functions should then be assigned to the AllocFunc, ReallocFunc, and FreeFunc fields of the initialization structure before it is passed to the *EssInit()* function (see “Initializing the C Main API” on page 90).

**Note:** If you decide to define your own custom memory management functions, you must create and assign functions for *all three* structure fields.

After you have defined your own custom memory management functions, you cannot use the default API memory management within that application, as any calls made to the Essbase memory management API functions, *EssAlloc()*, *EssRealloc()*, and *EssFree()*, from within your code will automatically invoke the equivalent custom functions you defined. However, any other applications simultaneously using the API will not be affected; each application which calls *EssInit()* can independently choose whether to define its own custom functions or use the default ones.

**Note:** You should not attempt to call any Essbase API functions from within your custom message function, with the exception of the memory management API functions, *EssAlloc()*, *EssRealloc()*, and *EssFree()*. 

## C Main API Message Handling

When your program calls the API, system messages and error messages are generated. Some of those messages are returned by the Essbase Server, and others are internal to the API. Your program must process these messages in some way, and if there is an error which causes the operation in progress to abort, the user may need to be informed.

This section explains the API’s message handling scheme, and then shows what C developers can do to implement custom message processing in their programs:

- “How the Essbase C Main API Handles Messages” on page 85
- “Defining a Custom Message Function in C Programs” on page 86

## How the Essbase C Main API Handles Messages

The following message levels are supported in Essbase:

- Information messages (for information only)
- Warning messages (operation will continue)
- Error messages (operation aborted)
Serious errors (operation aborted-system is unstable)

Fatal errors (operation aborted-system is halting)

When your program uses Essbase API default message handling, all messages of level Error or higher (Serious or Fatal) are displayed on the current application screen.

Defining a Custom Message Function in C Programs

The C API allows you to supply a custom message handling function which you can use to trap error messages before they are processed by the API. You may want to code a custom message handling function, either to trap particular error conditions, or to ensure uniform processing and display of all user messages throughout your program. If you choose not to supply a custom message function, all message processing is handled by the API default message handler.

To define a custom message function in a program, you must write the function and set the `MessageFunc` field in the API initialization structure to point to your custom function before calling `EssInit()`.

Coding the Custom Message Handling Function

You can use any name you wish for this function and its arguments, but it must be declared in the following form:

```c
ESS_FUNC_M CustomMessage ( 
ESS_PVOID_T UserContext,     /* user context pointer */
ESS_LONG_T MessageNumber,   /* Essbase message number */
ESS_USHORT_T Level,           /* message level */
ESS_STR_T LogString,       /* message log string */
ESS_STR_T MessageString    /* message string */
);
```

In this code, the fields are defined as follows:

- The `UserContext` argument is a copy of the pointer passed in the `UserContext` field of the initialization structure to the `EssInit()` function during API initialization (see “Initializing the C Main API” on page 90). You can use this pointer to contain any application-specific context information which is required during custom message processing, but typically it is used to pass a structure containing state information for your program.

- The `MessageNumber` argument is used to trap messages returned by specific error conditions (individual error message codes are defined in the header file `esserror.h`).

- The `Level` argument is used to trap messages based on the message level, which denotes whether the message is an information, warning, or error message.

- The `LogString` argument receives the server log entry information as a string. It passes strings of the form:

  `[Date & Time] Server/Application/Database/Username/Thread/Message#`

  For example:

  `[Fri Feb 04 11:51:18 1994] Elm/Sample/Basic/Admin//1012550`
- The **MessageString** argument contains the message text as a string. It passes the complete message text, for example:

```
Total Calc Elapsed Time : [46] seconds
```

- The default API message handler displays both the log string and the message string on successive lines, (either within the message dialog, or just written to the stdout stream). For example:

```
[Fri Feb 04 11:51:18 1994]Elm/Sample/Basic/Admin/\1012550 Total Calc Elapsed Time :
```

### Setting the MessageFunc Field to Point to Your Function

Pointers to the custom message function must be assigned to the **MessageFunc** field of the initialization structure passed to the **EssInit()** function (see “Initializing the C Main API” on page 90).

### Using a Custom Function to Control Message Processing

The custom message function is called before an Essbase Server returns a message or the Essbase API returns an error. When the function is called, the arguments passed to it contain the message number, message level, log string, and error string for that particular message. For each message, the function can use these argument values to choose whether to process the message, ignore it, or return it to the API for default processing:

- **What the return code means to the API**—A return value of zero denotes that the function processed the message successfully and that no further action needs to be taken by the API. If the return code is non-zero, the message is passed to the default API message handling function for further processing and display. To have your program ignore a message, simply return a zero from the custom message function.

  **Note:** The API automatically frees the log and message strings when it has finished processing the message. Do NOT attempt to free them within your code.

- **Determining which return code your function should generate**—To determine which return code to generate, you can code the custom message function to check the **MessageNumber** argument, and/or the **Level** argument. For example, a program might ignore all information messages, and possibly also any warning messages (you can make this a user-definable setting) by testing the **Level** argument against the appropriate constant defined in ESSAPI.H (for example, **ESS_LEVEL_WARNING**), and returning zero if the value is equal to or below the required value. For other messages the function should either process them internally and return a zero value, or return a non-zero value to ensure that they are processed by the default API message handler.

  **Note:** You should not attempt to call any Essbase API functions from within your custom message function, with the exception of the memory management API functions, **EssAlloc()**, **EssRealloc()**, and **EssFree()**.

If you define your own custom message handling function, any other applications simultaneously using the API will not be affected; each application which calls **EssInit()** can
independently choose whether to define its own custom message function or just use the default message handler.

**Choosing a Network Protocol**

Essbase supports several different network protocols and different network vendor implementations by providing a number of different Essbase network drivers. The driver you need to install depends on the exact hardware, operating system, and network platform of the client machine, and on the Essbase Server machine it is connecting to.

You need to determine the required network configuration and install the appropriate driver file.

**Calling C Main API Functions**

This section describes calling API functions, using instance and context handles, and handling return code.

**Function Declarations**

The API uses the `ESS_FUNC_M` macro to declare C API functions. This declares them to be of type unsigned long for all supported platforms. You must also use this macro to declare any custom functions which you pass to the API, such as custom memory management or message handling functions.

**Passing the Instance Handle or Context Handle**

You must pass the instance handle returned by the initial call to `EssInit()` in calls to `EssLogin()` or `EssTerm()`. You must pass the context handle returned by `EssLogin()` in any function calls associated with a specific login.

**Handling the Return Code**

All Essbase API functions return a status code of type `ESS_STS_T`. A return code of zero indicates that the function was executed successfully, and a non-zero value indicates an error condition. A full list of error return constants is contained in the header file `esserror.h`. The corresponding message text is in `messages.txt`.

**Note:** You should always check the return code from any Essbase API function. If the return code is non-zero, any pointers or values returned by the function are undefined.

**Internal Message Handling**

Essbase uses an internal message handling function for non-custom message handling. If an error event is encountered under a 32-bit Windows system, a text error message is generated.
Typical C Main API Task Sequence

The API requires that your program call certain functions before others. The basic ordering rules are:

- A program must call `EssInit()` before calling any other API functions.
- A program must call `EssLogin()` or `EssAutoLogin()` before calling any API functions which require a context handle argument (most API functions). Additionally, if you need to create a local context for API object functions to use, you must call `EssCreateLocalContext()` before calling any API functions requiring a context handle argument.
- Some API functions require an active application and database to be set. This is done by having the program call `EssSetActive()` or `EssAutoLogin()` before they are called.
- C programs cannot call any functions except memory management functions from within custom message handling functions.
- C programs cannot not call any API functions from within custom memory management functions.
- A program must not pass a context handle to any API functions after calling `EssLogout()` for that handle.
- A program must not call any API functions after calling `EssTerm()`.

This is the typical order of operations for a simple API application:

1. Create and initialize an `ESS_INIT_T` structure.
2. Initialize the API by calling `EssInit()`.
3. Allocate any local static or global structures.
4. Log in to the required server by calling `EssLogin()` or `EssAutoLogin()`.
5. Select an active application and database by calling `EssSetActive()` or `EssAutoLogin()`.
6. Retrieve (or lock) data by calling `EssReport()` or related functions.
7. Update data by calling `EssUpdate()` or related functions.
8. Recalculate the database by calling `EssCalc()` or related functions.
9. Produce reports against the data by calling `EssReport()` or related functions.
10. Log out from the server by calling `EssLogout()`.
11. Free any local static or global structures.
12. Terminate the API by calling `EssTerm()`.
**Initializing the C Main API**

A program *must* initialize the API by calling the `EssInit()` function before calling any other Essbase API functions. `EssInit()` initializes all internal API state variables, and also allows you to tailor the API to your program’s requirements.

The calling program must pass the `EssInit()` function an initialization structure. This structure is defined in ESSAPI.H as type “ESS_INIT_T” on page 141. It contains a series of fields which are used to customize the API and set up certain API defaults. You must declare an instance of this structure and initialize the relevant fields before calling `EssInit()`.

The `EssInit()` function returns an instance handle, which should then be passed as an argument to the API login function.

**Declaring the Initialization Structure**

The initialization structure passed to `EssInit()` can usually be declared as a local (i.e. stack) variable in the calling function, as it is usually not required once it has been passed to `EssInit()`. Alternatively, you can allocate the structure before calling `EssInit()`, then free it after returning.

If the initialization structure points to custom memory management functions in the initialization call, make sure your program frees the structure using the correct memory allocation scheme.

If any of the fields of the initialization structure are set to zero values or NULL pointers, the API will use the internal default values for those fields.

It is a good idea to clear out all structures (set to 0) before setting fields and calling the API function.

**Logging in to an Essbase Server**

In general, the first thing your program should do after calling `EssInit()` is to prompt the user for a server name, user name, and password (or use predefined defaults), then attempt to log in to that server by calling `EssLogin()`. Alternatively, use the encapsulated login function, `EssAutoLogin()`. If this call is successful, then the returned context handle should be stored and used for all subsequent API calls.

**Selecting an Active Application and Database**

In addition to the context handle, the login functions also return a list of the applications and databases to which the logged in user has access (a program can obtain this list at any time by calling the `EssListDatabases()` function. The program allows the user to select a specific application and database by calling the `EssSetActive()` function.

If `EssAutoLogin()` is used to log in, it can optionally set the active application and database.

To get information about an Essbase application (e.g. whether or not it is already loaded), call the `EssGetApplicationState()` or `EssGetApplicationInfo()` functions. To get information about
a specific database, call the `EssGetDatabaseState()` or `EssGetDatabaseInfo()` functions. You can call these functions before setting the active application and database.

## Retrieving and Updating Data

### Retrieving Data

To retrieve data from an Essbase database, either for reporting or for subsequent updating, your program needs to use a report specification. Report specifications can be in the form of a single text string (if it is less than 32 KB in length), a series of text strings, or a file. Report files can reside either on the client machine, or on the Essbase Server.

- **Sending a report specification as a single string**—To send a report specification as a single string, have the program call `EssReport()` passing the entire report string, not greater than 32 KB long, as an argument. If the `Output` flag is set to TRUE in the call to `EssReport()`, the program must also read the returned report data by calling `EssGetString()` repeatedly until a NULL string is returned. The returned data can then be displayed, written to a file, or printed, as required.

- **Sending a report specification as a series of strings**—To send a report specification as a series of strings, first call `EssBeginReport()`, then call `EssSendString()` repeatedly to send each string in the report specification (note that in Windows, each individual string must not be greater than 32 KB long). Finally, terminate the report specification by calling `EssEndReport()`. If the `Output` flag is set to TRUE in the call to `EssBeginReport()`, the program must also read the returned report data by calling `EssGetString()` repeatedly until a NULL string is returned. The returned data can then be displayed, written to a file, or printed, as required.

- **Sending a file as a report specification**—To send a file as a report specification, use the `EssReportFile()` function, passing the report file name. If the `Output` flag is set to TRUE in the call to `EssReportFile()`, the program must also read the returned report data by calling `EssGetString()` repeatedly until a NULL string is returned. The returned data can then be displayed, written to a file, or printed, as required.

### Updating Data

To update data in the database, you should first lock the blocks in the database which you are going to update.

To lock database blocks, select one method:

- **Send a report specification as described above, with the Output flag set to TRUE and Lock flag also set to TRUE.** The data output by this report can be modified, then sent back to the database as an update.

- **Alternatively, if there is new or modified data ready to be loaded,** a program can first use it as a report specification to lock the data blocks by setting the `Output` flag to `FALSE` and setting the `Lock` flag to `TRUE` when calling the appropriate report function.
The database can be updated either from a single string, a series of strings, or a file. Update data files can reside either on the client machine, or on the Essbase Server:

- **Sending update data as a single string**—To send an update as a single string, call `EssUpdate()` passing the entire string as an argument. (Note that in MS-Windows, the string must not be greater than 32 KB long). Set the `Store` flag to `TRUE` in the call to `EssUpdate()` so that the database will be updated. If the `Unlock` flag is also set to `TRUE`, any locked data blocks in the database will be unlocked once the data is updated, to allow other users to update those blocks.

- **Sending update data as a series of strings**—To send an update as a series of strings, first call `EssBeginUpdate()`, then call `EssSendString()` repeatedly to send all the data (note that in MS-Windows, each individual data string must not be greater than 32 KB long). Finally, terminate the update by calling `EssEndUpdate()`. Set the `Store` flag to `TRUE` in the call to `EssUpdate()` so that the database will be updated. If the `Unlock` flag is also set to `TRUE`, any locked data blocks in the database will be unlocked once the data is updated.

- **Sending update data as a file**—To send an update as a file, use the `EssUpdateFile()` function, passing the data file name. Set the `Store` flag to `TRUE` in the call to `EssUpdate()` so that the database will be updated. If the `Unlock` flag is also set to `TRUE`, any locked data blocks in the database will be unlocked once the data is updated.

---

## Recalculating the Database

After updating any data in the database it is essential to perform a recalculation to ensure that the consolidated totals are correct. To recalculate a database, you can either perform the default calculation, or send a specific calculation script. You can also set a calculation script to be the default calc script. Calc scripts can be sent either as a single string, a series of strings, or a file. Calc script files can reside either on the client machine, or on the Essbase Server.

### Sending a Calc Script as a Single String

To send a calc script as a single string, call `EssCalc()` passing the entire string as an argument (note that in MS-Windows, the string must not be greater than 32 KB long). Set the `Calculate` flag to `TRUE` in the call to `EssCalc()` so that the calc script will be executed. You will then need to check on the progress of the calculation at regular intervals.

### Sending a Calc Script as a Series of Strings

To send a calc script as a series of strings, first call `EssBeginCalc()`, then call `EssSendString()` repeatedly to send all the strings in the calc script (note that in MS-Windows, each individual string must not be greater than 32 KB long). Finally, terminate the script by calling `EssEndCalc()`. Set the `Calculate` flag to `TRUE` in the call to `EssBeginCalc()` so that the database will be recalculated. You will then need to check on the progress of the calculation at regular intervals (see "Checking the Progress of Calculations").

### Sending a Calc Script as a File

To send a calc script as a file, use the `EssCalcFile()` function, passing the calc script file name. Set the `Calculate` flag to `TRUE` in the call to `EssCalcFile()` so that the database will be
recalculated. You will then need to check on the progress of the calculation at regular intervals (see "Checking the Progress of Calculations").

Using the Default Calc Script

To recalculate a database using the current default calc script, use the EssDefaultCalc() function. To set the default calc script for a database, use EssSetDefaultCalc(), passing the calc script as a single string. To set the default calc script from a file, use the EssSetDefaultCalcFile() function, passing the calc script file name. Use EssGetProcessState() to determine when the calculation is finished (see "Checking the Progress of Calculations").

Checking the Progress of Calculations

After a database calculation is started, check the progress of the calculation at regular intervals (five seconds is recommended) by calling the EssGetProcessState() function. This function returns a structure indicating the calculation state. Call EssGetProcessState() until it indicates that the calculation is finished or that an error has occurred. You may also cancel a calculation in progress with the EssCancelProcess() function.

Caution! While a calculation is in progress, do not attempt to call any API functions other than EssGetProcessState() or EssCancelProcess() using the same context handle, until the calc operation has completed successfully or has been canceled. After EssGetProcessState() indicates the calc has finished, your program may continue performing other API operations with that context handle.

Logging Out from the Essbase Server and Terminating the C Main API

When all database operations are complete, the application should first log out by calling EssLogout(). This frees up any internal resources reserved within the database, and may also free the login port on the server for use by another user.

When an application program is about to terminate, it should call the EssTerm() function, passing the instance handle which was returned from the original call to EssInit(). This releases all resources used by the Essbase API. After calling this function, no other API calls can be made, unless EssInit() is called again to reinitialize the API.

C Main API Common Problems and Solutions

The Essbase API gives you unrestricted access to many of the same functions that Essbase Administration Server and MaxL use.

This section is a quick reference to help you in identifying and solving the most common problems.
<table>
<thead>
<tr>
<th>Problem</th>
<th>Solution</th>
</tr>
</thead>
</table>
| Your program is generating protection faults when allocating or freeing memory. | Here are some things C programmers can check:  
- Check that any memory returned from the API is being freed using the `EssFree()` function.  
- Check the declared indirection level of any pointers being passed to the API.  
- Use a memory checking program, such as Bounds Checker™ or Purify™, to determine the affected module.  

Even if the errors are occurring when accessing memory not used by Essbase, there may be some interference between the Essbase memory management scheme and your own. You might consider defining your own custom memory management functions. |
| Your program generates an Essbase error when calling an API function.   | Most of the Essbase error messages are self-explanatory, and it should be fairly obvious where the problem lies. However a couple of common errors to watch out for are (%n indicates a message argument which is replaced by a context-specific string):  
- "NULL argument (%1) passed to ESSAPI function %2". This message indicates that one or more arguments passed to the API function %2 were NULL. The %1 indicates the number of the first null argument (1-based).  
- "Invalid call sequence in ESSAPI function %1". This message indicates that you have made a call to an API function (%1) when another function call was required. For example, if you have executed a report function, such as `EssReport()`, make sure that you call `EssGetString()` repeatedly until a NULL string is returned; or if you have executed a calculation function, e.g. `EssCalc()`, that you repeatedly check the calculation state by calling `EssGetProcessState()` until the returned value indicates that the calc has completed.  
- "Local operation not allowed in ESSAPI function %s". You have passed a local context handle to a function which does not allow it; use a login context handle instead.  
- "Cannot open message database %s". The message database is not accessible on the machine on which your program is running. Ensure that the message database is where Essbase expects to find it. Essbase first examines the `MessagePath` field of the initialization structure passed to `EssInit()`, then the directory and file name specified by the ARBORMSGPATH environment variable, and finally, the `$ESSBASEPATH\BIN` directory where `$ESSBASEPATH` is an environment variable. If the message database is not available in any of these directories, Essbase returns an error message at run time. Verify which setting Essbase uses, and then verify that the message database is located where specified. See Chapter 3, “Integrating Essbase With Your Product” for more information. |
| Your program is consistently receiving an Essbase error return code from an API function, but no message is displayed, or a message saying "No message for message #%1 in message database" is generated. | Certain internal API errors cannot display a message, typically because the user's context information is not available when the message occurs. In these cases, make a note of the error code returned from the function, then refer to the list of error messages in `messages.txt` to find the corresponding message text. The error constants themselves are contained in `esserror.h`. |
| When accessing fields in API-defined structures, they appear to contain the wrong values, or the values seem to be "shifted" by a few bytes. | Check your compiler defaults to ensure you have structures aligned on byte boundaries. If the problem still occurs, make sure you are compiling with the most recent versions of the API header files, and linking with the most recent API DLLs. |
In This Chapter

- Standard C Language Types ................................................................. 95
- Constant Definitions (C) ............................................................................. 100
- LRO Constant and Structure Definitions (C) ............................................. 107
- Constant and Structure Definitions for Partitions (C) ............................. 109
- Drill-Through Constant and Structure Definitions ................................. 110
- C Main API Structures ........................................................................... 114

Standard C Language Types

The following data types are defined in the Essbase API for the C programming language:

- “Simple Data Types (C)” on page 95
- “Other Data Types (C)” on page 96
- “Bitmask Data Types (C)” on page 96
- “Pointer Types (C)” on page 98
- “Miscellaneous Types (C)” on page 99
- “Array Types (C)” on page 99
- “API Definitions (C)” on page 100

Simple Data Types (C)

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Essbase Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>typedef char</td>
<td>ESS_CHAR_T</td>
</tr>
<tr>
<td>typedef short</td>
<td>ESS_SHORT_T</td>
</tr>
<tr>
<td>typedef long</td>
<td>ESS_LONG_T</td>
</tr>
<tr>
<td>typedef unsigned char</td>
<td>ESS_UCHAR_T</td>
</tr>
<tr>
<td>typedef unsigned short</td>
<td>ESS_USHORT_T</td>
</tr>
<tr>
<td>typedef unsigned long</td>
<td>ESS_ULONG_T</td>
</tr>
</tbody>
</table>
typedef float ESS_FLOAT_T
typedef double ESS_DOUBLE_T

If win32 && _USE_32BIT_TIME_T defined:
typedef __time32_t ESS_TIME_T *
Otherwise,
typedef time_t

typedef unsigned short ESS_DATE_T

If win32 && _USE_32BIT_TIME_T defined:
typedef __time32_t ESS_DATETIME_T *
Otherwise,
typedef time_t

**Note:** * For Visual Studio 2005 or later compilers, the C library data type time_t can be long or int64 Windows datatypes, based on the compiler macro _USE_32BIT_TIME_T. Essbase data types ESS_TIME_T and ESS_DATETIME_T are long for 32-bit Windows platforms.

### Other Data Types (C)

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Essbase Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>typedef void</td>
<td>*ESS_HCTX_T</td>
<td>API context handle</td>
</tr>
<tr>
<td>typedef void</td>
<td>*ESS_HINST_T</td>
<td>API instance handle</td>
</tr>
<tr>
<td>typedef unsigned char</td>
<td>ESS_BOOL_T</td>
<td>boolean</td>
</tr>
<tr>
<td>typedef size_t</td>
<td>ESS_SIZE_T</td>
<td>size of a memory block</td>
</tr>
<tr>
<td>typedef char</td>
<td>*ESS_STR_T</td>
<td>string (array of char)</td>
</tr>
<tr>
<td>typedef void</td>
<td>ESS_VOID_T</td>
<td>void</td>
</tr>
</tbody>
</table>

### Bitmask Data Types (C)

The values for these data types consist of bit values that are combined to provide additional values when appropriate. For example, a caller needing WRITE access to a database must have the READ and WRITE privileges, thus ESS_ACCESS_WRITE equals the bit values for ESS_PRIV_READ and ESS_PRIV_WRITE. Similarly, ESS_OBJTYPE_BACKUP is a combination of ESS_OBJTYPE_ASCBACKUP and ESS_OBJTYPE_BINBACKUP.
<table>
<thead>
<tr>
<th>Data Type</th>
<th>Essbase Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>typedef</td>
<td>ESS_ACCESS_T</td>
<td>Security access level. Possible bit values are:</td>
</tr>
<tr>
<td>unsigned</td>
<td>short</td>
<td>- ESS_PRIV_NONE - 0x0000 - no privilege</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_PRIV_READ - 0x0001 - read data</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_PRIV_WRITE - 0x0002 - write data</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_PRIV_CALC - 0x0004 - calculate data</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_PRIV_DLOAD - 0x0010 - load and unload databases</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_PRIV_DBDESIGN - 0x0020 - manage databases</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_PRIV_DBCREATE - 0x0040 - create, delete, and edit databases</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_PRIV_APPLOAD - 0x0100 - load and unload applications</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_PRIV_APPDESIGN - 0x0200 - manage applications</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_PRIV_APPCREATE - 0x0400 - create, delete, and edit applications</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_PRIV_USERCREATE - 0x1000 - create, delete, and edit users</td>
</tr>
</tbody>
</table>

The access types are combinations of privileges. The valid values are:

- ESS_ACCESS_NONE - 0x0000
- ESS_ACCESS_READ - 0x0111
- ESS_ACCESS_WRITE - 0x0113
- ESS_ACCESS_CALC - 0x0117
- ESS_ACCESS_METAREAD - 0x0118
- ESS_ACCESS_DBMANAGE - 0x0137 (also known as ESS_ACCESS_DBDESIGN, preserved for backward compatibility)
- ESS_ACCESS_DBCREATE - 0x0177
- ESS_ACCESS_APPDESIGN - 0x0377
- ESS_ACCESS_APPCREATE - 0x0777
- ESS_ACCESS_FILTER - 0x0110
- ESS_ACCESS_DBALL - 0x0ff - full database access
- ESS_ACCESS_APPALL - 0xffff - full application/database access
- ESS_ACCESS_ADMIN - 0xffff - administrator (unrestricted access) (also known as ESS_ACCESS_SUPER, preserved for backward compatibility)

The Oracle’s Hyperion® Shared Services security role mappings are:

- ESS_USERPROVROLE_NONE = ESS_ACCESS_NONE = 0x0000
- ESS_USERPROVROLE_USERCREATE = ESS_PRIV_USERCREATE = 0x1000

**Note:** This role cannot be set by Essbase in Shared Services mode; it can only be set in Shared Services.

- ESS_USERPROVROLE_APPCREATE = ESS_PRIV_APPCREATE = 0x0400
- ESS_USERPROVROLE_APPMANAGER = ESS_ACCESS_APPMANAGE or ESS_ACCESS_APPDESIGN = 0x0377
- ESS_USERPROVROLE_APPLOAD = ESS_PRIV_APPLOAD = 0x0100
- ESS_USERPROVROLE_DBFILTER = ESS_ACCESS_FILTER = 0x0110
- ESS_USERPROVROLE_DBREAD = ESS_ACCESS_READ = 0x0111
- ESS_USERPROVROLE_DBWRITE = ESS_ACCESS_WRITE = 0x0113
- ESS_USERPROVROLE_DBCALC = ESS_ACCESS_CALC = 0x0117
- ESS_USERPROVROLE_DBMANAGER = ESS_ACCESS_DBMANAGE or ESS_ACCESS_DBDESIGN = 0x0137
- ESS_USERPROVROLE_ADMINISTRATOR = ESS_ACCESS_ADMIN or ESS_ACCESS_SUPER = 0xffff
### Data Type

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Essbase Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>typedef</td>
<td>ESS_OBJTYPE_T</td>
<td>File object type.</td>
</tr>
<tr>
<td>unsigned</td>
<td></td>
<td></td>
</tr>
<tr>
<td>long</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Single object types are:
- ESS_OBJTYPE_NONE
- ESS_OBJTYPE_OUTLINE
- ESS_OBJTYPE_CALCSCRIPT
- ESS_OBJTYPE_REPORT
- ESS_OBJTYPE_RULES
- ESS_OBJTYPE_ALIAS
- ESS_OBJTYPE_STRUCTURE
- ESS_OBJTYPE_ASCBACKUP
- ESS_OBJTYPE_BINBACKUP
- ESS_OBJTYPE_EXCEL
- ESS_OBJTYPE_LOTUS2 (No longer supported)
- ESS_OBJTYPE_LOTUS3 (No longer supported)
- ESS_OBJTYPE_TEXT
- ESS_OBJTYPE_PARTITION
- ESS_OBJTYPE_LOTUS4 (No longer supported)
- ESS_OBJTYPE_WIZARD
- ESS_OBJTYPE_OTL_E
- ESS_OBJTYPE_SELECTION
- ESS_OBJTYPE_LRO

#define ESS_OBJTYPE_MAX 0x08000000 /* maximum single object type value */ Combined object types are:
- ESS_OBJTYPE_BACKUP
- ESS_OBJTYPE_WORKSHEET
- ESS_OBJTYPE_DATA
- ESS_OBJTYPE_ALL

### Pointer Types (C)

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Essbase Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>*ESS_PCHAR_T</td>
<td>pointer to char</td>
</tr>
<tr>
<td>unsigned char</td>
<td>*ESS_PUCHAR_T</td>
<td>pointer to unsigned char</td>
</tr>
<tr>
<td>short</td>
<td>*ESS_PSHORT_T</td>
<td>pointer to short</td>
</tr>
<tr>
<td>unsigned short</td>
<td>*ESS_PUSHORT_T</td>
<td>pointer to unsigned short</td>
</tr>
<tr>
<td>long</td>
<td>*ESS_PLONG_T</td>
<td>pointer to long</td>
</tr>
<tr>
<td>Data Type</td>
<td>Essbase Type</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>------------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>unsigned long</td>
<td>*ESS_PULONG_T</td>
<td>pointer to unsigned long</td>
</tr>
<tr>
<td>double</td>
<td>*ESS_PDOUBLE_T</td>
<td>pointer to double</td>
</tr>
<tr>
<td>float</td>
<td>*ESS_PFLOAT_T</td>
<td>pointer to float</td>
</tr>
<tr>
<td>ESS_ACCESS_T</td>
<td>*ESS_PACCESS_T</td>
<td>pointer to security access level</td>
</tr>
<tr>
<td>ESS_BOOL_T</td>
<td>*ESS_PBOOL_T</td>
<td>pointer to boolean</td>
</tr>
<tr>
<td>ESS_HCTX_T</td>
<td>*ESS_PHCTX_T</td>
<td>pointer to API context handle</td>
</tr>
<tr>
<td>ESS_HINST_T</td>
<td>*ESS_PHINST_T</td>
<td>pointer to API instance handle</td>
</tr>
<tr>
<td>ESS_HCTX_T</td>
<td>*ESS_PHCTX_T</td>
<td>pointer to API context handle</td>
</tr>
<tr>
<td>ESS_SIZE_T</td>
<td>*ESS_PSIZE_T</td>
<td>pointer to size of a memory block</td>
</tr>
<tr>
<td>ESS_STR_T</td>
<td>*ESS_PSTR_T</td>
<td>pointer to string</td>
</tr>
<tr>
<td>ESS_VOID_T</td>
<td>*ESS_PVOID_T</td>
<td>pointer to void</td>
</tr>
</tbody>
</table>

**Miscellaneous Types (C)**

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Essbase Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>typedef long</td>
<td>ESS_STS_T</td>
<td>return value from API functions</td>
</tr>
<tr>
<td>typedef ESS_STS_T</td>
<td>(*ESS_FUNC_T)()</td>
<td>pointer to function</td>
</tr>
</tbody>
</table>

**Array Types (C)**

The following array types are defined using the appropriate maximum string length. For example, the type `ESS_USERNAME_T` is defined as `typedef char ESS_USERNAME_T[ESS_USERNAME_LEN].`

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Essbase Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>typedef char</td>
<td>ESS_USERNAME_T</td>
<td>user name</td>
</tr>
<tr>
<td>typedef char</td>
<td>ESS_PASSWORD_T</td>
<td>password</td>
</tr>
<tr>
<td>typedef char</td>
<td>ESS_SVRNAME_T</td>
<td>server name</td>
</tr>
<tr>
<td>typedef char</td>
<td>ESS_APPNAME_T</td>
<td>application name</td>
</tr>
<tr>
<td>typedef char</td>
<td>ESS_DBNAME_T</td>
<td>database name</td>
</tr>
<tr>
<td>typedef char</td>
<td>ESS_OBJNAME_T</td>
<td>object name</td>
</tr>
<tr>
<td>Data Type</td>
<td>Essbase Type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-----------------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>typedef char</td>
<td>ESS_MBRNAME_T</td>
<td>member name</td>
</tr>
<tr>
<td>typedef char</td>
<td>ESS_FTRNAME_T</td>
<td>filter name</td>
</tr>
<tr>
<td>typedef char</td>
<td>ESS_ALIASNAME_T</td>
<td>alias table name</td>
</tr>
<tr>
<td>typedef char</td>
<td>ESS_PATH_T</td>
<td>file path name</td>
</tr>
<tr>
<td>typedef char</td>
<td>ESS_DESC_T</td>
<td>app/database description</td>
</tr>
</tbody>
</table>

API Definitions (C)

<table>
<thead>
<tr>
<th>Essbase Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>#define ESS_TRUE</td>
<td>1</td>
</tr>
<tr>
<td>#define ESS_FALSE</td>
<td>0</td>
</tr>
<tr>
<td>#define ESS_NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>#define ESS_NATIVE_SECURITY</td>
<td>1</td>
</tr>
<tr>
<td>#define ESS_SS_SECURITY</td>
<td>2</td>
</tr>
</tbody>
</table>

Constant Definitions (C)

The following constants are defined in the Essbase API:

- “Attributes Constants (C)” on page 100
- “Dimension Tag Constants (C)” on page 103
- “Information Flag Constants (C)” on page 104
- “List Option Constants (C)” on page 105
- “Maximum String Lengths (C)” on page 105
- “Request Type Constants (C)” on page 106
- “Size Flag Constants (C)” on page 106

Attributes Constants (C)

The following constants define the data type of the member queried and returned for the `usInputMemberType` and `usOutputMemberType` fields of the “ESS_ATTRIBUTEQUERY_T” on page 697 structure.
<table>
<thead>
<tr>
<th>Value</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_BASE_DIMENSION</td>
<td>A dimension that is not an attribute dimension</td>
</tr>
<tr>
<td>ESS_BASE_MEMBER</td>
<td>A member that is not an attribute member</td>
</tr>
<tr>
<td>ESS_ATTRIBUTE_DIMENSION</td>
<td>An attribute dimension</td>
</tr>
<tr>
<td>ESS_ATTRIBUTE_MEMBER</td>
<td>An attribute member</td>
</tr>
<tr>
<td>ESS_ATTRIBUTED_MEMBER</td>
<td>A base member or dimension that has attributes associated with it. Also called a standard member or dimension.</td>
</tr>
</tbody>
</table>

The following constant defines the attribute member status for the *Status* field of the “ESS_MBRINFO_T” on page 699 structure.

<table>
<thead>
<tr>
<th>Value</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_MBRSTS_ATTRIBUTE</td>
<td>Attribute member status</td>
</tr>
</tbody>
</table>

The following constants define the attribute dimension tag type for the *DimTag* field of the “ESS_DIMENSIONINFO_T” on page 132 structure.

<table>
<thead>
<tr>
<th>Value</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_TTYPE_ATTRIBUTE</td>
<td>Attribute tag</td>
</tr>
<tr>
<td>ESS_TTYPE_ATTRCALC</td>
<td>Attribute calculation tag. Used internally for aggregation.</td>
</tr>
</tbody>
</table>

The following constants define the attribute member data type for the *usDataType* field of the “ESS_ATTRIBUTEVALUE_T” on page 118 structure and the *DimDataType* field of the “ESS_DIMENSIONINFO_T” on page 132 structure.

<table>
<thead>
<tr>
<th>Value</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_ATTRMBRD_BOOL</td>
<td>Boolean data type</td>
</tr>
<tr>
<td>ESS_ATTRMBRD_DATETIME</td>
<td>Datetime data type</td>
</tr>
<tr>
<td>ESS_ATTRMBRD_DOUBLE</td>
<td>Double data type</td>
</tr>
<tr>
<td>ESS_ATTRMBRD_STRING</td>
<td>String data type</td>
</tr>
<tr>
<td>ESS_ATTRMBRD_NONE</td>
<td>No data type</td>
</tr>
</tbody>
</table>

The following constants define the type of attribute query operation for the *usOperation* field of the “ESS_ATTRIBUTEQUERY_T” on page 697 structure.

<table>
<thead>
<tr>
<th>Value</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_EQ</td>
<td>Equal to</td>
</tr>
<tr>
<td>Term</td>
<td>Definition</td>
</tr>
<tr>
<td>-----------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>bucketing type</td>
<td>When building a dimension, you can associate a zero-level attribute member of type <code>ESS_ATTRMBRDTYPE_DOUBLE</code> with a range of data in a relational source. Bucketing type determines the upper or lower limit of the data range. See <code>usBucketingType</code>.</td>
</tr>
<tr>
<td>ESS_ATTRIBUTE_DIMENSION</td>
<td><code>ESS_ATTRIBUTE_DIMENSION</code> is an attribute dimension.</td>
</tr>
<tr>
<td>ESS_ATTRIBUTE_MEMBER</td>
<td><code>ESS_ATTRIBUTE_MEMBER</code> is a member of an attribute dimension. See “<code>ESS_ATTRIBUTEQUERY_T</code>” on page 697.</td>
</tr>
<tr>
<td></td>
<td>Also see <code>EssCheckAttributes</code>.</td>
</tr>
<tr>
<td>ESS_ATTRIBUTED_MEMBER</td>
<td><code>ESS_ATTRIBUTED_MEMBER</code> is a member (of a base dimension) which has an attribute member associated with it. See “<code>ESS_ATTRIBUTEQUERY_T</code>” on page 697.</td>
</tr>
<tr>
<td></td>
<td>Also see <code>EssCheckAttributes</code>.</td>
</tr>
<tr>
<td>ESS_BASE_DIMENSION</td>
<td><code>ESS_BASE_DIMENSION</code> is a standard dimension that has an attribute dimension associated with it.</td>
</tr>
<tr>
<td>ESS_BASE_MEMBER</td>
<td><code>ESS_BASE_MEMBER</code> is a member of a base dimension. See “<code>ESS_ATTRIBUTEQUERY_T</code>” on page 697.</td>
</tr>
<tr>
<td></td>
<td>Also see <code>EssCheckAttributes</code>.</td>
</tr>
<tr>
<td>ESS_STANDARD_DIMENSION</td>
<td><code>ESS_STANDARD_DIMENSION</code> is any dimension that is not an attribute dimension.</td>
</tr>
<tr>
<td>ESS_STANDARD_MEMBER</td>
<td><code>ESS_STANDARD_MEMBER</code> is a member of a standard dimension. See “<code>ESS_ATTRIBUTEQUERY_T</code>” on page 697.</td>
</tr>
<tr>
<td></td>
<td>Also see <code>EssCheckAttributes</code>.</td>
</tr>
</tbody>
</table>
### Dimension Tag Constants (C)

The following constants define the available information flags used in the \textit{DimTag} field of the \textit{“ESS\_DIMENSIONINFO\_T”} on page 132 structure.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_TTYPE_NONE</td>
<td>No dimension type. Value for \textit{DimTag} field of ESS_DIMENSIONINFO_T</td>
</tr>
<tr>
<td>ESS_TTYPE_CCATEGORY</td>
<td>Accounts: Currency \textit{ACCOUNTS} tag. Value for \textit{DimTag} field of ESS_DIMENSIONINFO_T</td>
</tr>
<tr>
<td>ESS_TTYPE_CNAME</td>
<td>Country: Currency \textit{COUNTRY} tag. Value for \textit{DimTag} field of ESS_DIMENSIONINFO_T</td>
</tr>
<tr>
<td>ESS_TTYPE_CTIME</td>
<td>Time: Currency \textit{TIME} tag. Value for \textit{DimTag} field of ESS_DIMENSIONINFO_T</td>
</tr>
<tr>
<td>ESS_TTYPE_CTYPE</td>
<td>Type: Currency \textit{TYPE} tag. Value for \textit{DimTag} field of ESS_DIMENSIONINFO_T</td>
</tr>
<tr>
<td>ESS_TTYPE_CPARTITION</td>
<td>Currency \textit{PARTITION} tag. Value \textit{DimTag} field of ESS_DIMENSIONINFO_T</td>
</tr>
</tbody>
</table>

### Implied Share Setting (C)

Implied Share settings can apply to a specific outline, using the \texttt{EssOtlGetImpliedShare} and \texttt{EssOtlSetImpliedShare} functions.

No changes take effect until the outline is saved and restructured.
Note: The explicit settings are especially useful if the application later is copied, as the setting would be “sticky” and follow the outline independent of any application name specific entry in the Essbase.cfg file.

The setting can have the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS IMPLIEDSHARE_DEFAULT</td>
<td>Can be set using EssOtlSetImpliedShare. When set, immediately gets converted to either _ON or _OFF. If returned:</td>
</tr>
<tr>
<td></td>
<td>- Outline has no Implied Share setting</td>
</tr>
<tr>
<td></td>
<td>- Implied Share is ON</td>
</tr>
<tr>
<td>ESS IMPLIEDSHARE_DEFAULT_ON</td>
<td>Return value only available with EssOtlGetImpliedShare. If returned:</td>
</tr>
<tr>
<td></td>
<td>- Outline uses Implied Share default setting in Essbase.cfg</td>
</tr>
<tr>
<td></td>
<td>- Essbase.cfg might contain an Implied Share entry (ON)</td>
</tr>
<tr>
<td></td>
<td>- Essbase.cfg should contain no entry</td>
</tr>
<tr>
<td></td>
<td>- Implied Share is ON</td>
</tr>
<tr>
<td>ESS IMPLIEDSHARE_DEFAULT_OFF</td>
<td>Return value only available with EssOtlGetImpliedShare. If returned:</td>
</tr>
<tr>
<td></td>
<td>- Outline uses Implied Share default setting in Essbase.cfg</td>
</tr>
<tr>
<td></td>
<td>- Essbase.cfg contains an Implied Share entry (OFF)</td>
</tr>
<tr>
<td></td>
<td>- Implied Share is OFF</td>
</tr>
<tr>
<td>ESS IMPLIEDSHARE _FORCE_ON</td>
<td>Can be set using EssOtlSetImpliedShare. An explicit setting indicating the outline always has Implied Share ON.</td>
</tr>
<tr>
<td>ESS IMPLIEDSHARE _FORCE_OFF</td>
<td>Can be set using EssOtlSetImpliedShare. An explicit setting indicating the outline always has Implied Share OFF.</td>
</tr>
</tbody>
</table>

Information Flag Constants (C)

The following constants define the available information flags used in the DbReqFlags (Data Load) field of the “ESS_DBREQINFO_T” on page 126 structure.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_DBREQFLAG_CALCDEF</td>
<td>Default flag for DbReqFlags field. Used the default calc script. Value: 0x00000001.</td>
</tr>
<tr>
<td>ESS_DBREQFLAG_CALCDSCR</td>
<td>Custom calc script flag for DbReqFlags field. Used a custom calc script. Value: 0x00000002.</td>
</tr>
</tbody>
</table>
List Option Constants (C)

The following constants define request types used by the ListOption field of the

EssListTransactions function.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_LIST_TRANSACTIONS_TOCLIENT</td>
<td>Write the output to the screen.</td>
</tr>
<tr>
<td>LIST_TRANSACTIONS_TOFILE</td>
<td>- Write output to a CSV file.</td>
</tr>
<tr>
<td></td>
<td>- Output is not returned in ppResults.</td>
</tr>
<tr>
<td></td>
<td>- pCount and ppResults will be NULL.</td>
</tr>
<tr>
<td></td>
<td>- Content is written to the FileName as comma separated file.</td>
</tr>
<tr>
<td></td>
<td>- If the specified file name exists, the command fails.</td>
</tr>
<tr>
<td>ESS_LIST_TRANSACTIONS_FORCETOFILE</td>
<td>- Write output to a CSV file.</td>
</tr>
<tr>
<td></td>
<td>- Output is not returned in ppResults.</td>
</tr>
<tr>
<td></td>
<td>- pCount and ppResults will be NULL.</td>
</tr>
<tr>
<td></td>
<td>- Content is written to the FileName as comma separated file.</td>
</tr>
<tr>
<td></td>
<td>- If the specified file name exists, it is overwritten with the new output.</td>
</tr>
</tbody>
</table>

Maximum String Lengths (C)

The following constants define the maximum lengths of various string types in the Essbase API. All of these constants include the terminating NULL character:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_ALIASNAMELEN</td>
<td>Maximum length of an alias table name</td>
</tr>
<tr>
<td>ESS_APPNAMELEN</td>
<td>Maximum length of an application name</td>
</tr>
<tr>
<td>ESS_CRDB_MAXIMUM</td>
<td>Maximum dimension number for a Currency database</td>
</tr>
<tr>
<td>ESS_DBNAMELEN</td>
<td>Maximum length of a database name</td>
</tr>
<tr>
<td>ESS_DESCLEN</td>
<td>Maximum length of an application or database description</td>
</tr>
<tr>
<td>ESS_FTRNAMELEN</td>
<td>Maximum length of a filter name</td>
</tr>
<tr>
<td>ESS_LINELEN</td>
<td>Maximum length of a line in a report</td>
</tr>
<tr>
<td>ESS_MBRCOMMENTEXLEN</td>
<td>Maximum length of an extended member comment</td>
</tr>
<tr>
<td>ESS_MBRNAMELEN</td>
<td>Maximum length of a member name</td>
</tr>
<tr>
<td>ESS_NAMELEN</td>
<td>Maximum length of a general name</td>
</tr>
<tr>
<td>ESS_PASSWORDLEN</td>
<td>Maximum length of a user password</td>
</tr>
<tr>
<td>ESS_PATHLEN</td>
<td>Maximum length of a file path name</td>
</tr>
<tr>
<td>Constant</td>
<td>Definition</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>ESS_OBJNAMELEN</td>
<td>Maximum length of an object name</td>
</tr>
<tr>
<td>ESS_SVRNAMELEN</td>
<td>Maximum length of a server name</td>
</tr>
<tr>
<td>ESS_USERNAMELEN</td>
<td>Maximum length of a user or group name</td>
</tr>
</tbody>
</table>

**Request Type Constants (C)**

The following constants define request types used by the `ucReqType` field of the “ESS_TRANSACTION_REQSPECIFIC_T” on page 188 structure.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_TRLOG_CALCSCRIPT_SERVER</td>
<td>Calculation script name</td>
</tr>
<tr>
<td>ESS_TRLOG_CALCSCRIPT_IMMEDIATE</td>
<td>No data</td>
</tr>
<tr>
<td>ESS_TRLOG_CALCSCRIPT_DEFAULT</td>
<td>No data</td>
</tr>
<tr>
<td>ESS_TRLOG_CALCSCRIPT_SETDEFAULT</td>
<td>No data</td>
</tr>
<tr>
<td>ESS_TRLOG_DATALOAD_SERVER</td>
<td>Server side data load file</td>
</tr>
<tr>
<td>ESS_TRLOG_DATALOAD_IMMEDIATE</td>
<td>Data was input from client side</td>
</tr>
<tr>
<td>ESS_TRLOG_DATALOAD_SQL</td>
<td>SQL data source</td>
</tr>
<tr>
<td>ESS_TRLOG_DATALOAD_CLEARDB</td>
<td>Clear all data</td>
</tr>
<tr>
<td>ESS_TRLOG_DATALOAD_RESETDB</td>
<td>Clear all data and out line</td>
</tr>
<tr>
<td>ESS_TRLOG_SSUPDATE</td>
<td>Spreadsheet updates</td>
</tr>
<tr>
<td>ESS_TRLOG_DATALOAD_FTP</td>
<td>FTP data source</td>
</tr>
</tbody>
</table>

**Size Flag Constants (C)**

The following constants define the maximum and minimum size for the `MaxMemIndex` and `IndexPageSize` fields of the “ESS_DBSTATE_T” on page 127 structure.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_INDEXCACHEMIN_SIZE</td>
<td>Minimum index cache size for the <code>MaxMemIndex</code> field of the ESS_DBSTATE_T structure. Value: 1048576. No maximum value is defined.</td>
</tr>
<tr>
<td>ESS_INDEXPAGEMAX_SIZE</td>
<td>Maximum index page size for the <code>IndexPageSize</code> field of the ESS_DBSTATE_T structure. Value: 8192</td>
</tr>
<tr>
<td>ESS_INDEXPAGEMIN_SIZE</td>
<td>Minimum index page size for the <code>IndexPageSizeMin</code> field of the ESS_DBSTATE_T structure. Value: 1024</td>
</tr>
</tbody>
</table>
**Unicode Mode Constants (C)**

The following constants enable Unicode-mode client programs. These constants are the valid values for the `usApiType` field of the `ESS_INIT_T` structure. The `ESS_INIT_T` structure is used by `EssInit()` and defines whether the client program is in Unicode mode. Only Unicode-mode client programs can send UTF-8 encoded text to Essbase Server.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_API_NONUNICODE</td>
<td>0x0002</td>
<td>The program is a non-Unicode mode client program. The client program is passing a non_Unicode encoded argument to the API. This is the default value.</td>
</tr>
<tr>
<td>ESS_API_UTF8</td>
<td>0x0003</td>
<td>The program is a Unicode mode client program. The client program is passing a UTF-8 encoded argument to the API.</td>
</tr>
</tbody>
</table>

**LRO Constant and Structure Definitions (C)**

The following constants and structures are defined specifically for use with Linked Reporting Objects (LROs):

- “Constants for LROs (C)” on page 107
- “ESS_CELLADDR_API_T” on page 108
- “ESS_LRODESC_API_T” on page 108
- “ESS_LRONOTE_API_T” on page 109
- “ESS_LROHANDLE_API_T” on page 109
- “ESS_LROINFO_API_T” on page 109

**Constants for LROs (C)**

The following constants define various values used by LRO functions and structures in the Essbase API.

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_LRODESCLEN_API</td>
<td>79</td>
<td>Maximum length of an object description</td>
</tr>
<tr>
<td>ESS_LRONOTELEN_API</td>
<td>599</td>
<td>Maximum length of a cell note</td>
</tr>
<tr>
<td>ESS_ONAMELEN_API</td>
<td>511</td>
<td>Length of an object name consisting of file name and path</td>
</tr>
<tr>
<td>ESS_DATESIZE</td>
<td>12</td>
<td>Size of date string</td>
</tr>
<tr>
<td>ESS_STORE_OBJECT_API</td>
<td>0x0010</td>
<td>Value to store a linked object on the server</td>
</tr>
<tr>
<td>ESS_NOSTORE_OBJECT_API</td>
<td>0x0001</td>
<td>Value to not store a linked object on the server</td>
</tr>
<tr>
<td>ESS_LROTYPE_CELLNOTE_API</td>
<td>0</td>
<td>Value specifying that a linked object is a cell note</td>
</tr>
<tr>
<td>ESS_LROTYPE_WINAPP_API</td>
<td>1</td>
<td>Value specifying that a linked object is a Windows application</td>
</tr>
</tbody>
</table>
### ESS_LROTYPE_URL_API_T

Contains information describing a specific object linked to a data cell in an Essbase database. The fields are described as follows:

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_ULONG_T</td>
<td>cellOffset</td>
<td>Cell offset within a data block</td>
</tr>
<tr>
<td>ESS_SECPART_T</td>
<td>blkOffset</td>
<td>Block offset</td>
</tr>
<tr>
<td>ESS_SECPART_T</td>
<td>segment</td>
<td>Segment number</td>
</tr>
</tbody>
</table>

### ESS_CELLADDR_API_T

Contains information about the address of a data cell in an Essbase database. Essbase derives the cell address from the member combination and uses the address to keep track of objects linked to data cells. You cannot modify fields in this structure through the API. The fields are described as follows:

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_ULONG_T</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ESS_SECPART_T</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ESS_SECPART_T</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### ESS_LRODESC_API_T

Contains information describing a specific object linked to a data cell in an Essbase database. The fields are described as follows:

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>struct ESS_LRODESC_API_T</td>
<td>next</td>
<td>(The next field is for internal use only.)</td>
</tr>
<tr>
<td>ESS_USHORT_T</td>
<td>usObjType</td>
<td>The object type</td>
</tr>
<tr>
<td>ESS_USHORT_T</td>
<td>status</td>
<td>The catalog entry status</td>
</tr>
<tr>
<td>ESS_LROHANDLE_API_T</td>
<td>linkId</td>
<td>Link ID of the LRO</td>
</tr>
<tr>
<td>ESS_CHAR_T</td>
<td>userName[ESS_USERNAMELEN]</td>
<td>The name of the last user to modify the object</td>
</tr>
<tr>
<td>ESS_TIME_T</td>
<td>updateDate</td>
<td>The last date the object was modified</td>
</tr>
<tr>
<td>ESS_ACCESS_T</td>
<td>accessLevel</td>
<td>The access level of the member combination</td>
</tr>
<tr>
<td>ESS_ULONG_T</td>
<td>memCount</td>
<td>The number of members in the member combination</td>
</tr>
<tr>
<td>ESS_PMBRNAME_NONUNI_T</td>
<td>pMemComb</td>
<td>The member combination associated with the object</td>
</tr>
<tr>
<td>ESS_LROINFO_API_T</td>
<td>lroInfo</td>
<td>The LRO information structure, associated by union</td>
</tr>
<tr>
<td>ESS_CHAR_T</td>
<td>note[ESS_LRONOTELEN_API]</td>
<td>A cell note, associated by union</td>
</tr>
</tbody>
</table>
**ESS_LROHANDLE_API_T**

Provides an identifier for a linked object. The identifier consists of a cell address and an internal object handle. You should not modify fields in this structure because it contains information concerning the linked object. The fields are described as follows:

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_CELLADDR_API_T</td>
<td>cellKey</td>
<td>Cell address</td>
</tr>
<tr>
<td>ESS_LONG_T</td>
<td>hObject</td>
<td>Internal object handle</td>
</tr>
</tbody>
</table>

**ESS_LROINFO_API_T**

Contains information about a specific object linked to a data cell in an Essbase database. You should not modify fields in this structure because it contains information concerning the linked object. The fields are described as follows:

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_CHAR_T</td>
<td>objName[ESS_ONAMELEN_API]</td>
<td>Source file name of object linked to a data cell. ESS_ONAMELEN_API specifies the maximum length of an object name; the default value is 511.</td>
</tr>
<tr>
<td>ESS_CHAR_T</td>
<td>objDesc[ESSW_LRODESCLEN_API]</td>
<td>Description of an object linked to a data cell. ESS_LRODESCLEN_API specifies the maximum length of the description; the default value is 79.</td>
</tr>
</tbody>
</table>

**Constant and Structure Definitions for Partitions (C)**

“ESS_PART_T” on page 152

“ESS_PART_CONNECT_INFO_T” on page 153

“ESS_PART_DEFINED_T” on page 153

“ESS_PART_INFO_T” on page 154

“ESS_PART_REPL_T” on page 155

“ESS_PARTDEF_INVALID_T” on page 155

“ESS_PARTDEF_CONNECT_T” on page 156

“ESS_PARTDEF_MAP_T” on page 157

“ESS_PARTDEF_T” on page 157

“ESS_PARTDEF AREAS_T” on page 158

“ESS_PARTDEF_TYPE_T” on page 158

“ESS_PARTHDR_T” on page 159

“ESS_PARTOTL_DIMASSOCCHG_API_T” on page 162

“ESS_PARTOTL_DIM_ATTRIB_API_T” on page 161
Drill-Through Constant and Structure Definitions

These topics discuss the C Main API constants and structures that are defined specifically for use with Drill-Through:

- “C Main API Drill-Through Constants and Structures (essdt.dll)” on page 110
- “C Main Drill-Through Constants and Structures (essdtapi.dll)” on page 112

C Main API Drill-Through Constants and Structures (essdt.dll)

Structures

- “ESS_DTBUFFER_T” on page 136
- “ESS_DTDATA_T” on page 136
- “ESS_DTHEADER_T” on page 137

Constants for Maximum String Length

The following constants define the maximum lengths of various string types in the Essbase API. All of these constants include the terminating NULL character:
<table>
<thead>
<tr>
<th>Constant</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_ALIASNAMELEN</td>
<td>Maximum length of an alias table name</td>
</tr>
<tr>
<td>ESS_APPNAMELEN</td>
<td>Maximum length of an application name</td>
</tr>
<tr>
<td>ESS_CRDB_MAXIMUM</td>
<td>Maximum dimension number for a Currency database</td>
</tr>
<tr>
<td>ESS_DBNAMELEN</td>
<td>Maximum length of a database name</td>
</tr>
<tr>
<td>ESS_DESCLEN</td>
<td>Maximum length of an application or database description</td>
</tr>
<tr>
<td>ESS_DESCRIPTION_LEN</td>
<td>Maximum string length (255) used for drill-through</td>
</tr>
<tr>
<td>ESS_DTREPORT_NAME</td>
<td>Maximum string length (80) used for drill-through</td>
</tr>
<tr>
<td>ESS_FIRNAMELEN</td>
<td>Maximum length of a filter name</td>
</tr>
<tr>
<td>ESS_LINELEN</td>
<td>Maximum length of a line in a report</td>
</tr>
<tr>
<td>ESS_MAX_DATALEN</td>
<td>Maximum string length (255) used for drill-through</td>
</tr>
<tr>
<td>ESS_MAX_NAME</td>
<td>Maximum string length (30) used for drill-through</td>
</tr>
<tr>
<td>ESS_MBRCOMMENTEXLEN</td>
<td>Maximum length of an extended member comment</td>
</tr>
<tr>
<td>ESS_MBRNAMELEN</td>
<td>Maximum length of a member name</td>
</tr>
<tr>
<td>ESS_NAMELEN</td>
<td>Maximum length of a general name</td>
</tr>
<tr>
<td>ESS_PASSWORDLEN</td>
<td>Maximum length of a user password</td>
</tr>
<tr>
<td>ESS_PATHLEN</td>
<td>Maximum length of a file path name</td>
</tr>
<tr>
<td>ESS_OBJNAMELEN</td>
<td>Maximum length of an object name</td>
</tr>
<tr>
<td>ESS_SVRNAMELEN</td>
<td>Maximum length of a server name</td>
</tr>
<tr>
<td>ESS_USERNAMELEN</td>
<td>Maximum length of a user or group name</td>
</tr>
</tbody>
</table>

**Pointer Types**

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Essbase Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>*ESS_PCHAR_T</td>
<td>pointer to char</td>
</tr>
<tr>
<td>unsigned char</td>
<td>*ESS_PUCHAR_T</td>
<td>pointer to unsigned char</td>
</tr>
<tr>
<td>short</td>
<td>*ESS_PSHORT_T</td>
<td>pointer to short</td>
</tr>
<tr>
<td>unsigned short</td>
<td>*ESS_PUSHORT_T</td>
<td>pointer to unsigned short</td>
</tr>
<tr>
<td>long</td>
<td>*ESS_PLONG_T</td>
<td>pointer to long</td>
</tr>
<tr>
<td>unsigned long</td>
<td>*ESS_PULONG_T</td>
<td>pointer to unsigned long</td>
</tr>
<tr>
<td>double</td>
<td>*ESS_PDOUBLE_T</td>
<td>pointer to double</td>
</tr>
</tbody>
</table>
### C Main Drill-Through Constants and Structures (essdtapi.dll)

#### Structures
- “ESS_DTAPICOLUMN_T” on page 134
- “ESS_DTAPIDATA_T” on page 134
- “ESS_DTAPIHEADER_T” on page 135
- “ESS_DTAPIINFO_T” on page 135
- “ESS_DTAPIREPORT_T” on page 136

#### Constants for Maximum String Length
The following constants define the maximum lengths of various string types in the Essbase API. All of these constants include the terminating NULL character:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_ALIASNAMELEN</td>
<td>Maximum length of an alias table name</td>
</tr>
<tr>
<td>ESS_APPNAMELEN</td>
<td>Maximum length of an application name</td>
</tr>
<tr>
<td>ESS_CRDB_MAXIMUM</td>
<td>Maximum dimension number for a Currency database</td>
</tr>
<tr>
<td>ESS_DBNAMELEN</td>
<td>Maximum length of a database name</td>
</tr>
<tr>
<td>ESS_DESCLEN</td>
<td>Maximum length of an application or database description</td>
</tr>
<tr>
<td>ESSID_DESCRIPTION_LEN</td>
<td>Maximum string length (255) used for drill-through</td>
</tr>
<tr>
<td>Constant</td>
<td>Definition</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>---------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ESS_DTREPORT_NAME</td>
<td>Maximum string length (80) used for drill-through</td>
</tr>
<tr>
<td>ESS_FTRNAMELEN</td>
<td>Maximum length of a filter name</td>
</tr>
<tr>
<td>ESS_LINELEN</td>
<td>Maximum length of a line in a report</td>
</tr>
<tr>
<td>ESS_MAX_DATALEN</td>
<td>Maximum string length (255) used for drill-through</td>
</tr>
<tr>
<td>ESS_MAX_NAME</td>
<td>Maximum string length (30) used for drill-through</td>
</tr>
<tr>
<td>ESS_MBRCOMMENTEXLEN</td>
<td>Maximum length of an extended member comment</td>
</tr>
<tr>
<td>ESS_MBRNAMELEN</td>
<td>Maximum length of a member name</td>
</tr>
<tr>
<td>ESS_NAMELEN</td>
<td>Maximum length of a general name</td>
</tr>
<tr>
<td>ESS_PASSWORDLEN</td>
<td>Maximum length of a user password</td>
</tr>
<tr>
<td>ESS_PATHLEN</td>
<td>Maximum length of a file path name</td>
</tr>
<tr>
<td>ESS_OBJNAMELEN</td>
<td>Maximum length of an object name</td>
</tr>
<tr>
<td>ESS_SVRNAMELEN</td>
<td>Maximum length of a server name</td>
</tr>
<tr>
<td>ESS_USERNAMELEN</td>
<td>Maximum length of a user or group name</td>
</tr>
</tbody>
</table>

Drill-Through Connection Values for uInputOption in ESS_DTAPIINFO_T

The following constants define input values to connect to Oracle Essbase Studio for drill-through.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_DTAPI_PROMPT_HISNAME</td>
<td>A value for uInputOption which means that the user can connect to Essbase Studio to perform a drill-through</td>
</tr>
<tr>
<td>ESS_DTAPI_PROMPT_LOGIN</td>
<td>A value for uInputOption which means that a password is required to connect to Essbase Studio to perform a drill-through</td>
</tr>
</tbody>
</table>

Pointer Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Essbase Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>*ESS_PCHAR_T</td>
<td>pointer to char</td>
</tr>
<tr>
<td>unsigned char</td>
<td>*ESS_PCHAR_T</td>
<td>pointer to unsigned char</td>
</tr>
<tr>
<td>short</td>
<td>*ESS_PSHORT_T</td>
<td>pointer to short</td>
</tr>
<tr>
<td>unsigned short</td>
<td>*ESS_PSHORT_T</td>
<td>pointer to unsigned short</td>
</tr>
<tr>
<td>long</td>
<td>*ESS_PLONG_T</td>
<td>pointer to long</td>
</tr>
</tbody>
</table>
### Data Type | Essbase Type | Description
--- | --- | ---
unsigned long | *ESS_PULONG_T | pointer to unsigned long
double | *ESS_PDOUBLE_T | pointer to double
float | *ESS_PFLOAT_T | pointer to float
ESS_ACCESS_T | *ESS_PACCESS_T | pointer to security access level
ESS_BOOL_T | *ESS_PBOOL_T | pointer to boolean
ESS_DTAPIHINST_T | *ESS_PDTAPIHINST_T | pointer to a drill-through initialization structure
ESS_DTHINST_T | *ESS_PDTHINST_T | pointer to a drill-through initialization structure
ESS_HCTX_T | *ESS_PHCTX_T | pointer to API context handle
ESS_HINST_T | *ESS_PHINST_T | pointer to API instance handle
ESS_HCTX_T | *ESS_PHTCTX_T | pointer to API context handle
ESS_SIZE_T | *ESS_PSIZE_T | pointer to size of a memory block
ESS_STR_T | *ESS_PSTR_T | pointer to string
ESS_VOID_T | *ESS_PVOID_T | pointer to void

### C Main API Structures

Consult the Contents pane for the list of C Main API structures.

**ESS_APPDB_T**

This application and database name structure returns application and database names. The fields are:

```c
typedef struct  ESS_APPDB_T
{
    ESS_APPNAME_T AppName;
    ESS_DBNAME_T  DbName;
} ESS_APPDB_T, *ESS_PAPPDB_T, **ESS_PPAPPDB_T;
```

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_APPNAME_T</td>
<td><em>AppName</em></td>
<td>The application name</td>
</tr>
<tr>
<td>ESS_DBNAME_T</td>
<td><em>DbName</em></td>
<td>The database name</td>
</tr>
</tbody>
</table>
ESS_APPINFO_T

This Application Info Structure returns information about a specific application. Fields in this structure cannot be modified using the API. See the “ESS_APPSTATE_T” on page 117 structure, which contains additional application state parameters that can be modified. The fields are:

**Note:** Refer also to the locale-specific extended Application Info structure, “ESS_APPINFOEX_T” on page 116.

typedef struct ESS_APPINFO_T
{
    ESS_APPNAME_T Name;
    ESS_SVRNAME_T Server;
    ESS_USHORT_T Status;
    ESS_USHORT_T, AppType;
    ESS_CHAR_T, AppLocale, ESS_LOCALESTRING_LENGTH;
    ESS_USHORT_T nConnects;
    ESS_TIME_T ElapsedAppTime;
    ESS_USHORT_T nDbs;
    ESS_DATA_STORAGE_T StorageType;
    ESS_DBNAME_T DbNames[1];
} ESS_APPINFO_T, *ESS_PAPPINFO_T, **ESS_PPAPPINFO_T;

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_APPNAME_T</td>
<td>Name</td>
<td>The application name</td>
</tr>
<tr>
<td>ESS_SVRNAME_T</td>
<td>Server</td>
<td>The server name</td>
</tr>
<tr>
<td>ESS_USHORT_T</td>
<td>Status</td>
<td>The application load status (loaded or not loaded). This field can contain</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the following values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_STATUS_NOTLOADED</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_STATUS_LOADING</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_STATUS_LOADED</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_STATUS_UNLOADING</td>
</tr>
<tr>
<td>ESS_USHORT_T</td>
<td>AppType</td>
<td>The type of application. Valid values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_APP_UNICODE - 0x0003 - The program is a Unicode client program. The</td>
</tr>
<tr>
<td></td>
<td></td>
<td>function fails if the server is not in Unicode mode. This is the default</td>
</tr>
<tr>
<td></td>
<td></td>
<td>value.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_APP_NONUNICODE - 0x0002 - The program is a non-Unicode mode client</td>
</tr>
<tr>
<td></td>
<td></td>
<td>program.</td>
</tr>
<tr>
<td>ESS_CHAR_T</td>
<td>AppLocale</td>
<td>The application locale description, of type ESS_LOCALESTRING_LENGTH.</td>
</tr>
<tr>
<td>ESS_USHORT_T</td>
<td>nConnects</td>
<td>The number of users currently connected to the application</td>
</tr>
<tr>
<td>ESS_TIME_T</td>
<td>ElapsedAppTime</td>
<td>Elapsed number of seconds since application loading</td>
</tr>
<tr>
<td>ESS_USHORT_T</td>
<td>nDbs</td>
<td>The number of databases in this application</td>
</tr>
</tbody>
</table>
### ESS_DATA_STORAGE_T

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_DATA_STORAGE_T</td>
<td>StorageType</td>
<td>The storage type. The valid values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● 0 - the default</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● 1 - multidimensional</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● 2 - DB2 relational</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● 3 - Oracle relational</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● 4 - aggregate storage (ASO)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● 1000 - Undefined</td>
</tr>
</tbody>
</table>

### ESS_DBNAME_T

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_DBNAME_T</td>
<td>DbNames[1]</td>
<td>A dynamic array (with nDb elements) of database name strings listing all the databases in the application.</td>
</tr>
</tbody>
</table>

### ESS_APPINFOEX_T

This extended Application Info structure is slightly different from the standard "ESS_APPINFO_T" on page 115 structure used by `EssGetApplicationInfo`. This extended structure is used by `EssGetApplicationInfoEx`.

The fields are:

```c
typedef struct ESS_APPINFOEX_T {
    ESS_APPNAME_T Name;
    ESS_SVRNAME_T Server;
    ESS_USHORT_T AppType;
    ESS_CHAR_T AppLocale, ESS_LOCALESTRING_LENGTH;
    ESS_USHORT_T Status;
    ESS_USHORT_T nConnects;
    ESS_TIME_T ElapsedAppTime;
    ESS_DATA_STORAGE_T StorageType;
} ESS_APPINFOEX_T, *ESS_PAPPINFOEX_T, **ESS_PPAPPINFOEX_T;
```

### Data Type

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_APPNAME_T</td>
<td>Name</td>
<td>Application name</td>
</tr>
<tr>
<td>ESS_SVRNAME_T</td>
<td>Server</td>
<td>Server name</td>
</tr>
<tr>
<td>ESS_USHORT_T</td>
<td>AppType</td>
<td>The type of application. Valid values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESS_APP_UNICODE - 0x0003 - The program is a Unicode client program. The function fails if the server is not in Unicode mode. This is the default value.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESS_APP_NONUNICODE - 0x0002 - The program is a non-Unicode mode client program.</td>
</tr>
<tr>
<td>ESS_CHAR_T</td>
<td>AppLocale</td>
<td>The application locale description, of type ESS_LOCALESTRING_LENGTH.</td>
</tr>
</tbody>
</table>
### Data Type | Field      | Description
---|-------------|-------------------
ESS_USHORT_T | Status      | The application load status (loaded or not loaded). This field can contain the following values:
|             |             | - ESS_STATUS_NOTLOADED
|             |             | - ESS_STATUS_LOADING
|             |             | - ESS_STATUS_LOADED
|             |             | - ESS_STATUS_UNLOADING

ESS_USHORT_T | nConnects   | The number of users currently connected to the application

ESS_TIME_T | ElapsedAppTime | Elapsed number of seconds since application loading

ESS_DATA_STORAGE_T | StorageType | The storage type. The valid values are:
|             |             | - 0 - the default
|             |             | - 1 - multidimensional
|             |             | - 2 - DB2 relational
|             |             | - 3 - Oracle relational
|             |             | - 4 - aggregate storage (ASO)
|             |             | - 1000 - Undefined

## ESS_APPSTATE_T

This Application State Structure gets and sets the state parameters for a specific application. All fields in this structure can be modified using the API, with the exception that some fields do not apply to aggregate storage databases. See also the “ESS_APPINFO_T” on page 115 structure, which contains additional application information that cannot be modified. The fields are:

```c
typedef struct  ESS_APPSTATE_T
{
    ESS_DESC_T   Description;
    ESS_BOOL_T   Loadable;
    ESS_BOOL_T   Autoload;
    ESS_ACCESS_T Access;
    ESS_BOOL_T   Connects;
    ESS_BOOL_T   Commands;
    ESS_BOOL_T   Updates;
    ESS_BOOL_T   Security;
    ESS_ULONG_T  LockTimeout;
    ESS_ULONG_T  lroSizeLimit;
} ESS_APPSTATE_T, *ESS_PAPPSTATE_T, **ESS_PPAPPSTATE_T;
```

### Data Type | Field    | Description
---|-----------|-------------------
ESS_DESC_T | Description | The application description (up to 80 characters)

ESS_BOOL_T | Loadable   | Flag to indicate whether application can be loaded (ESS_TRUE : application is loadable)

ESS_BOOL_T | Autoload   | Flag to indicate whether the application is loaded automatically when Essbase is started (ESS_TRUE if the application will be automatically loaded)
<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_ACCESS_T</td>
<td>Access</td>
<td>The default access to databases in the application (the lowest possible level of access for all users). This field can contain the following values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_PRIV_NONE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_PRIV_DBDESIGN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_PRIV_CALC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_PRIV_WRITE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_PRIV_READ</td>
</tr>
<tr>
<td>ESS_BOOL_T</td>
<td>Connects</td>
<td>Flag to indicate whether users can connect to the application (ESS_TRUE if users can connect).</td>
</tr>
<tr>
<td>ESS_BOOL_T</td>
<td>Commands</td>
<td>Flag to indicate whether users can issue commands to the application (ESS_TRUE if the application is accepting user commands).</td>
</tr>
<tr>
<td>ESS_BOOL_T</td>
<td>Updates</td>
<td>Flag to indicate whether users can update data in the application (ESS_TRUE if the application is accepting user update commands).</td>
</tr>
<tr>
<td>ESS_BOOL_T</td>
<td>Security</td>
<td>Flag to indicate whether application security is enabled (ESS_TRUE if security is enabled).</td>
</tr>
<tr>
<td>ESS_ULONG_T</td>
<td>LockTimeout</td>
<td>Timeout period (in seconds) after which block-level locks are automatically removed. This field does not apply to aggregate storage databases.</td>
</tr>
<tr>
<td>ESS_ULONG_T</td>
<td>LroSizeLimit</td>
<td>Limit on the size of LRO files. This limit is set for each application and enables the administrator or program to protect the server from overly large linked files. Essbase itself does not limit the size or have a default value. This limit does not apply to LRO URLs (limited to 512 characters) or to LRO cell notes (limited to 599 characters). This field does not apply to aggregate storage databases.</td>
</tr>
</tbody>
</table>

**ESS_ATTRIBUTEINFO_T**

Contains attribute information on a specific member. It is used by `EssGetAttributeInfo`.

```c
typedef struct ESS_ATTRIBUTEINFO_T
{
    ESS_MBRNAME_T         MbrName;
    ESS_MBRNAME_T         DimName;
    ESS_ATTRIBUTEVALUE_T  Attribute;
} ESS_ATTRIBUTEINFO_T, *ESS_PATTRIBUTEINFO_T, **ESS_PPATTRIBUTEINFO_T;
```

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_MBRNAME_T</td>
<td>MbrName</td>
<td>Attribute member name from “ESS_MEMBERINFO_T” on page 149 or “ESS_MBRINFO_T” on page 699, including a long name</td>
</tr>
<tr>
<td>ESS_MBRNAME_T</td>
<td>DimName</td>
<td>Attribute dimension name</td>
</tr>
<tr>
<td>“ESS_ATTRIBUTEVALUE_T” on page 118</td>
<td>Attribute</td>
<td>Attribute value</td>
</tr>
</tbody>
</table>

**ESS_ATTRIBUTEVALUE_T**

Contains information on the type and value of attribute members.
typedef struct ESS_ATTRIBUTEVALUE_T
{
    ESS_USHORT_T                usDataType;
    union
    {
        ESS_BOOL_T      bData;
        ESS_STR_T       strData;
        ESS_DATETIME_T  dtData;
        ESS_DOUBLE_T    dblData;
    }                           value;
} ESS_ATTRIBUTEVALUE_T, *ESS_PATTRIBUTEVALUE_T, **ESS_PPATTRIBUTEVALUE_T;

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_USHORT_T</td>
<td>usDataType</td>
<td>A constant identifier indicating the data type of an attribute dimension or member.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>One of the following values for an attribute dimension or zero-level (leaf node) attribute member:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESS_ATTRMBRDT_BOOL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESS_ATTRMBRDT_STRING</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESS_ATTRMBRDT_DATETIME</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESS_ATTRMBRDT_DOUBLE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• One of the following values for an attribute member, but not an attribute dimension:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESS_ATTRMBRDT_NONE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESS_ATTRMBRDT_AUTO</td>
</tr>
<tr>
<td>ESS_BOOL_T</td>
<td>value</td>
<td>A union variable for the following attribute member values:</td>
</tr>
<tr>
<td>ESS_STR_T</td>
<td>value.bData</td>
<td>• Boolean value</td>
</tr>
<tr>
<td>ESS_DATETIME_T</td>
<td>value.strData</td>
<td>• String value</td>
</tr>
<tr>
<td>ESS_DOUBLE_T</td>
<td>value.dtData</td>
<td>• Date and time value</td>
</tr>
<tr>
<td></td>
<td>value.dblData</td>
<td>• Double value</td>
</tr>
</tbody>
</table>

**ESS_ATTRSPECS_T**

Used by EssOtlSetAttributeSpecifications() to set attribute specifications for the outline, and by EssOtlGetAttributeSpecifications() and EssGetAttributeSpecifications() to get attribute specifications for the outline.

typedef struct ESS_ATTRSPECS_T
{
    ESS_USHORT_T  usGenNameBy;
    ESS_USHORT_T  usUseNameOf;
    ESS_CHAR_T    cDelimiter;
    ESS_USHORT_T  usDateFormat;
    ESS_USHORT_T  usBucketingType;
    ESS_STR_T     pszDefaultTrueString;
    ESS_STR_T     pszDefaultFalseString;
    ESS_STR_T     pszDefaultAttrCalcDimName;
    ESS_STR_T     pszDefaultSumMbrName;
    ESS_STR_T     pszDefaultCountMbrName;
    ESS_STR_T     pszDefaultAverageMbrName;
    ESS_STR_T     pszDefaultMinMbrName;
}
```c

Data Type | Field | Description
----------|-------|--------------
ESS_USHORT_T | usGenNameBy | A constant identifier indicating whether to use the generation(s) of the zero-level member as the prefix or the suffix when generating a long name:
- ESS_GENNAMEBY_PREFIX (the default value)
- ESS_GENNAMEBY_SUFFIX

ESS_USHORT_T | usUseNameOf | A constant identifier indicating which generation(s) of the zero-level member to use when generating a long name:
- ESS_USENAMEOF_NONE (the default value)
- ESS_USENAMEOF_PARENT
- ESS_USENAMEOF_GRANDPARENTANDPARENT
- ESS_USENAMEOF_ALLANCESTORS
- ESS_USENAMEOF_DIMENSION

ESS_UCHAR_T | cDelimiter | A constant identifier indicating the delimiter to use when generating a long name:
- ESS_DELIMITER_UNDERSCORE (the default value)
- ESS_DELIMITER_PIPE
- ESS_DELIMITER_CARET

ESS_USHORT_T | usDateFormat | A constant identifier indicating the format for a datetime attribute:
- ESS_DATEFORMAT_MMDDYYYY (the default value)
- ESS_DATEFORMAT_DDMMYYYY

ESS_USHORT_T | usBucketingType | A constant identifier indicating a numeric attribute's bucketing type:
- ESS_UPPERBOUNDINCLUSIVE (the default value)
- ESS_UPPERBOUNDNONINCLUSIVE
- ESS_LOWERBOUNDINCLUSIVE
- ESS_LOWERBOUNDNONINCLUSIVE

ESS_STR_T | pszDefaultTrueString | The string used with the boolean attribute to indicate TRUE. The default value is ESS_DEFAULT_TRUESTRING ("True").

ESS_STR_T | pszDefaultFalseString | The string used with the boolean attribute to indicate FALSE. The default value is ESS_DEFAULT_FALSESTRING ("False").

ESS_STR_T | pszDefaultAttrCalcDimName | The name of the attribute calculations (aggregate) dimension. The default value is ESS_DEFAULT_ATTRIBUTECALCULATIONS ("Attribute Calculations").

ESS_STR_T | pszDefaultSumMbrName | The name used with the attribute calculations (aggregate) dimension to indicate SUM. The default value is ESS_DEFAULT_SUM ("Sum").

ESS_STR_T | pszDefaultCountMbrName | The name used with the attribute calculations (aggregate) dimension to indicate COUNT. The default value is ESS_DEFAULT_COUNT ("Count").

ESS_STR_T | pszDefaultAverageMbrName | The name used with the attribute calculations (aggregate) dimension to indicate AVERAGE. The default value is ESS_DEFAULT_AVERAGE ("Average").
```
<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_STR_T</td>
<td>pszDefaultMinMbrName</td>
<td>The name used with the attribute calculations (aggregate) dimension to indicate MINIMUM. The default value is ESS_DEFAULT_MIN (&quot;Min&quot;).</td>
</tr>
<tr>
<td>ESS_STR_T</td>
<td>pszDefaultMaxMbrName</td>
<td>The name used with the attribute calculations (aggregate) dimension to indicate MAXIMUM. The default value is ESS_DEFAULT_MAX (&quot;Max&quot;).</td>
</tr>
</tbody>
</table>

**ESS_BLDDL_STATE_T**

Contains information about dimension-build and data-load progress.

typedef struct ESS_BLDDL_STATE_T
{
    ESS_USHORT_T usProcessState;
    ESS_USHORT_T usProcessStage;
    ESS_LONG_T ilProcessStatus;
    ESS_ULONG_T ulRecordsProcessed;
    ESS_ULONG_T ulRecordsRejected;
} ESS_BLDDL_STATE_T, *ESS_PBLDDL_STATE_T;

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_USHORT_T</td>
<td>usProcessState</td>
<td>The state of dimension build/data load process: whether it is in progress, in the final stages, or completed. For values, see &quot;Constant Values for usProcessState.&quot;</td>
</tr>
<tr>
<td>ESS_USHORT_T</td>
<td>usProcessStage</td>
<td>The stage of the dimension build/data load process: whether opening the data source, reading the outline, building dimensions, verifying an outline, or writing an outline. For values, see &quot;Constant Values for usProcessStage.&quot;</td>
</tr>
<tr>
<td>ESS_LONG_T</td>
<td>ilProcessStatus</td>
<td>The status of the dimension build/data load process (same as function return status)</td>
</tr>
<tr>
<td>ESS_ULONG_T</td>
<td>ulRecordsProcessed</td>
<td>The number of data records processed so far</td>
</tr>
<tr>
<td>ESS_ULONG_T</td>
<td>ulRecordsRejected</td>
<td>The number of data records rejected so far</td>
</tr>
</tbody>
</table>

**Constant Values for usProcessState**

#define ESS_BLDDL_STATE_DONE 0 /* No process, or process complete */
#define ESS_BLDDL_STATE_INPROGRESS 1 /* Process is in progress */
#define ESS_BLDDL_STATE_FINALSTAGE 5 /* Process at final stage */

**Constant Values for usProcessStage**

#define ESS_BLDDL_STAGE_NONE 0 /* No process */
#define ESS_BLDDL_STAGE_OPENDATASOURCE 1 /* Process at opening data source */
#define ESS_BLDDL_STAGE_OPENOTL 2 /* Process at reading outline */
#define ESS_BLDDL_STAGE_BUILDOTL 3 /* Process at building dimension */
#define ESS_BLDDL_STAGE_VERIFYOTL 4 /* Process at verifying outline */
#define ESS_BLDDL_STAGE_WRITEOTL 5 /* Process at writing outline */
#define ESS_BLDDL_STAGE_RESTRUCT 6 /* Process at restructuring database */
#define ESS_BLDDL_STAGE_DATALOAD 7  /* Process at loading data */
#define ESS_BLDDL_STAGE_FINALIZE 8  /* Process at finalizing*/

ESS_CONNECTINFO_T

Stores information about the processes connected to a specific server.

typedef struct ESS_CONNECTINFO_T
{
    ESS_USERNAME_T Name;          /* logged in user name */
    ESS_APPNAME_T  AppName;       /* connected application */
    ESS_DBNAME_T   DbName;        /* connected database */
    ESS_SVRNAME_T  LoginMachine;  /* login machine name */
    ESS_ULONG_T    LoginIP;       /* IPv4 address of the login machine */
    ESS_TIME_T     LastLogin;     /* login time */
} ESS_CONNECTINFO_T, *ESS_CONNECTINFO_T, **ESS_CONNECTINFO_T;

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_USERNAME_T</td>
<td>Name</td>
<td>The name of the logged in user.</td>
</tr>
<tr>
<td>ESS_APPNAME_T</td>
<td>AppName</td>
<td>Name of currently connected application (if applicable).</td>
</tr>
<tr>
<td>ESS_DBNAME_T</td>
<td>DbName</td>
<td>Name of the currently connected database (if applicable).</td>
</tr>
<tr>
<td>ESS_SVRNAME_T</td>
<td>LoginMachine</td>
<td>The name of the logged in machine. If the machine name cannot be resolved on the network this field contains the IP address formatted as a string. An asterisk (*) denotes the session which called EssListLogins.</td>
</tr>
<tr>
<td>ESS_ULONG_T</td>
<td>LoginIP</td>
<td>The IP address of the logged in machine.</td>
</tr>
<tr>
<td>ESS_TIME_T</td>
<td>LastLogin</td>
<td>The time of the last login.</td>
</tr>
</tbody>
</table>

ESS_CONNECTINFOEX_T

Stores information about the processes connected to a specific server. This structure is similar to ESS_CONNECTINFO_T, with the addition of the ProviderName and connparam fields.

typedef struct ESS_CONNECTINFOEX_T
{
    ESS_USERNAME_T Name;
    ESS_USERNAME_T ProviderName;
    ESS_CONNPARAM_T connparam;
    ESS_APPNAME_T  AppName;
    ESS_DBNAME_T   DbName;
    ESS_SVRNAME_T  LoginMachine;
    ESS_ULONG_T    LoginIP;
    ESS_TIME_T     LastLogin;
} ESS_CONNECTINFOEX_T, *ESS_PCONNECTINFOEX_T, **ESS_PPCONNECTINFOEX_T;
<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_USERNAME_T</td>
<td>Name</td>
<td>Name of the logged in user</td>
</tr>
<tr>
<td>ESS_USERNAME_T</td>
<td>ProviderName</td>
<td>Name of the user directory. Example: @Native Directory</td>
</tr>
<tr>
<td>ESS_CONNPARAM_T</td>
<td>connparam</td>
<td>Unique identity attribute identifying user or group in a directory. Example: native://nvid=f0ed2a6d7fb07688:5a342200:1265973105c:-7f46?USER</td>
</tr>
<tr>
<td>ESS_APPNAME_T</td>
<td>AppName</td>
<td>Name of the currently connected application (if applicable)</td>
</tr>
<tr>
<td>ESS_DBNAME_T</td>
<td>DbName</td>
<td>Database name</td>
</tr>
<tr>
<td>ESS_SVRNAME_T</td>
<td>LoginMachine</td>
<td>Name of the logged in machine. If the machine name cannot be resolved on the network, this field contains the IP address formatted as a string. An asterisk (*) denotes the session that called EssListLogins.</td>
</tr>
<tr>
<td>ESS_UULONG_T</td>
<td>LoginIP</td>
<td>IP address of the logged in machine</td>
</tr>
<tr>
<td>ESS_TIME_T</td>
<td>LastLogin</td>
<td>Time of the last login</td>
</tr>
</tbody>
</table>

**ESS_DBFILEINFO_T**

Contains information on an index or data file retrieved by EssListDbFiles.

typedef struct ess_dbfileinfo_t{
    ESS_APPNAME_T AppName;
    ESS_DBNAME_T DbName;
    ESS_FILENAME_T FilePath;
    ESS_SIZE_T FileSize;
    ESS_USHORT_T FileSequenceNum;
    ESS_USHORT_T FileCount;
    ESS_USHORT_T FileType;
    ESS_BOOL_T FileOpen;
} ESS_DBFILEINFO_T, *ESS_PDBFILEINFO_T, **ESS_PPDBFILEINFO_T;

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_APPNAME_T</td>
<td>AppName</td>
<td>Application name</td>
</tr>
<tr>
<td>ESS_DBNAME_T</td>
<td>DbName</td>
<td>Database name</td>
</tr>
<tr>
<td>ESS_FILENAME_T</td>
<td>FilePath</td>
<td>File path</td>
</tr>
<tr>
<td>ESS_SIZE_T</td>
<td>FileSize</td>
<td>File size in bytes</td>
</tr>
<tr>
<td>ESS_USHORT_T</td>
<td>FileSequenceNum</td>
<td>The 1-based sequence number of the file within the set of files of its FileType for the specified database</td>
</tr>
<tr>
<td>ESS_USHORT_T</td>
<td>FileCount</td>
<td>Number of files of its FileType returned</td>
</tr>
</tbody>
</table>
### Data Type | Field | Description
--- | --- | ---
ESS_USHORT_T | FileType | One of the following file types:
- ESS_FILETYPE_INDEX
- ESS_FILETYPE_DATA

ESS_BOOL_T | FileOpen | Flag indicating whether the file is open: 0 if the file is closed, nonzero if the file is open

---

**ESS_DBINFO_T**

This databaseInfo Structure gets information about a specific database. Fields in this structure cannot be modified using the API. See also the “ESS_DBSTATE_T” on page 127 structure, which contains additional database state parameters that can be modified, and the “ESS_DBSTATS_T” on page 130 structure. The fields are:

```c
typedef struct ESS_DBINFO_T
{
    ESS_APPNAME_T      AppName;
    ESS_DBNAME_T       Name;
    ESS_USHORT_T       DbType;
    ESS_USHORT_T       Status;
    ESS_USHORT_T       nConnects;
    ESS_USHORT_T       nLocks;
    ESS_U_LONG_T       nDims;
    ESS_USHORT_T       Data;
    ESS_MBRNAME_T      Country;
    ESS_MBRNAME_T      Time;
    ESS_MBRNAME_T      Category;
    ESS_MBRNAME_T      Type;
    ESS_MBRNAME_T      CrPartition;
    ESS_TIME_T         ElapsedDbTime;
    ESS_U_LONG_T       DataFileCacheSetting;
    ESS_U_LONG_T       DataFileCacheSize;
    ESS_U_LONG_T       DataCacheSetting;
    ESS_U_LONG_T       DataCacheSize;
    ESS_U_LONG_T       IndexCacheSetting;
    ESS_U_LONG_T       IndexCacheSize;
    ESS_U_LONG_T       IndexPageSetting;
    ESS_U_LONG_T       IndexPageSize;
    ESS_DBREQINFO_T    DbReqInfoAry[ESS_DBREQNUM];
    ESS_BOOL_T         bDbReadOnly;
    ESS_BOOL_T         bDataCompress;
    ESS_USHORT_T       usDataCompressType;
    ESS_U_LONG_T       ulRetrievalBuffer;
    ESS_U_LONG_T       ulRetrievalSortBuffer;
    ESS_BOOL_T         bCacheMemLocking;
    ESS_BOOL_T         bPreImage;
    ESS_USHORT_T       usIsolationLevel;
    ESS_LONG_T         lTimeOut;
    ESS_U_LONG_T       ulCommitBlocks;
    ESS_U_LONG_T       ulCommitRows;
    ESS_U_LONG_T       ulDiskVolumeCount;
    ESS_DISKVOLUME_T   aDiskVolume[1];
};
```
<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_APPNAME_T</td>
<td>AppName</td>
<td>The associated application name</td>
</tr>
<tr>
<td>ESS_DBNAME_T</td>
<td>Name</td>
<td>The database name</td>
</tr>
<tr>
<td>ESS_USHORT_T</td>
<td>DbType</td>
<td>Database type. Values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_DBTYPE_NORMAL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_DBTYPE_CURRENCY</td>
</tr>
<tr>
<td>ESS_USHORT_T</td>
<td>Status</td>
<td>DatabaseLoad status (loaded or not loaded). Values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_STATUS_NOTLOADED</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_STATUS_LOADING</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_STATUS_LOADED</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_STATUS_UNLOADING</td>
</tr>
<tr>
<td>ESS_USHORT_T</td>
<td>nConnects</td>
<td>Number of users currently connected to the database</td>
</tr>
<tr>
<td>ESS_USHORT_T</td>
<td>nLocks</td>
<td>Number of data blocks currently exclusively locked</td>
</tr>
<tr>
<td>ESS_ULONG_T</td>
<td>nDims</td>
<td>Number of dimensions in database</td>
</tr>
<tr>
<td>ESS_USHORT_T</td>
<td>Data</td>
<td>Flag indicating loading state of the data in the database. Values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_DBDATA_NONE: no data loaded</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_DBDATA_LOADNOCALC: data loaded but not calculated</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_DBDATA_CLEAN: data loaded and calculated</td>
</tr>
<tr>
<td>ESS_MBRNAME_T</td>
<td>Country</td>
<td>The currency country dimension member, if any. If none, the first byte is NULL.</td>
</tr>
<tr>
<td>ESS_MBRNAME_T</td>
<td>Time</td>
<td>Currency time dimension member, if any. If none, the first byte is NULL.</td>
</tr>
<tr>
<td>ESS_MBRNAME_T</td>
<td>Category</td>
<td>The currency category dimension member, if any. If none, the first byte is NULL.</td>
</tr>
<tr>
<td>ESS_MBRNAME_T</td>
<td>Type</td>
<td>Currency type dimension member (currency databases only). If none exists, the first byte is NULL.</td>
</tr>
<tr>
<td>ESS_MBRNAME_T</td>
<td>CrPartition</td>
<td>The currency partition member (non-currency databases only)</td>
</tr>
<tr>
<td>ESS_TIME_T</td>
<td>ElapsedTime</td>
<td>Number of seconds the database has been loaded</td>
</tr>
<tr>
<td>ESS_ULONG_T</td>
<td>DataFileCacheSetting</td>
<td>The Data File Cache Size setting value currently in effect.</td>
</tr>
<tr>
<td>ESS_ULONG_T</td>
<td>DataFileCacheSize</td>
<td>The Run-time data file cache size (in KB) currently in use by database. Note that once you have changed the data file cache size you must stop and restart the database in order for the new data file cache size to take effect.</td>
</tr>
<tr>
<td>Data Type</td>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ESS_ULONG_T</td>
<td>DataCacheSetting</td>
<td>The Data Cache Size setting value (in KB) currently in effect.</td>
</tr>
<tr>
<td>ESS_ULONG_T</td>
<td>DataCacheSize</td>
<td>The run-time size (in KB) of the Data Cache.</td>
</tr>
<tr>
<td>ESS_ULONG_T</td>
<td>IndexCacheSetting</td>
<td>The Index Cache Size (in KB) setting value currently in effect.</td>
</tr>
<tr>
<td>ESS_ULONG_T</td>
<td>IndexCacheSize</td>
<td>Run-time size (in KB) of the Index Cache.</td>
</tr>
<tr>
<td>ESS_ULONG_T</td>
<td>IndexPageSetting</td>
<td>The Index Page Size setting (in KB) currently in effect.</td>
</tr>
<tr>
<td>ESS_ULONG_T</td>
<td>IndexPageSize</td>
<td>Run-time size (in KB) of an Index Page.</td>
</tr>
</tbody>
</table>

“ESS_DBREQINFO_T” on page 126

<table>
<thead>
<tr>
<th>ESS_BOOL_T</th>
<th>bDbReadOnly</th>
<th>TRUE if the database is in read-only mode; FALSE otherwise.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_BOOL_T</td>
<td>bDataCompress</td>
<td>Optional compression flag (the default is YES).</td>
</tr>
<tr>
<td>ESS_USHORT_T</td>
<td>usDataCompressType</td>
<td>The data compression type if the optional compression flag is set (the default is BitMap).</td>
</tr>
<tr>
<td>ESS_ULONG_T</td>
<td>uiRetrievalBuffer</td>
<td>Retrieval buffer size allocated per retrieval request (the default is 2048 bytes).</td>
</tr>
<tr>
<td>ESS_ULONG_T</td>
<td>ulRetrievalSortBuffer</td>
<td>Retrieval sort buffer size allocated per retrieval request (the default is 10240 bytes).</td>
</tr>
<tr>
<td>ESS_BOOL_T</td>
<td>bCacheMemLocking</td>
<td>TRUE if index and data cache memory pages are locked into physical memory.</td>
</tr>
<tr>
<td>ESS_BOOL_T</td>
<td>bPreImage</td>
<td>Flag to read previously committed.</td>
</tr>
<tr>
<td>ESS_USHORT_T</td>
<td>usIsolationLevel</td>
<td>Isolation level (the default is UNCOMMITTED).</td>
</tr>
<tr>
<td>ESS_LONG_T</td>
<td>lTimeOut</td>
<td>Time out set in seconds for COMMITTED access only.</td>
</tr>
<tr>
<td>ESS_ULONG_T</td>
<td>ulCommitBlocks</td>
<td>The number of data blocks updated before the explicit commit is performed (during calculation and spreadsheet updates).</td>
</tr>
<tr>
<td>ESS_ULONG_T</td>
<td>ulCommitRows</td>
<td>The number of rows of the input file processed before the explicit commit is performed during the dataload.</td>
</tr>
<tr>
<td>ESS_ULONG_T</td>
<td>ulDiskVolumeCount</td>
<td>The number of disk volume settings for this database.</td>
</tr>
<tr>
<td>ESS_DISKVOLUME_T</td>
<td>aDiskVolume[1]</td>
<td>an array of disk volume settings</td>
</tr>
</tbody>
</table>

**ESS_DBREQINFO_T**

Used by `EssGetDatabaseInfo()`. Essbase has three types of requests for which information exists: data load, calculation, and outline update. The following Essbase API constants identify each type of request:
typedef struct ESS_DBREQINFO_T
{
    ESS_ULONG_T      DbReqType;
    ESS_USERNAME_T   User;
    ESS_TIMERECORD_T StartTimeRec;
    ESS_TIMERECORD_T EndTimeRec;
    ESS_ULONG_T      DbReqFlags;
} ESS_DBREQINFO_T, *ESS_PDBREQINFO_T;

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_DBREQTYPE_DATLOAD</td>
<td>0</td>
<td>Data Load</td>
</tr>
<tr>
<td>ESS_DBREQTYPE_CALC</td>
<td>1</td>
<td>Calculation</td>
</tr>
<tr>
<td>ESS_DBREQTYPE_OTLUPD</td>
<td>2</td>
<td>Outline Update</td>
</tr>
</tbody>
</table>

The fields are:

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_ULONG_T</td>
<td>DbReqType</td>
<td>Type of database request</td>
</tr>
<tr>
<td>ESS_USERNAME_T</td>
<td>User</td>
<td>User name</td>
</tr>
<tr>
<td>&quot;ESS_TIMERECORD_T&quot; on page 186</td>
<td>StartTimeRec</td>
<td>Request start time</td>
</tr>
<tr>
<td>&quot;ESS_TIMERECORD_T&quot; on page 186</td>
<td>EndTimeRec</td>
<td>Request end time</td>
</tr>
</tbody>
</table>
| ESS_ULONG_T     | DbReqFlags     | Bit map of information flags that provide additional information about the database request. Used when DbReqType is CALC. Available flags:
|                 |                | • Default (currently contains no information). ESS API constant: ESS_DBREQFLAG_CALCDEF (default calc was run) |
|                 |                | • Custom calc script. ESS API constant: ESS_DBREQFLAG_CALCSRC (custom calc was run) |

**ESS_DBSTATE_T**

This database state structure gets and sets the state parameters for a specific database. All fields in this structure can be modified using the API. See also the “ESS_DBINFO_T” on page 124 and “ESS_DBSTATS_T” on page 130 structures, which contain additional database information that cannot be modified.

typedef struct ESS_DBSTATE_T
{
    ESS_DESC_T        Description;
    ESS_BOOL_T        Loadable;
    ESS_BOOL_T        Autoload;
    ESS_ACCESS_T      Access;
    ESS_SHORT_T       IndexType;
    ESS_ULONG_T       MaxMem;
    ESS_ULONG_T       MaxCompMem;
}
The fields are:

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_DESC_T</td>
<td>Description</td>
<td>The database description (up to 80 characters)</td>
</tr>
<tr>
<td>ESS_BOOL_T</td>
<td>Loadable</td>
<td>Flag to indicate whether the database can be loaded (ESS_TRUE if the database is loadable)</td>
</tr>
<tr>
<td>ESS_BOOL_T</td>
<td>Autoload</td>
<td>Flag to indicate whether the database will be loaded automatically be loaded when the application is started (ESS_TRUE if the database will be automatically loaded)</td>
</tr>
<tr>
<td>ESS_ACCESS_T</td>
<td>Access</td>
<td>The default access level to the database. See &quot;Bitmask Data Types (C)&quot; on page 96 for a list of values this field can contain.</td>
</tr>
<tr>
<td>ESS_USHORT_T</td>
<td>IndexType</td>
<td>The database index type (array or tree). This field can contain the following values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESS_INDEXTYPE_ARRAY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESS_INDEXTYPE_AVL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For API releases 4 and later, the IndexType field is obsolete.</td>
</tr>
<tr>
<td>ESS ULONG_T</td>
<td>MaxMem</td>
<td>The maximum memory reserved for non-compressed data blocks in the database(in bytes)</td>
</tr>
<tr>
<td>ESS ULONG_T</td>
<td>MaxCompMem</td>
<td>The maximum memory reserved for compressed data blocks in database(in bytes)</td>
</tr>
<tr>
<td>ESS_BOOL_T</td>
<td>CalcNoAggMissing</td>
<td>Flag to suppress aggregation of members if all their children are missing (ESS_TRUE if missing values are not aggregated)</td>
</tr>
<tr>
<td>ESS_BOOL_T</td>
<td>CalcNoAvgMissing</td>
<td>Flag to suppress inclusion of missing members in calculating averages (ESS_TRUE if missing values are not included)</td>
</tr>
<tr>
<td>Data Type</td>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ESS_BOOL_T</td>
<td>CalcTwoPass</td>
<td>Flag to force two pass calculation when running full calculation of database (ESS_TRUE if two pass calculation is enabled)</td>
</tr>
<tr>
<td>ESS_BOOL_T</td>
<td>CalcCreateBlock</td>
<td>Flag to force creation of data block on constant assignment calc equation (only valid for sparse dimensions). Set to ESS_TRUE if blocks are forcibly created.</td>
</tr>
<tr>
<td>ESS_DBNAME_T</td>
<td>CrDbName</td>
<td>The name of associated currency database (valid in non-currency databases).</td>
</tr>
<tr>
<td>ESS_MBRNAME_T</td>
<td>CrTypeMember</td>
<td>The name of Currency Conversion type member (valid in non-currency databases)</td>
</tr>
</tbody>
</table>
| ESS_USHORT_T| CrConvType       | Currency Conversion type (whether currency conversions are calculated by multiplication or division). Values:
  * ESS_CRCTYPE_DIV
  * ESS_CRCTYPE_MULT                                                                 |
| ESS_ULONG_T | MaxMemIndex      | Minimum index cache size. Value: 1048576. Set using the constant ESS_INDEXCACHEMIN_SIZE                                                                                                                   |
| ESS_ULONG_T | IndexPageSize    | Size of index page in which buffer pool is constructed in (in bytes). Minimum index page size. Value: 1024. Set using the constant ESS_INDEXPAGEMIN_SIZE
  Maximum page size for the IndexPageSize field. Value: 8192. Set using the constant ESS_INDEXPAGEMAX_SIZE |
| ESS_BOOL_T  | DataCompress     | Optional Flag to determine whether to compress blocks for this database.                                                                                                                                 |
| ESS_USHORT_T| DataCompressType | The data compression type used for write operations if the optional compression flag is set.
  * Bitmap—Uses a bitmap to represent data cells (the default).
  * Run-Length Encoding—Compresses any consecutive repetitive values.
  * No Compression—Does not compress the data. |
<p>| ESS_ULONG_T | RetrievalBuffer  | Specifies the size, in bytes, of the server buffer that holds extracted row data cells before they are evaluated by the RESTRICT, TOP, or BOTTOM commands. The default is 10240 bytes. The minimum is 2048 bytes and the maximum is 102400000 bytes. |
| ESS_ULONG_T | RetrievalSortBuffer | Specifies the size, in bytes, of the server buffer that holds the data to be sorted during a retrieval. The minimum is 2048 bytes and the maximum is 102400000 bytes.                       |
| ESS_BYTE_T  | clIOAccessFlagInUse | The type of I/O Access in use by the active current database. The two types of access are ESS_IO_ACCESS_BUFFERED and ESS_IO_ACCESS_DIRECT. Even when clIOAccessFlagPending has been set to ESS_IO_ACCESS_DIRECT, some operations might still require buffering. Also direct access may not be supported on a given platform. This field is read only. |
| ESS_BOOL_T  | bNoWaitIO        | This controls whether or not Essbase will wait for certain direct I/O operations to finish. This only applies on platforms that support direct I/O and if clIOAccessFlag is ESS_IO_ACCESS_DIRECT. This field is read only. The default is TRUE. |</p>
<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_USHORT_T</td>
<td>IsolationLevel</td>
<td>The isolation level:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- COMMITTED—Write locks on all affected data blocks restrict access until the transaction commits.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- UNCOMMITTED (default)—Write locks are acquired and released as needed during the transaction.</td>
</tr>
<tr>
<td>ESS_BOOL_T</td>
<td>PreImage</td>
<td>The flag to read previously committed data during read-only requests. This flag can only be set for COMMITTED access. The default is YES.</td>
</tr>
<tr>
<td>ESS_BYTE_T</td>
<td>cIOMode</td>
<td>The type of I/O Access (direct or buffered) that Essbase will use. This setting takes effect after the next DBLoad (open operation).</td>
</tr>
<tr>
<td>ESS_LONG_T</td>
<td>TimeOut</td>
<td>The timeout interval in seconds. This can only be set for COMMITTED access.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-1 is Indefinite wait.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 is Immediate access, no wait (the default).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>n is the specified interval in seconds.</td>
</tr>
<tr>
<td>ESS_ULONG_T</td>
<td>CommitBlocks</td>
<td>The number of data blocks modified before performing the explicit commit (only used when isolation level is UNCOMMITTED).</td>
</tr>
<tr>
<td>ESS_ULONG_T</td>
<td>CommitRows</td>
<td>The number of rows of the input file to data load before performing the explicit commit (only used when isolation level is UNCOMMITTED).</td>
</tr>
<tr>
<td>ESS_ULONG_T</td>
<td>nVolumes</td>
<td>The number of disk volume settings for this database.</td>
</tr>
</tbody>
</table>

**ESS_DBSTATS_T**

This database statistics structure gets run-time statistical information about a specific database. Fields in this structure cannot be modified using the API. See also the “ESS_DBSTATE_T” on page 127 structure, which contains additional database state parameters that can be modified, and the “ESS_DBINFO_T” on page 124 structure. The fields are:

```c
typedef struct ESS_DBSTATS_T
{
    ESS_USHORT_T      IndexType;
    ESS_ULONG_T       nDims;
    ESS_ULONG_T       DeclaredBlockSize;
    ESS_ULONG_T       ActualBlockSize;
    ESS_DOUBLE_T       DeclaredMaxBlocks;
    ESS_DOUBLE_T       ActualMaxBlocks;
    ESS_DOUBLE_T       NonMissingLeafBlocks;
    ESS_DOUBLE_T       NonMissingNonLeafBlocks;
    ESS_DOUBLE_T       NonMissingBlocks;
    ESS_DOUBLE_T       PagedOutBlocks;
    ESS_DOUBLE_T       PagedInBlocks;
    ESS_DOUBLE_T       InMemCompBlocks;
    ESS_DOUBLE_T       TotalBlocks;
    ESS_DOUBLE_T       AverageFragmentationQuotient;
    ESS_DOUBLE_T       BytesOfRecoverableFreeSpace;
} ESS_DBSTATS_T;
```
<table>
<thead>
<tr>
<th><strong>Data Type</strong></th>
<th><strong>Field</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_USHORT_T</td>
<td>IndexType</td>
<td>The database index type (array or tree). This field can contain the following values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESS_INDEXTYPE_ARRAY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESS_INDEXTYPE_AVL</td>
</tr>
<tr>
<td>ESS_ULONG_T</td>
<td>nDims</td>
<td>The number of dimensions in database.</td>
</tr>
<tr>
<td>ESS_ULONG_T</td>
<td>DeclaredBlockSize</td>
<td>The declared data block size.</td>
</tr>
<tr>
<td>ESS_ULONG_T</td>
<td>ActualBlockSize</td>
<td>The actual data block size</td>
</tr>
<tr>
<td>ESS_DOUBLE_T</td>
<td>DeclaredMaxBlocks</td>
<td>The declared maximum number of blocks in the database.</td>
</tr>
<tr>
<td>ESS_DOUBLE_T</td>
<td>ActualMaxBlocks</td>
<td>The actual maximum number of blocks in the database.</td>
</tr>
<tr>
<td>ESS_DOUBLE_T</td>
<td>NonMissingLeafBlocks</td>
<td>The number of non-missing leaf (lowest level) blocks in the database.</td>
</tr>
<tr>
<td>ESS_DOUBLE_T</td>
<td>NonMissingNonLeafBlocks</td>
<td>The number of non-missing, non-leaf (upper level) blocks in the database.</td>
</tr>
<tr>
<td>ESS_DOUBLE_T</td>
<td>NonMissingBlocks</td>
<td>Obsolete. Returns zero.</td>
</tr>
<tr>
<td>ESS_DOUBLE_T</td>
<td>PagedOutBlocks</td>
<td>The number of database blocks currently paged out to disk.</td>
</tr>
<tr>
<td>ESS_DOUBLE_T</td>
<td>PagedInBlocks</td>
<td>The total number of database blocks currently paged into memory.</td>
</tr>
<tr>
<td>ESS_DOUBLE_T</td>
<td>InMemCompBlocks</td>
<td>The number of database blocks currently paged into compressed memory.</td>
</tr>
<tr>
<td>ESS_DOUBLE_T</td>
<td>TotalBlocks</td>
<td>Total number of existing data blocks (not the maximum).</td>
</tr>
<tr>
<td>ESS_DOUBLE_T</td>
<td>AverageFragmentationQuotient</td>
<td>Percentage of space within the data file that is free space or not used by Essbase.</td>
</tr>
<tr>
<td>ESS_DOUBLE_T</td>
<td>BytesOfRecoverableFreeSpace</td>
<td>• Estimated bytes of recoverable free space</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• -1 if free space recovery is not necessary</td>
</tr>
<tr>
<td>ESS_DOUBLE_T</td>
<td>TotMemPagedInBlocks</td>
<td>The total memory used for all paged-in (uncompressed) database blocks.</td>
</tr>
<tr>
<td>ESS_DOUBLE_T</td>
<td>TotMemBlocks</td>
<td>The total memory used for all database blocks.</td>
</tr>
<tr>
<td>ESS_DOUBLE_T</td>
<td>TotMemIndex</td>
<td>The total memory used for the database index.</td>
</tr>
<tr>
<td>Data Type</td>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ESS_DOUBLE_T</td>
<td>TotMemInMemCompBlocks</td>
<td>The total memory used for database blocks currently paged into compressed memory.</td>
</tr>
<tr>
<td>ESS_DOUBLE_T</td>
<td>BlockDensity</td>
<td>The average database block density (calculated using all currently loaded blocks).</td>
</tr>
<tr>
<td>ESS_DOUBLE_T</td>
<td>SparseDensity</td>
<td>Average density of the sparse dimensions in the database.</td>
</tr>
<tr>
<td>ESS_DOUBLE_T</td>
<td>CompressionRatio</td>
<td>Average data block compression ratio on the disk.</td>
</tr>
<tr>
<td>ESS_DOUBLE_T</td>
<td>ClusterRatio</td>
<td>A measure of the fragmentation of the page file. A value close to 1 indicates the degree of fragmentation is low. A value close to zero indicates a high degree of fragmentation that could affect calculation and query performance.</td>
</tr>
</tbody>
</table>

"ESS_DIMSTATS_T" on page 133


**ESS_DIMENSIONINFO_T**

Used in `EssGetDimensionInfo()`. The fields are:

```c
typedef struct ESS_DIMENSIONINFO_T {
    ESS_MBRNAME_T DimName;
    ESS_DIMNUM_T DimNumber;
    ESS_USHORT_T DimType;
    ESS_USHORT_T DimTag;
    ESS_ULONG_T DeclaredDimSize;
    ESS_ULONG_T ActualDimSize;
    ESS_DESC_T Description;
    ESS_USHORT_T DimDataType;
} ESS_DIMENSIONINFO_T, *ESS_PDIMENSIONINFO_T, **ESS_PPDIMENSIONINFO_T;
```

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_MBRNAME_T</td>
<td>DimName</td>
<td>Dimension name</td>
</tr>
<tr>
<td>ESS_DIMNUM_T</td>
<td>DimNumber</td>
<td>Dimension number</td>
</tr>
<tr>
<td>ESS_USHORT_T</td>
<td>DimType</td>
<td>Dimension type. Values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_DIMTYPE_DENSE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_DIMTYPE_SPARSE</td>
</tr>
</tbody>
</table>
**ESS_DIMSTATS_T**

This is a Dimension Statistical Structure used to get information about a specific database dimension. Fields in this structure cannot be modified using the API. An array of these structures is included at the end of the "ESS_DBSTATS_T" on page 130 structure to provide information about each dimension in the database. The fields are:

```c
typedef struct ESS_DIMSTATS_T {
    ESS_MBRNAME_T DimName;
    ESS_USHORT_T DimType;
    ESS_ULONG_T DeclaredDimSize;
    ESS_ULONG_T ActualDimSize;
} ESS_DIMSTATS_T, *ESS_PDIMSTATS_T;
```

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_MBRNAME_T</td>
<td>DimName</td>
<td>The dimension member name</td>
</tr>
<tr>
<td>ESS_USHORT_T</td>
<td>DimType</td>
<td>The dimension type (sparse or dense). This field can contain the following values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_DIMTYPE_SPARSE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_DIMTYPE_DENSE</td>
</tr>
<tr>
<td>ESS_ULONG_T</td>
<td>DeclaredDimSize</td>
<td>The declared dimension size (the number of members declared in the specified dimension, including any label only or shared members in that dimension)</td>
</tr>
<tr>
<td>Data Type</td>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ESS_ULONG_T</td>
<td>ActualDimSize</td>
<td>The actual dimension size (the number of members in the specified dimension, excluding any label only or shared members in that dimension)</td>
</tr>
</tbody>
</table>

### ESS_DTAPICOLUMN_T

Defines the header information for a specific column.

typedef struct ESS_DTAPICOLUMN_T
{
    ESS_LONG_T nColumnIdx;
    ESS_LONG_T nDisplayOrder;
    ESS_CHAR_T sViewName[ESS_MBRNAMELEN];
    ESS_CHAR_T sColumnName[ESS_MBRNAMELEN];
    ESS_USHORT_T uDatatype;
    ESS_LONG_T nSortOrder;
    ESS_LONG_T nSortSequence;
    ESS_BOOL_T bFilterOnly;
    ESS_CHAR_T sFilter[ESS_MAX_DATALEN + 1];
} ESS_DTAPICOLUMN_T, *ESS_PDTAPICOLUMN_T, **ESS_PPDTAPICOLUMN_T;

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_LONG_T</td>
<td>nColumnIdx</td>
<td>0-based index of the column position (read only)</td>
</tr>
<tr>
<td>ESS_LONG_T</td>
<td>nDisplayOrder</td>
<td>The order in which columns are displayed</td>
</tr>
<tr>
<td>ESS_CHAR_T</td>
<td>sViewName</td>
<td>(read only)</td>
</tr>
<tr>
<td>ESS_CHAR_T</td>
<td>sColumnName</td>
<td>Heading text for the given column of data (read only)</td>
</tr>
<tr>
<td>ESS_USHORT_T</td>
<td>uDataType</td>
<td>Data type of the given column of data</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESS_DT_STRING</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESS_DT_DATETIME</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESS_DT_DOUBLE</td>
</tr>
<tr>
<td>ESS_LONG_T</td>
<td>nSortOrder</td>
<td></td>
</tr>
<tr>
<td>ESS_LONG_T</td>
<td>nSortSequence</td>
<td></td>
</tr>
<tr>
<td>ESS_BOOL_T</td>
<td>bFilterOnly</td>
<td>ESS_TRUE = filter only.</td>
</tr>
<tr>
<td>ESS_CHAR_T</td>
<td>sFilter</td>
<td></td>
</tr>
</tbody>
</table>

### ESS_DTAPIDATA_T

Defines the report data for a specific data cell.

typedef struct ESS_DTAPIDATA_T
{
    ESS_ULONG_T nRowIdx;
    ESS_ULONG_T nColumnIdx;
...
ESS_CHAR_T sData[ESS_MAX_DATALEN + 1];
} ESS_DTAPIDATA_T, *ESS_PDTAPIDATA_T, **ESS_PPDTAPIDATA_T;

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_ULONG_T</td>
<td>nRowIdx</td>
<td>0-indexed row number for the given data block</td>
</tr>
<tr>
<td>ESS_ULONG_T</td>
<td>nColumnIdx</td>
<td>0-indexed column number for the given data block</td>
</tr>
<tr>
<td>ESS_CHAR_T</td>
<td>sData [ESS_MAX_DATALEN + 1]</td>
<td>Data value for the given data block</td>
</tr>
</tbody>
</table>

**ESS_DTAPIHEADER_T**

Defines header information for a specific column.

typedef struct __ess_dtapiheader_t
{
    ESS_ULONG_T nColumnIdx;
    ESS_CHAR_T sViewName[ESS_MBRNAMELEN];
    ESS_CHAR_T sColumnName[ESS_MBRNAMELEN];
    ESS_USHORT_T uDatatype;
} ESS_DTAPIHEADER_T, *ESS_PDTAPIHEADER_T, **ESS_PPDTAPIHEADER_T;

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_ULONG_T</td>
<td>nColumnIdx</td>
<td>0-indexed column number for the given data block</td>
</tr>
<tr>
<td>ESS_CHAR_T</td>
<td>sViewName [ESS_MBRNAMELEN]</td>
<td></td>
</tr>
<tr>
<td>ESS_CHAR_T</td>
<td>sColumnName [ESS_DESCRIPTION_LEN + 1]</td>
<td>Data value for the given data block.</td>
</tr>
<tr>
<td>ESS_USHORT_T</td>
<td>uDatatype</td>
<td></td>
</tr>
</tbody>
</table>

**ESS_DTAPIINFO_T**

Defines the connection information for a range of data cells.

typedef struct ESS_DTAPIINFO_T
{
    ESS_CHAR_T sHisName[ESS_MAX_NAME + 1];
    ESS_CHAR_T sUsername[ESS_MAX_NAME + 1];
    ESS_CHAR_T sPassword[ESS_MAX_NAME + 1];
    ESS_USHORT_T uInputOption;
} ESS_DTAPIINFO_T, *ESS_PDTAPIINFO_T, **ESS_PPDTAPIINFO_T;

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_CHAR_T</td>
<td>sHisName [ESS_MAX_NAME + 1]</td>
<td></td>
</tr>
<tr>
<td>ESS_CHAR_T</td>
<td>sUsername [ESS_MAX_NAME + 1]</td>
<td></td>
</tr>
<tr>
<td>ESS_CHAR_T</td>
<td>sPassword [ESS_MAX_NAME + 1]</td>
<td>(write only)</td>
</tr>
</tbody>
</table>
### ESS_DTAPIREPORT_T

Defines the header information for a specific column.

```c
typedef struct ESS_DTAPIREPORT_T {
    ESS_LONG_T nReportId;
    ESS_CHAR_T sName[ESS_DTREPORT_NAME + 1];
    ESS_LONG_T nCustomize;
    ESS_LONG_T nRowGovernor;
    ESS_LONG_T nTimeGovernor;
} ESS_DTAPIREPORT_T, *ESS_PDTAPIREPORT_T, **ESS_PPDTAPIREPORT_T;
```

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_LONG_T</td>
<td>nReportId</td>
<td></td>
</tr>
<tr>
<td>ESS_CHAR_T</td>
<td>sName</td>
<td>[ESS_DTREPORT_NAME + 1]</td>
</tr>
<tr>
<td>ESS_LONG_T</td>
<td>nCustomize</td>
<td></td>
</tr>
<tr>
<td>ESS_LONG_T</td>
<td>nRowGovernor</td>
<td></td>
</tr>
<tr>
<td>ESS_LONG_T</td>
<td>nTimeGovernor</td>
<td></td>
</tr>
</tbody>
</table>

### ESS_DTBUFFER_T

Defines a report data cell.

```c
typedef struct ESS_DTBUFFER_T {
    ESS_ULONG_T row;
    ESS_ULONG_T column;
    ESS_CHAR_T  data[ESS_DESCRIPTION_LEN + 1];
} ESS_DTBUFFER_T, *ESS_PDTBUFFER_T, **ESS_PPDTBUFFER_T;
```

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_ULONG_T</td>
<td>row</td>
<td>0-indexed row number for the given data block.</td>
</tr>
<tr>
<td>ESS_ULONG_T</td>
<td>column</td>
<td>0-indexed column number for the given data block.</td>
</tr>
<tr>
<td>ESS_CHAR_T</td>
<td>data</td>
<td>Data value for the given data block.</td>
</tr>
</tbody>
</table>

### ESS_DTDATA_T

Defines a report data cell.
typedef struct ESS_DTDATA_T
{
    ESS_ULONG_T row;
    ESS_ULONG_T column;
    ESS_CHAR_T  data[ESS_DESCRIPTION_LEN + 1];
} ESS_DTDATA_T, *ESS_PDTDATA_T, **ESS_PPDTDATA_T;

### Data Type
<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_ULONG_T</td>
<td>row</td>
</tr>
<tr>
<td>ESS_ULONG_T</td>
<td>column</td>
</tr>
<tr>
<td>ESS_CHAR_T</td>
<td>data [ESS_DESCRIPTION_LEN + 1]</td>
</tr>
</tbody>
</table>

**ESS_DTHEADER_T**

Defines header information for a specific column.

typedef struct ESS_DTHEADER_T
{
    ESS_ULONG_T colIndex;
    ESSDTREPORTDATATYPE dataType;
    ESS_CHAR_T  data[ESS_DESCRIPTION_LEN + 1];
} ESS_DTHEADER_T, *ESS_PDTHEADER_T, **ESS_PPDTHEADER_T;

### Data Type
<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_ULONG_T</td>
<td>colIndex</td>
</tr>
<tr>
<td>ESSDTREPORTDATATYPE</td>
<td>dataType</td>
</tr>
<tr>
<td>ESS_CHAR_T</td>
<td>data [ESS_DESCRIPTION_LEN + 1]</td>
</tr>
</tbody>
</table>

**ESS_DISKVOLUME_REPLACE_T**

Contains the names of the source and destination disk volume labels. The source currently exists, and will be replaced with the destination.

typedef struct ess_diskvolume_replace_t
{
    ESS_FILENAME_T szPartition_Src;
    ESS_FILENAME_T szPartition_Dest;
} ESS_DISKVOLUME_REPLACE_T;

### Data Type
<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_FILENAME_T</td>
<td>szPartition_Src</td>
</tr>
<tr>
<td>ESS_FILENAME_T</td>
<td>szPartition_Dest</td>
</tr>
</tbody>
</table>
**ESS_DURLINFO_T**
Captures drill-through URL information.

See “Drill-through URL Limits” on page 1729.

typedef struct url
{
    ESS_CHAR_T    bIsLevel0;
    ESS_STR_T     cpURLName;
    ESS_USHORT_T  iURLXmlSize;
    ESS_BYTE_T*   cpURLXml;
    ESS_USHORT_T  iCountOfDrillRegions;
    ESS_PSTR_T    cppDrillRegions;
} ESS_DURLINFO_T;

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_CHAR_T</td>
<td>bIsLevel0</td>
<td>If 1, then URL definition is restricted to level-0 data; if 0, there is no restriction</td>
</tr>
<tr>
<td>ESS_STR_T</td>
<td>cpURLName</td>
<td>Name of the drill-through URL</td>
</tr>
<tr>
<td>ESS_USHORT_T</td>
<td>iURLXmlSize</td>
<td>Size of the URL XML text</td>
</tr>
<tr>
<td>ESS_BYTE_T*</td>
<td>cpURLXml</td>
<td>Pointer to the URL XML text</td>
</tr>
<tr>
<td>ESS_USHORT_T</td>
<td>iCountOfDrillRegions</td>
<td>Number of regions referenced by the drill-through URL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The number of drillable regions in a drill-through URL is limited to 256.</td>
</tr>
<tr>
<td>ESS_PSTR_T</td>
<td>cppDrillRegions</td>
<td>List of regions referenced by the drill-through URL</td>
</tr>
</tbody>
</table>

**ESS_EXTUSERINFO_T**
Stores information about an externally authenticated user. The fields are:

typedef struct ESS_EXTUSERINFO_T
{
    ESS_USERNAME_T    Name;
    ESS_APPNAME_T     AppName;
    ESS_DBNAME_T      DbName;
    ESS_BOOL_T        Login;
    ESS_USHORT_T      Type;
    ESS_ACCESS_T      Access;
    ESS_ACCESS_T      MaxAccess;
    ESS_DATE_T        Expiration;
    ESS_TIME_T        LastLogin;
    ESS_TIME_T        DbConnectTime;
    ESS_USHORT_T      FailCount;
    ESS_LOGINID_T     LoginId;
    ESS_DESC_T        Description;
    ESS_EMAIL_T       EMailID;
    ESS_BOOL_T        LockedOut;
    ESS_BOOL_T        PwdChgNow;
    ESS_USHORT_T      authType;
    ESS_PROTOCOL_T    protocol;  
}
ESS_CONNPARAM_T connParam;
) ESS_EXTUSERINFO_T, *ESS_PEXTUSERINFO_T, **ESS_PPEXTUSERINFO_T;

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_USERNAME_T</td>
<td>Name</td>
<td>User name</td>
</tr>
<tr>
<td>ESS_APPNAME_T</td>
<td>AppName</td>
<td>Name of currently connected application (if applicable)</td>
</tr>
<tr>
<td>ESS_DBNAME_T</td>
<td>DbName</td>
<td>Name of the currently connected database (if applicable)</td>
</tr>
<tr>
<td>ESS_BOOL_T</td>
<td>Login</td>
<td>Flag to indicate login status.</td>
</tr>
</tbody>
</table>
| ESS_UHOST_T        |Typ        | Type of the structure. This field can contain the following values:
|                    |           | • ESS_TYPE_USER
|                    |           | • ESS_TYPE_GROUP |
| ESS_ACCESS_T       |Access     | User assigned default access privileges. Values: any combination of the following bit values:
|                    |           | • ESS_ACCESS_SUPER /* Supervisor, all bits set */
|                    |           | • ESS_PRIV_APPCREATE /* App create/delete privilege */
|                    |           | • ESS_PRIV_USERCREATE /* user create/delete privilege */ |
| ESS_ACCESS_T       |MaxAccess  | User's maximum access privileges (including individual access and access levels due to group membership). |
| ESS_DATE_T         |Expiration | User's password expiration date. |
| ESS_TIME_T         |LastLogin  | Date of user's last successful login stated as Greenwich Mean Time. |
| ESS_TIME_T         |DbConnectTime | Local (server) time of database connection. Read-only. Cannot be set by EssSetUser. |
| ESS_UHOST_T        |FailCount  | Count of the failed login attempts since the last successful login. |
| ESS_LOGINID_T      |LoginId    | The user login identification tag. |
| ESS_DESC_T         |Description| User description. |
| ESS_EMAIL_T        |EMailID    | User email address. |
| ESS_BOOL_T         |LockedOut  | Flag that user is locked out. |
| ESS_UHOST_T        |PwdChgNow  | Flag that user must change password. |
| ESS_UHOST_T        |authType   | Authentication type. |
| ESS_PROTOCOL_T     |protocol   | External authentication protocol: css for Shared Services mode. |
| ESS_CONNPARAM_T    |connParam  | External authentication connection parameters. Null if protocol is css. |

ESS_GENLEVELNAMEEX_T
Contains information about generation or level names and the member-name uniqueness settings for generation and levels. The fields are:
typedef struct ESS_GENLEVELNAMEEX_T
{
    ESS_USHORT_T, usNumber;
    ESS_BOOL_T, bNameUnique;
    ESS_MBRNAME_T, szName;
} ESS_GENLEVELNAMEEX_T, ESS_PGENLEVELNAMEEX_T, ESS_GENLEVELNAMEEX_T **,
ESS_PPGENLEVELNAMEEX_T;

Data Type | Field     | Description                      |
-----------|-----------|----------------------------------|
ESS_USHORT_T | usNumber  | Generation or level number       |
ESS_BOOL_T  | bNameUnique | Generation or level member-name uniqueness |
ESS_MBRNAME_T | szName   | Generation or level name         |

**ESS_GLOBAL_T**

Contains global server system parameters used for administrative purposes. All of the fields in this structure except *Currency* can be modified using the API. The fields are:

typedef struct ESS_GLOBAL_T
{
    ESS_BOOL_T   Security;
    ESS_BOOL_T   Logins;
    ESS_ACCESS_T Access;
    ESS_USHORT_T Validity;
    ESS_BOOL_T   Currency;
    ESS_USHORT_T PwMin;
    ESS_TIME_T   InactivityTime;
    ESS_TIME_T   InactivityCheck;
    ESS_USHORT_T InvalidAttempts;
    ESS_USHORT_T InactivityLockout;
    ESS_USHORT_T NumPwExpWarn;
    ESS_USHORT_T Pw StoredNum;
} ESS_GLOBAL_T, *ESS_PGLOBAL_T, **ESS_PPGLOBAL_T;

Data Type | Field     | Description                                      |
-----------|-----------|--------------------------------------------------|
ESS_BOOL_T | Security  | Flag to indicate whether global security is enabled (default is ESS_TRUE, indicating security is enabled) |
ESS_BOOL_T | Logins    | Flag to indicate whether user logins are enabled (default is ESS_TRUE, indicating logins are enabled). |
ESS_ACCESS_T | Access   | The default access level for newly-created applications (default is ESS_ACCESS_NONE). See "Bitmask Data Types (C)" on page 96 for a list of possible values. |
ESS_USHORT_T | Validity  | The default password validity period (default is 365 days). |
ESS_BOOL_T | Currency  | Flag to indicate whether currency option is supported (this flag is read only). Set to ESS_TRUE if the currency option is enabled. |
<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_USHORT_T</td>
<td>PwMin</td>
<td>The minimum permitted password length (default is 6 characters).</td>
</tr>
<tr>
<td>ESS_TIME_T</td>
<td>InactivityTime</td>
<td>Maximum length of time, in seconds, the user can be inactive before automatic logout from all applications and the Agent. Default value: 3600 seconds. Minimum value: 300 seconds. To disable auto logout, set InactivityTime to 0.</td>
</tr>
<tr>
<td>ESS_TIME_T</td>
<td>InactivityCheck</td>
<td>Frequency of checks for auto logout, in seconds. Default value: 300 seconds. Minimum value: 30 seconds. Must be smaller than InactivityTime setting or InactivityCheck is set to the value of InactivityTime and a warning message occurs. To disable auto logout, set InactivityCheck to 0.</td>
</tr>
<tr>
<td>ESS_USHORT_T</td>
<td>InvalidAttempts</td>
<td>The number of invalid attempts allowed by a user before the system administrator is warned and the user is locked out.</td>
</tr>
<tr>
<td>ESS_USHORT_T</td>
<td>InactivityLockout</td>
<td>The duration of a period of inactivity (between logins) for any user before that user is locked out (default is 365 days)</td>
</tr>
<tr>
<td>ESS_USHORT_T</td>
<td>NumPwExpWarn</td>
<td>The number of expired password warnings issued to a user before that user is locked out.</td>
</tr>
<tr>
<td>ESS_USHORT_T</td>
<td>PwStoredNum</td>
<td>The number of passwords stored for any user.</td>
</tr>
</tbody>
</table>

**ESS_INIT_T**

Passed to the API initialization function `EssInit()` and contains fields that let API developers customize their usage of the API. If any of the fields of the structure are set to zero (or NULL for pointers), the API defaults are used. (See “Using Memory in C Programs” on page 84 for more information.).

typedef struct ESS_INIT_T
{
    ESS_ULONG_T Version;
    ESS_PVOID_T UserContext;
    ESS_USHORT_T MaxHandles;
    ESS_SIZE_T MaxBuffer;
    ESS_STR_T LocalPath;
    ESS_STR_T MessageFile;
    ESS_PFUNC_T AllocFunc;
    ESS_PFUNC_T ReallocFunc;
    ESS_PFUNC_T FreeFunc;
    ESS_PFUNC_T MessageFunc;
    ESS_STR_T HelpFile;
    ESS_ULONG_T Ess_System;
    #ifdef AD_UTF8
    ESS_USHORT_T usApiType;
    #endif
    ESS_PCATCFUNC_T, CatchFunc;
    ESS_PCATC_INIT_FUNC_T, CatchInitFunc;
    ESS_PCATC_TERM_FUNC_T, CatchTermFunc;
    ESS_PCOOKIE_CREATE_FUNC_T, CookieCreate Func;
    ESS_PCOOKIE_DELETE_FUNC_T, CookieDeleteFunc;
} ESS_INIT_T, *ESS_PINIT_T;
<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_ULONG_T</td>
<td>Version</td>
<td>Version of Essbase API used to compile the application. Should be set to ESS_API_VERSION. Used for backward compatibility.</td>
</tr>
<tr>
<td>ESS_PVOID_T</td>
<td>UserContext</td>
<td>An optional pointer to a user-defined message context (passed as argument to a user-defined MessageFunction)</td>
</tr>
<tr>
<td>ESS_USHORT_T</td>
<td>MaxHandles</td>
<td>The maximum number of simultaneous context handles required by the API program (between 1 and 255). The default is 255. Reducing this number may decrease the amount of client memory used within the API for your program.</td>
</tr>
<tr>
<td>ESS_SIZE_T</td>
<td>MaxBuffer</td>
<td>The maximum size buffer that can be allocated in the client program (typically 64 KB). The default is 64 KB.</td>
</tr>
<tr>
<td>ESS_STR_T</td>
<td>LocalPath</td>
<td>The default local path name to use for file and object operations on the client. If this is not set, Essbase uses the ESSBASEPATH environment variable by default, and appends \CLIENT to the directory name passed in.</td>
</tr>
<tr>
<td>ESS_STR_T</td>
<td>MessageFile</td>
<td>Qualified path name of the message database file, ESSBASE.MDB. If this is not set, Essbase first tries to use the fully qualified path in the ARBORMSGPATH environment variable, otherwise, it uses (ESSBASEPATH) \BIN\ESSBASE.MDB. If ESSBASEPATH is not defined, an error is returned at run time.</td>
</tr>
<tr>
<td>ESS_PFUNC_T</td>
<td>AllocFunc</td>
<td>Pointer to the user-defined memory allocation function. All platforms: memory allocation functions use the malloc() function.</td>
</tr>
<tr>
<td>ESS_PFUNC_T</td>
<td>ReallocFunc</td>
<td>Pointer to the user-defined memory reallocation function. All platforms: memory allocation functions use the realloc() function.</td>
</tr>
<tr>
<td>ESS_PFUNC_T</td>
<td>FreeFunc</td>
<td>A pointer to the user-defined memory free function. All platforms: memory allocation functions use the free() function.</td>
</tr>
<tr>
<td>ESS_PFUNC_T</td>
<td>MessageFunc</td>
<td>A pointer to the user-defined message callback function. Messages sent to the user-defined Callback function are passed to Essbase in EssInit. Previous to Release 6.2, if a message contained NLS characters (foreign language characters, such as accented characters), Essbase provided them in OEM (DOS) format. In Release 6.2 and later, these messages are completely in character (Windows) format, to avoid the misinterpretation of certain characters. This only affects localized versions of Essbase.</td>
</tr>
<tr>
<td>ESS_STR_T</td>
<td>HelpFile</td>
<td>Fully-qualified path name of the user-defined application help file, used for help for the AutoLogin dialog box. The login help context must be defined in the help file.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>See Chapter 3, “Integrating Essbase With Your Product.”</td>
</tr>
<tr>
<td></td>
<td></td>
<td>By default, clicking the Help button displays the Essbase System Login help topic shipped with the Oracle Essbase Spreadsheet Add-in User’s Guide online help.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If ESSBASEPATH is not defined, the help file name is set to null.</td>
</tr>
<tr>
<td>ESS_ULONG_T</td>
<td>Ess_System</td>
<td>Reserved for internal use. Set to NULL</td>
</tr>
<tr>
<td>ESS_USHORT_T</td>
<td>usApiType</td>
<td>Required. Defines whether the program is in Unicode or non-Unicode mode. For valid values, see “Unicode Mode Constants (C)” on page 107.</td>
</tr>
<tr>
<td>typedef ESS_ULONG_T</td>
<td>CatchFunc</td>
<td>If implemented by the client, Essbase calls this function intermittently (every few seconds) during queries. If the routine returns TRUE, the API call gets canceled.</td>
</tr>
</tbody>
</table>
typedef ESS_STS_T (*ESS_PCATCH_INIT_FUNC_T)(ESS_HCTX_T);

**CatchInitFunc**
This function initializes resources for whatever state is needed for the CatchFunc call. For example, if you want to terminate a query based on whether a user hits the ESC key, and CatchFunc calls on a routine to get data from the keyboard, you may need to pre-initialize memory so that it is not initialized for every CatchFunc call.

Essbase executes the following process during a query:
1. Calls CatchInitFunc, if it is non NULL.
2. Executes query, intermittently calling CatchFunc.
3. Calls CatchTermFunc, if it is non NULL.

typedef ESS_STS_T (*ESS_PCATCH_TERM_FUNC_T)(ESS_HCTX_T);

**CatchTermFunc**
This function terminates resources initialized by CatchInitFunc.

typedef ESS_STS_T (*ESS_PCOOKIE_CREATE_FUNC_T)(ESS_HCTX_T);

**CookieCreateFunc**
Essbase calls this function at SetActive time. You would use this function if user information is required for the CatchFunc, CatchInitFunc, or CatchTermFunc calls. For example, if you want to terminate a query based on certain user activities, you may need to create a cookie to be used by the CatchFunc call. You obtain the cookie by calling EssGetCookie.

typedef ESS_STS_T (*ESS_PCOOKIE_DELETE_FUNC_T)(ESS_HCTX_T);

**CookieDeleteFunc**
This function deletes the cookie created by CookieCreateFunc. Essbase calls this function at ClearActive time.

---

**Query Cancellation Using Essbase API**

Programs developed using the Essbase API can optionally register custom query-cancellation functions at initialization. ESS_INIT_T has five fields that enable development of custom callback functions for query cancellation. These fields are CatchFunc, CatchInitFunc, CatchTermFunc, CookieCreateFunc, and CookieDeleteFunc. By default, they are set to null.

**Query Cancellation Usage Example**

The following code enables query cancellation when the Escape key is hit. KbdHitEx gets the next key that was entered from the keyboard, and writes the value of the key to kbfinfo.chChar.

```c
ESS_INIT_STRUCT InitStruct;
   InitStruct.CatchFunc = KillReqCatcher;

ESS_BOOL_T KillReqCatcher(ESS_HCTX_T hCtx)
{
   KBDINFO_T kbfinfo;
   if (KbdHitEx(&kbfinfo) & kbfinfo.chChar == KB_ESC)
      return ESS_TRUE;
   else
      return ESS_FALSE;
}
```
However, suppose the routine KdbHitEx requires that an initialization routine InitializeMyKeyboard be called first, and a terminate TerminateMyKeyboard routine be called later. Here you would use CatchInitFunc and CatchTermFunc.

```c
InitStruct.CatchInitFunc = InitKeyboard;
InitStruct.CatchTermFunc = TerminateKeyboard;
```

```c
ESS_STS_T InitKeyboard (ESS_HCTX_T hCtx)
{
    return InitializeMyKeyboard ();
}
ESS_STS_T TerminateKeyboard (ESS_HCTX_T hCtx)
{
    return TerminateMyKeyboard ();
}
```

Now suppose that the InitializeMyKeyboard and TerminateMyKeyboard routines need to retain status information. You can use a cookie to retain the status. The cookie created by CookieCreateFunc can be accessed in CatchFunc, CatchInitFunc, and CatchTermFunc by EssGetCookie.

```c
InitStruct.CatchInitFunc = InitKeyboard2;
InitStruct.CatchTermFunc = TerminateKeyboard2;
InitStruct.CookieCreateFunc = AllocKeyboardState;
InitStruct.CookieDeleteFunc = FreeKeyboardState;
```

```c
ESS_STS_T InitKeyboard2 (ESS_HCTX_T hCtx)
{
    ESS_PVOID_T cookie;
    ESS_STS_T sts;
    sts = EssGetCookie(hCtx, &cookie);
    if (sts)
        return sts;
    return InitializeMyKeyboard (cookie);
}

ESS_STS_T TerminateKeyboard2 (ESS_HCTX_T hCtx)
{
    ESS_PVOID_T cookie;
    ESS_STS_T sts;
    sts = EssGetCookie(hCtx, &cookie);
    if (sts)
        return sts;
    return TerminateMyKeyboard (cookie);
}

ESS_STS_T AllocKeyboardState(ESS_PVOID_T pKbdState)
{
    *pKbdState = malloc(KBDSTRUCT_SIZE);
    if (*pKbdState)
        return 0;
    else
        return -1;
}

ESS_STS_T FreeKeyboardState (ESS_PVOID_T kbdState)
{
    if (kbdState)
```
free(kbdState);
    return 0;
}

**ESS_LOAD_BUFFER_T**

Contains information about aggregate storage data load buffers. It is used by `EssListExistingLoadBuffers`.

typedef struct ESS_LOAD_BUFFER_T
{
    ESS_ULONG_T      ulBufferId;
    ESS_ULONG_T      ulDuplicateAggregationMethod;
    ESS_ULONG_T      ulOptionFlags;
    ESS_ULONG_T      ulSize;
    ESS_BOOL_T       bInternal;
    ESS_BOOL_T       bActive;
    ESS_BOOL_T       bReserved01;
    ESS_BOOL_T       bReserved02;
    ESS_ULONG_T      ulReserved01;
    ESS_ULONG_T      ulReserved02;
    ESS_ULONG_T      ulReserved03;
} ESS_LOAD_BUFFER_T;

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_ULONG_T</td>
<td>ulBufferId</td>
<td>ID of a data load buffer (a number between 1 and 4294967296).</td>
</tr>
<tr>
<td>ESS_ULONG_T</td>
<td>ulDuplicateAggregationMethod</td>
<td>One of the following constants describing how to combine multiple values for the same cell within the buffer:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>* ESS_ASO_DATA_LOAD_BUFFER_DUPLICATES_ADD: Add values when the buffer contains multiple values for the same cell.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>* ESS_ASO_DATA_LOAD_BUFFER_DUPLICATES_ASSUME_EQUAL: Verify that multiple values for the same cells are identical, and ignore the duplicates if they are. Stop the data load with an error message if they differ.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>* ESS_ASO_DATA_LOAD_BUFFER_DUPLICATES_USE_LAST: Use the last value loaded into the buffer as the final value for the cell.</td>
</tr>
<tr>
<td>ESS_ULONG_T</td>
<td>ulOptionFlags</td>
<td>Either (or a combination) of the following load buffer options:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>* ESS_ASO_DATA_LOAD_BUFFER_IGNORE_MISSING_VALUES</td>
</tr>
<tr>
<td></td>
<td></td>
<td>* ESS_ASO_DATA_LOAD_BUFFER_IGNORE_ZERO_VALUES</td>
</tr>
<tr>
<td>ESS_ULONG_T</td>
<td>ulSize</td>
<td>The percentage of the aggregate storage cache that the data load buffer is allowed to use (a number between 1 and 100, inclusive)</td>
</tr>
<tr>
<td>ESS_BOOL_T</td>
<td>bInternal</td>
<td>ESS_TRUE if the buffer was created by Essbase; ESS_FALSE for user-created buffers</td>
</tr>
<tr>
<td>ESS_BOOL_T</td>
<td>bActive</td>
<td>ESS_TRUE if the buffer is currently in use by a data load</td>
</tr>
<tr>
<td>ESS_BOOL_T</td>
<td>bReserved01</td>
<td>Not used</td>
</tr>
<tr>
<td>ESS_BOOL_T</td>
<td>bReserved02</td>
<td>Not used</td>
</tr>
<tr>
<td>Data Type</td>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>---------------</td>
<td>----------------</td>
</tr>
<tr>
<td>ESS_ULONG_T</td>
<td>ulReserved01</td>
<td>Not used</td>
</tr>
<tr>
<td>ESS_ULONG_T</td>
<td>ulReserved02</td>
<td>Not used</td>
</tr>
<tr>
<td>ESS_ULONG_T</td>
<td>ulReserved03</td>
<td>Not used</td>
</tr>
</tbody>
</table>

### ESS_LOCKINFO_T

Contains information about data blocks exclusively locked, as returned by the `istLocks()` function. Fields in this structure cannot be modified using the API.

```c
typedef struct ESS_LOCKINFO_T {
    ESS_USERNAME_T UserName;
    ESS_USHORT_T nLocks;
    ESS_TIME_T Time;
    ESS_LOGINID_T LoginId;
} ESS_LOCKINFO_T, *ESS_PLOCKINFO_T, **ESS_PPLOCKINFO_T;
```

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_USERNAME_T</td>
<td>UserName</td>
<td>The user name</td>
</tr>
<tr>
<td>ESS_USHORT_T</td>
<td>nLocks</td>
<td>The number of blocks exclusively locked by this user</td>
</tr>
<tr>
<td>ESS_TIME_T</td>
<td>Time</td>
<td>The maximum time (in seconds) that blocks have been exclusively locked</td>
</tr>
<tr>
<td>ESS_LOGINID_T</td>
<td>LoginId</td>
<td>The user login identification tag</td>
</tr>
</tbody>
</table>

### ESS_LOCKINFOEX_T

Contains information about data blocks exclusively locked, as returned by the `ListLocks()` function. This structure is similar to `ESS_LOCKINFO_T`, with the addition of the `ProviderName` and `connparam` fields. Fields in this structure cannot be modified using the API.

```c
typedef struct ESS_LOCKINFOEX_T {
    ESS_USERNAME_T UserName;
    ESS_USERNAME_T ProviderName;
    ESS_CONNPARAM_T connparam
    ESS_USHORT_T nLocks;
    ESS_TIME_T Time;
    ESS_LOGINID_T LoginId;
} ESS_LOCKINFOEX_T, *ESS_PLOCKINFOEX_T, **ESS_PPLOCKINFOEX_T;
```
Data Type | Field | Description
--- | --- | ---
ESS_USERNAME_T | UserName | User name
ESS_USERNAME_T | ProviderName | Name of the user directory. Example: @Native Directory
ESS_CONNPARAM_T | connparam | Unique identity attribute identifying user or group in a directory. Example: native://nvid=f0ed2a6d7fb07688:5a342200:1265973105c:-7f46?USER
ESS_USHORT_T | nLocks | Number of blocks exclusively locked by the user
ESS_TIME_T | Time | Maximum time (in seconds) that blocks have been exclusively locked
ESS_LOGINID_T | LoginId | User login identification tag

**ESS_LOG_DATALOAD_T**

Contains metadata describing dataloads.

typedef struct ESS_LOG_DATALOAD_T {
    ESS_OBJTYPE_T, datfile_type;
    ESS_UCHAR_T, datfile_loc;
    ESS_FILENAME_T, dat_filename;
    ESS_UCHAR_T, isRuleFile;
    ESS_UCHAR_T, rulfile_loc;
    ESS_FILENAME_T, rul_filename;
    ESS_USERNAME_T, sql_username;
    ESS_PASSWORD_T, sql_password;
    ESS_UCHAR_T, isAbortOnErr;
    ESS_ULONG_T, reserved0;
    ESS_ULONG_T, reserved1;
    ESS_ULONG_T, reserved2;
} ESS_LOG_DATALOAD_T;

Data Type | Field | Description
--- | --- | ---
ESS_OBJTYPE_T | datfile_type | Data file type
ESS_UCHAR_T | datfile_loc | Data file location/SQL
ESS_FILENAME_T | dat_filename | Data file name
ESS_UCHAR_T | isRuleFile | Is there a rule file
ESS_UCHAR_T | rulfile_loc | Rule file location
ESS_FILENAME_T | rul_filename | Rule file name
ESS_USERNAME_T | sql_username | SQL connection username
ESS_PASSWORD_T | sql_password | SQL connection password
ESS_UCHAR_T | isAbortOnErr | Is there an error file name required
### ESS_MBRALT_T

Contains information about a specified member alias table. Fields in this structure cannot be modified using the API. The fields are:

```c
typedef struct ESS_MBRALT_T {
    ESS_MBRNAME_T MbrName;
    ESS_MBRNAME_T AltName;
} ESS_MBRALT_T, *ESS_PMBRALT_T, **ESS_PPMBRALT_T;
```

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_MBRNAME_T</td>
<td>MbrName</td>
<td>The member name</td>
</tr>
<tr>
<td>ESS_MBRNAME_T</td>
<td>AltName</td>
<td>The associated alias name</td>
</tr>
</tbody>
</table>

### ESS_MBRERR_T

Used for a linked list of member errors. It is used by EssImport().

```c
typedef struct ESS_MBRERR_T {
    struct ess_mbrerr_t *pNext;
    ESS_USHORT_T ErrType;
    ESS_STR_T Name;
    ESS_STR_T Record;
} ESS_MBRERR_T, *ESS_PMBRERR_T, **ESS_PPMBRERR_T;
```

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>struct ESS_MBRERR_T</td>
<td>*pNext</td>
<td>Pointer to next structure in list</td>
</tr>
<tr>
<td>ESS_USHORT_T</td>
<td>ErrType</td>
<td>The type of error</td>
</tr>
<tr>
<td>ESS_STR_T</td>
<td>Name</td>
<td>The member name</td>
</tr>
<tr>
<td>ESS_STR_T</td>
<td>Record</td>
<td>The file record containing the error</td>
</tr>
</tbody>
</table>

### ESS_MBRUSER_T

An external data source user information structure. Fields in this structure cannot be modified by the API. The fields are:

```c
typedef struct ESS_MBRUSER_T {
    ESS_STR_T User;
```
ESS_STR_T Password;
} ESS_MBRUSER_T, *ESS_PMBRUSER_T;

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_STR_T</td>
<td>User</td>
<td>The external data source user name</td>
</tr>
<tr>
<td>ESS_STR_T</td>
<td>Password</td>
<td>The external data source password</td>
</tr>
</tbody>
</table>

**ESS_MEMBERINFO_T**

Contains information about a specified database member. Fields in this structure cannot be modified using the API. The fields are:

typedef struct ESS_MEMBERINFO_T
{
    ESS_MBRNAME_T      MbrName;
    ESS_MEMNUM_T       MbrNumber;
    ESS_MBRNAME_T      DimName;
    ESS_DIMNUM_T       DimNumber;
    ESS_USHORT_T       Status;
    ESS_SHORT_T        Level;
    ESS_SHORT_T        Generation;
    ESS_SHORT_T        UnaryCalc;
    ESS_USHORT_T       MbrTagType;
    ESS_BOOL_T         CurrConvert;
    ESS_MBRNAME_T      CrmbrName;
    ESS_DESC_T         Description;
    ESS_MBRNAME_T      ParentMbrName;
    ESS_MBRNAME_T      ChildMbrName;
    ESS_MBRNAME_T      PrevMbrName;
    ESS_MBRNAME_T      NextMbrName;
    ESS_BOOL_T         fAttributed;
    ESS_ATTRIBUTEVALUE_T Attribute;
    ESS_BOOL_T         fHasRelDesc;
    ESS_BOOL_T         fHasHAEEnabled;
} ESS_MEMBERINFO_T, *ESS_PMEMBERINFO_T, **ESS_PPMEMBERINFO_T;

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_MBRNAME_T</td>
<td>MbrName</td>
<td>The member name</td>
</tr>
<tr>
<td>ESS_MEMNUM_T</td>
<td>MbrNumber</td>
<td>The member number in the database outline</td>
</tr>
<tr>
<td>ESS_MBRNAME_T</td>
<td>DimName</td>
<td>The member's dimension name</td>
</tr>
<tr>
<td>ESS_DIMNUM_T</td>
<td>DimNumber</td>
<td>The member's dimension number</td>
</tr>
<tr>
<td>Data Type</td>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ESS_USHORT_T</td>
<td>Status</td>
<td>The member’s share status is derived by performing a logical AND between the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>contents of this field and each of the constant values of the form</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESS_MBRSTS_xxx:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESS_MBRSTS_NOTSET</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESS_MBRSTS_NEVER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESS_MBRSTS_LABEL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESS_MBRSTSREFER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESS_MBRSTS_REFNME</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESS_MBRSTS_SHARE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESS_MBRSTS_VIRTSTORE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESS_MBRSTS_VIRTNOSTORE</td>
</tr>
<tr>
<td>ESS_SHORT_T</td>
<td>Level</td>
<td>The member level number (zero-based), counting up from the lowest descendent</td>
</tr>
<tr>
<td></td>
<td></td>
<td>of the specified member</td>
</tr>
<tr>
<td>ESS_SHORT_T</td>
<td>Generation</td>
<td>The member generation number (one-based), counting down from the specified</td>
</tr>
<tr>
<td></td>
<td></td>
<td>member’s dimension member</td>
</tr>
<tr>
<td>ESS_SHORT_T</td>
<td>UnaryCalc</td>
<td>The default unary rollup for this member. A value of the form ESS_UCALC_xxx</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(add, subtract, multiply, divide, percent, none, or never).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ESS_UCALC_ADD</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ESS_UCALC_SUB</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ESS_UCALC_MULT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ESS_UCALC_DIV</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ESS_UCALC_PERCENT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ESS_UCALC_NOOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ESS_UCALC_NEVER</td>
</tr>
<tr>
<td>ESS_USHORT_T</td>
<td>MbrTagType</td>
<td>A 16 bit mask for the member’s tagged types. A value of the form ESS_ATYPE_</td>
</tr>
<tr>
<td></td>
<td></td>
<td>xxx.</td>
</tr>
<tr>
<td>ESS_BOOL_T</td>
<td>CurrConvert</td>
<td>Currency Conversion. Values: ESS_TRUE and ESS_FALSE</td>
</tr>
<tr>
<td>ESS_MBRNAME_T</td>
<td>CrMbrName</td>
<td>Name of the tagged currency database member.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• For Time dimension, gives the name of the tagged time member.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• For Country dimension, gives the name of the tagged currency member.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• For Accounts dimension, gives the name of the tagged category member.</td>
</tr>
<tr>
<td>ESS_DESC_T</td>
<td>Description</td>
<td>Member description</td>
</tr>
<tr>
<td>ESS_MBRNAME_T</td>
<td>ParentMbrName</td>
<td>Specified member’s parent member name or empty string if member has no parent</td>
</tr>
<tr>
<td>ESS_MBRNAME_T</td>
<td>ChildMbrName</td>
<td>Specified member’s first child member name</td>
</tr>
<tr>
<td>ESS_MBRNAME_T</td>
<td>PrevMbrName</td>
<td>Specified member’s previous sibling member name</td>
</tr>
<tr>
<td>ESS_MBRNAME_T</td>
<td>NextMbrName</td>
<td>Specified member’s next sibling member name</td>
</tr>
<tr>
<td>ESS_BOOL_T</td>
<td>fAttributed</td>
<td>Indicates whether the member has attributes associated with it. Values: ESS_TRUE and ESS_FALSE.</td>
</tr>
</tbody>
</table>
### ESS_ATTRIBUTEVALUE_T

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;ESS_ATTRIBUTEVALUE_T&quot; on page 118</td>
<td>Attribute</td>
<td>Attribute value</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_BOOL_T</td>
<td>fHasRelDesc</td>
<td>The member has one or more relational children.</td>
</tr>
<tr>
<td>ESS_BOOL_T</td>
<td>fHasHAEnabled</td>
<td>The dimension has Hybrid Analysis relational partitions enabled. Valid only for Dimension members.</td>
</tr>
</tbody>
</table>

### ESS_NEWSHAREDSERVICESNATIVEUSERINFO_T

Contains the user names and corresponding passwords resulting from a migration of users and groups to Shared Services using the option of automatically generated password assignment.

Users were created as new Shared Services users during migration because there were no matching names found. The fields are:

```c
typedef struct ESS_NEWSHAREDSERVICESNATIVEUSERINFO_T
{
    ESS_USERNAME_T,   Name;
    ESS_PASSWORD_T,    Password;
} ESS_NEWHUBNATIVEUSERINFO_T;
```

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_USERNAME_T</td>
<td>Name</td>
<td>User or group name.</td>
</tr>
<tr>
<td>ESS_PASSWORD_T</td>
<td>Password</td>
<td>User password.</td>
</tr>
</tbody>
</table>

### ESS_OBJDEF_T

Provides summary object information. It is used by `EssImport` and `EssBuildDimension`. The fields in this structure cannot be modified by the API.

```c
typedef struct ESS_OBJDEF_T
{
    ESS_HCTX_T    hCtx;
    ESS_OBJTYPE_T ObjType;
    ESS_STR_T    AppName;
    ESS_STR_T    DbName;
    ESS_STR_T     FileName;
} ESS_OBJDEF_T, *ESS_POBJDEF_T;
```

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_HCTX_T</td>
<td>hCtx</td>
<td>Object context handle</td>
</tr>
<tr>
<td>ESS_OBJTYPE_T</td>
<td>ObjType</td>
<td>Object type. See &quot;Bitmask Data Types (C)&quot; on page 96 for a list of object types.</td>
</tr>
<tr>
<td>ESS_STR_T</td>
<td>AppName</td>
<td>Application name</td>
</tr>
</tbody>
</table>
### ESS_OBJINFO_T

Contains information about a specific file object. You cannot modify fields in this structure through the API. The fields are:

```c
typedef struct ESS_OBJINFO_T {
    ESS_OBJNAME_T    Name;
    ESS_OBJTYPE_T    Type;
    ESS_APPNAME_T   AppName;
    ESS_DBNAME_T     DbName;
    ESS_ULONG_T      FileSize;
    ESS_BOOL_T       Locked;
    ESS_USERNAME_T   User;
    ESS_TIME_T       TimeStamp;
    ESS_TIME RECORD_T TimeModified;
} ESS_OBJINFO_T, *ESS_POBJINFO_T, **ESS_PPOBJINFO_T;
```

### ESS_PART_T

Main shared partition data structure.
typedef struct ESS_PART_T
{
    ESS_PARTHDR_T    file_header;
    ESS_USHORT_T     part_count;
    ESS_PARTDEF_T    *parts;
    ESS_ULONG_T      maxserialno;
} ESS_PART_T, *ESS_PPART_T, **ESS_PPPART_T;

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;ESS_PARTHDR_T&quot; on page 159</td>
<td>file_header</td>
<td>File header.</td>
</tr>
<tr>
<td>ESS_USHORT_T</td>
<td>part_count</td>
<td>Number of shared partitions.</td>
</tr>
<tr>
<td>&quot;ESS_PARTDEF_T&quot; on page 157</td>
<td>partitions</td>
<td>Array of shared partition definitions.</td>
</tr>
<tr>
<td>ESS_ULONG_T</td>
<td>maxserialno</td>
<td>High water mark for serial number.</td>
</tr>
</tbody>
</table>

**ESS_PART_CONNECT_INFO_T**

Specifies a database.

typedef struct ESS_PART_CONNECT_INFO_T
{
    ESS_STR_T   pszHostName;
    ESS_STR_T   pszAppName;
    ESS_STR_T   pszDbName;
} ESS_PART_CONNECT_INFO_T, *ESS_PPART_CONNECT_INFO_T, **ESS_PPPART_CONNECT_INFO_T;

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_STR_T</td>
<td>pszHostName</td>
<td>Host name.</td>
</tr>
<tr>
<td>ESS_STR_T</td>
<td>pszAppName</td>
<td>Application name.</td>
</tr>
<tr>
<td>ESS_STR_T</td>
<td>pszDbName</td>
<td>Database name.</td>
</tr>
</tbody>
</table>

**ESS_PART_DEFINED_T**

Specifies a shared partition.

typedef struct ESS_PART_DEFINED_T
{
    ESS_USHORT_T                 usType;
    ESS_USHORT_T                 usDirection;
    ESS_PART_CONNECT_INFO_T      HostDatabase;
} ESS_PART_DEFINED_T, *ESS_PPART_DEFINED_T, **ESS_PPPART_DEFINED_T;

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_USHORT_T</td>
<td>usType</td>
<td>One of the Operation Type constants listed below.</td>
</tr>
<tr>
<td>ESS_USHORT_T</td>
<td>usDirection</td>
<td>One of the Directions constants listed below.</td>
</tr>
</tbody>
</table>
### Operation Type Constants

```c
#define ESS_PARTITION_OP_REPLICATED 0x0001
#define ESS_PARTITION_OP_LINKED 0x0002
#define ESS_PARTITION_OP_TRANSPARENT 0x0004
#define ESS_PARTITION_OP_ALLTYPES (ESS_PARTITION_OP_REPLICATED | ESS_PARTITION_OP_LINKED | ESS_PARTITION_OP_TRANSPARENT)
```

### Direction Constants

```c
#define ESS_PARTITION_DATA_SOURCE 0x0001
#define ESS_PARTITION_DATA_TARGET 0x0002
#define ESS_PARTITION_DATA_BOTH (ESS_PARTITION_DATA_SOURCE | ESS_PARTITION_DATA_TARGET)
```

### ESS_PART_INFO_T

Holds the multicube shared partition information.

```c
typedef struct ESS_PART_INFO_T
{
    ESS_USHORT_T OperationType;
    ESS_USHORT_T DataDirection;
    ESS_USHORT_T MetaDirection;
    ESS_SVRNAME_T SvrName;
    ESS_APPNAME_T AppName;
    ESS_DBNAME_T DbName;
    ESS_TIME_T LastMetaUpdateTime;
    ESS_TIME_T LastRefreshTime;
    ESS_BOOL_T AreaUpdatable;
    ESS_BOOL_T IncrRefreshAllowed;
    ESS_TIME_T LastUpdateTime;
} ESS_PART_INFO_T, *ESS_PPART_INFO_T, **ESS_PPPART_INFO_T;
```

### Data Type | Field | Description
---|---|---
ESS_USHORT_T | OperationType | Operation type supported by this partition.
ESS_USHORT_T | DataDirection | Remote connection information (is this the source or target side?).
ESS_SVRNAME_T | SvrName | Host for the other side of the partition definition.
ESS_APPNAME_T | AppName | Application for the other side of the partition definition.
ESS_DBNAME_T | DbName | Database for other side of the partition definition; meta data change information.
ESS_TIME_T | LastMetaUpdateTime | Last time meta data was updated.
<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>The following fields only apply to replication data targets</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ESS_TIME_T</td>
<td>LastRefreshTime</td>
<td>Last time data at target was refreshed.</td>
</tr>
<tr>
<td>ESS_BOOL_T</td>
<td>partitionUpdatable</td>
<td>Are changes allowed to replicated data?</td>
</tr>
</tbody>
</table>

| **The following fields only apply to replication data sources** |                         |                                                       |
| ESS_BOOL_T    | IncrRefreshAllowed     | Can we refresh only the changed data?                 |
| ESS_TIME_T    | LastUpdateTime         | Time of last change to data in the partition.         |

**Operation Type Constants**

```
#define ESS_PARTITION_OP_REPLICATED 0x0001
#define ESS_PARTITION_OP_LINKED 0x0002
#define ESS_PARTITION_OP_TRANSPARENT 0x0004
#define ESS_PARTITION_OP_ALLTYPES (ESS_PARTITION_OP_REPLICATED | 
                                 ESS_PARTITION_OP_LINKED | 
                                 ESS_PARTITION_OP_TRANSPARENT)
```

**Direction Constants**

```
#define ESS_PARTITION_DATA_SOURCE 0x0001
#define ESS_PARTITION_DATA_TARGET 0x0002
#define ESS_PARTITION_DATA_BOTH (ESS_PARTITION_DATA_SOURCE | 
                              ESS_PARTITION_DATA_TARGET)
```

**ESS_PART_REPL_T**

Queries shared partitions.

```c
typedef struct ESS_PART_REPL_T {
    ESS_LONG_T         lAreaCount;
    ESS_BOOL_T         bUpdatedOnly;
    ESS_PPART_CONNECT_INFO_T pHostDatabase;
} ESS_PART_REPL_T, *ESS_PPART_REPL_T, **ESS_PPPART_REPL_T;
```

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_LONG_T</td>
<td>lPartitionCount</td>
<td>Number of partitions to refresh from (-1 == ALL)</td>
</tr>
<tr>
<td>ESS_BOOL_T</td>
<td>bUpdatedOnly</td>
<td>Refreshes only the cells modified at the source since the last refresh operation.</td>
</tr>
</tbody>
</table>

**ESS_PARTDEF_INVALID_T**

This is the shared partition verification structure.

"ESS_PART_CONNECT_INFO_T" on page 153
typedef struct ESS_PARTDEF_INVALID_T
{
    ESS_USHORT_T error_type;
    ESS_ULONG_T line_number;
    ESS_ULONG_T overlap_number;
    ESS_CHAR_T member_name[ESS_MBRNAMELEN];
    ESS_CHAR_T error_message[ESS_LINELEN];
} ESS_PARTDEF_INVALID_T, *ESS_PPARTDEF_INVALID_T, **ESS_PPPARTDEF_INVALID_T;

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_USHORT_T</td>
<td>error_type</td>
<td>One of the Error constants listed below.</td>
</tr>
<tr>
<td>ESS_ULONG_T</td>
<td>line_number</td>
<td>Line number for the erroneous line.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For partition defn: line number</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For global map: line number</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For slice map: slice number.</td>
</tr>
<tr>
<td>ESS_ULONG_T</td>
<td>overlap_number</td>
<td>Slice number for overlapped slices, partition number for overlapped partition.</td>
</tr>
<tr>
<td>ESS_CHAR_T</td>
<td>member_name[ESS_MBRNAMELEN]</td>
<td>Erroneous member name, used only for mapping rules.</td>
</tr>
<tr>
<td>ESS_CHAR_T</td>
<td>error_message[ESS_LINELEN]</td>
<td>One of the Error constants listed below.</td>
</tr>
</tbody>
</table>

Error Constants

define ESS_PARTITION_DEF_ERROR           = 1
define ESS_PARTITION_GLOBAL_MAP_ERROR    = 2
define ESS_PARTITION_AREA_MAP_ERROR      = 3
define ESS_PARTITION_AREA_OVERLAP_ERROR  = 4
define ESS_PARTITION_OVERLAP_ERROR       = 5
define ESS_PARTITION_CELLCOUNT_MISMATCH  = 6
define ESS_PARTITION_TYPE_CONFLICT       = 8
define ESS_PARTITION_DEFAULT_LOGIN_ERROR = 9
define ESS_PARTITION_INVALID_USER        = 10
define ESS_PARTITION_INVALID_PW          = 11

ESS_PARTDEF_CONNECT_T

Holds connection information.

typedef struct ESS_PARTDEF_CONNECT_T
{
    ESS_CHAR_T svrname[ESS_SVRNAMELEN];
    ESS_CHAR_T appname[ESS_APPNAMELEN];
    ESS_CHAR_T dbname[ESS_DBNAMELEN];
    ESS_CHAR_T username[ESS_USERNAMELEN];
    ESS_CHAR_T password[ESS_PASSWORDLEN];
} ESS_PARTDEF_CONNECT_T, *ESS_PPARTDEF_CONNECT_T, **ESS_PPPARTDEF_CONNECT_T;

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_CHAR_T</td>
<td>svrname [ESS_SVRNAMELEN]</td>
<td>Server name.</td>
</tr>
<tr>
<td>Data Type</td>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>---------------------------</td>
<td>------------------------------------</td>
</tr>
<tr>
<td>ESS_CHAR_T</td>
<td>appname [ESS_APPNAMELEN]</td>
<td>Application name.</td>
</tr>
<tr>
<td>ESS_CHAR_T</td>
<td>dbname [ESS_DBNAMELEN]</td>
<td>Database name.</td>
</tr>
<tr>
<td>ESS_CHAR_T</td>
<td>username [ESS_USERNAMELEN]</td>
<td>Administrator username.</td>
</tr>
<tr>
<td>ESS_CHAR_T</td>
<td>password [ESS_PASSWORDLEN]</td>
<td>Administrator password.</td>
</tr>
</tbody>
</table>

**ESS_PARTDEF_MAP_T**

Holds mapping information.

typedef struct ESS_PARTDEF_MAP_T {
    ESS_ULONG_T mbr_count;
    ESS_STR_T *src_mbrs;
    ESS_STR_T *dest_mbrs;
} ESS_PARTDEF_MAP_T, *ESS_PPARTDEF_MAP_T, **ESS_PPPARTDEF_MAP_T;

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_ULONG_T</td>
<td>mbr_count</td>
<td>Size of remapping arrays.</td>
</tr>
<tr>
<td>ESS_STR_T</td>
<td>src_mbrs</td>
<td>Array of member names at src.</td>
</tr>
<tr>
<td>ESS_STR_T</td>
<td>dest_mbrs</td>
<td>Array of member names at target.</td>
</tr>
</tbody>
</table>

**ESS_PARTDEF_T**

Contains the partition definition.

typedef struct ESS_PARTDEF_T {
    ESS_PARTDEF_CONNECT_T connection;
    ESS_STR_T description;
    ESS_PARTDEF_AREAS_T shape_defn;
    ESS_PARTDEF_TYPE_T typedata;
    ESS_ULONG_T serialno;
    ESS_TIME_T meta_last_updated;
} ESS_PARTDEF_T, *ESS_PPARTDEF_T, **ESS_PPPARTDEF_T;

<table>
<thead>
<tr>
<th>&quot;ESS_PARTDEF_CONNECT_T&quot; on page 156</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>connection</td>
<td></td>
<td>Connection information.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ESS_STR_T</th>
<th>description</th>
<th>User's description of partition.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>&quot;ESS_PARTDEF_AREAS_T&quot; on page 158</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>shape_defn</td>
<td></td>
<td>Shape definition.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>&quot;ESS_PARTDEF_TYPE_T&quot; on page 158</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>typedata</td>
<td></td>
<td>Type-specific data.</td>
</tr>
</tbody>
</table>
### ESS_PARTDEF_AREAS_T

Holds shape definitions. A shape is composed of multiple slices.

```c
typedef struct ESS_PARTDEF_AREAS_T
{
    ESS_USHORT_T slice_count;
    ESS_STR_T *slices;
} ESS_PARTDEF_AREAS_T, *ESS_PPARTDEF_AREAS_T, **ESS_PPPARTDEF_AREAS_T;
```

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_USHORT_T</td>
<td>slice_count</td>
<td>Number of slices.</td>
</tr>
<tr>
<td>ESS_STR_T</td>
<td>slices</td>
<td>Array of slice definition strings.</td>
</tr>
</tbody>
</table>

### ESS_PARTDEF_TYPE_T

Holds partition type-specific information.

```c
typedef struct ESS_PARTDEF_TYPE_T
{
    ESS_USHORT_T operation_type;
    ESS_USHORT_T direction_type;
    ESS_USHORT_T meta_direction_type;
    ESS_PARTDEF_MAP_T area_map;
    ESS_PARTDEF_MAP_T *slice_maps;
    ESS_TIME_T last_refreshed;
    ESS_BOOL_T incr_refresh;
    ESS_BOOL_T updatable;
    ESS_CHAR_T defaultuser[ESS_USERNAMELEN];
    ESS_CHAR_T defaultpass[ESS_PASSWORDLEN];
} ESS_PARTDEF_TYPE_T, *ESS_PPARTDEF_TYPE_T, **ESS_PPPARTDEF_TYPE_T;
```

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_USHORT_T</td>
<td>operation_type</td>
<td>One of the Operation Type constants listed below.</td>
</tr>
<tr>
<td>ESS_USHORT_T</td>
<td>direction_type</td>
<td>One of the Direction constants listed below.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note:</strong> Fields marked as SVR: should only be modified by server code.</td>
</tr>
<tr>
<td>ESS_USHORT_T</td>
<td>meta_direction_type</td>
<td>Source of metadata identified by one of the Direction constants listed below.</td>
</tr>
</tbody>
</table>
## Data Type
- **Field**
- **Description**

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>The following fields are applicable for replication sources</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ESS_BOOL_T</td>
<td>incr_refresh</td>
<td>SVR: incr. refresh allowed?</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>The following fields are applicable for all targets</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;ESS_PARTDEF_MAP_T&quot; on page 157</td>
<td>partition_map</td>
<td>Main shared partition member map.</td>
</tr>
<tr>
<td>&quot;ESS_PARTDEF_MAP_T&quot; on page 157</td>
<td>slice_maps</td>
<td>Slice-specific mappings.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>The following fields are applicable to replication targets</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ESS_TIME_T</td>
<td>last_refreshed</td>
<td>SVR: time of last refresh.</td>
</tr>
<tr>
<td>ESS_BOOL_T</td>
<td>updatable</td>
<td>Is data at target updatable?</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>The following fields are applicable to link targets</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ESS_CHAR_T</td>
<td>defaultuser [ESS_USERNAMELEN]</td>
<td>Default username</td>
</tr>
<tr>
<td>ESS_CHAR_T</td>
<td>defaultpass [ESS_PASSWORDLEN]</td>
<td>Default password</td>
</tr>
</tbody>
</table>

### Define ESS_PARTITION_OP_ALLTYPES
```
define ESS_PARTITION_OP_REPLICATED 0x0001
define ESS_PARTITION_OP_LINKED 0x0002
define ESS_PARTITION_OP_TRANSPARENT 0x0004
define ESS_PARTITION_OP_ALLTYPES (ESS_PARTITION_OP_REPLICATED |
                                      ESS_PARTITION_OP_LINKED |
                                      ESS_PARTITION_OP_TRANSPARENT)
```

### Define ESS_PARTITION_DATA_BOTH
```
define ESS_PARTITION_DATA_SOURCE 0x0001
define ESS_PARTITION_DATA_TARGET 0x0002
define ESS_PARTITION_DATA_BOTH (ESS_PARTITION_DATA_SOURCE |
                                  ESS_PARTITION_DATA_TARGET)
```

## ESS_PARTHDR_T
Specifies an Essbase database and application.

```
typedef struct ESS_PARTHDR_T
{
  ESS_SVRNAME_T           zServer;
  ESS_APPNAME_T           zApplication;
  ESS_DBNAME_T            zDatabase;
  ESS_USERNAME_T          zUser;
  ESS_TIME_T              tTime;
} ESS_PARTHDR_T, *ESS_PPARTHDR_T, *ESS_PPPARTHDR_T;
```

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_SVRNAME_T</td>
<td>zServer</td>
<td>The server name.</td>
</tr>
<tr>
<td>ESS_APPNAME_T</td>
<td>zApplication</td>
<td>The application name.</td>
</tr>
<tr>
<td>Data Type</td>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>ESS_DBNAME_T</td>
<td>zDatabase</td>
<td>The database name.</td>
</tr>
<tr>
<td>ESS_USERNAME_T</td>
<td>zUser</td>
<td>The user name.</td>
</tr>
<tr>
<td>ESS_TIME_T</td>
<td>tTime</td>
<td>Last restructure affecting this partition.</td>
</tr>
</tbody>
</table>

**ESS_PARTOTL_CHANGE_API_T**

typedef struct ESS_PARTOTL_CHANGE_API_T
{
  ESS_ULONG_T ulDimensionCount;
  ESS_PPARTOTL_DIMCHG_API_T pDimchg;
  ESS_ULONG_T ulAliasTableCount;
  ESS_PPARTOTL_NAMEMAP_API_T pAliasTableChg;
} ESS_PARTOTL_CHANGE_API_T, *ESS_PPARTOTL_CHANGE_API_T, **ESS_PPPARTOTL_CHANGE_API_T;

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_ULONG_T</td>
<td>ulDimensionCount</td>
<td>Number of dimension changes.</td>
</tr>
<tr>
<td>&quot;ESS_PARTOTL_DIMCHG_API_T&quot; on page 162</td>
<td>pDimchg</td>
<td>Pointer to a link list of dimension changes.</td>
</tr>
<tr>
<td>ESS_ULONG_T</td>
<td>ulAliasTableCount</td>
<td>Count of alias table changes.</td>
</tr>
<tr>
<td>&quot;ESS_PARTOTL_NAMEMAP_API_T&quot; on page 167</td>
<td>pAliasTableChg</td>
<td>Linked list of table changes.</td>
</tr>
</tbody>
</table>

Notes

The ESS_PARTOTL_CHANGE_API_T structure categorizes database outline changes by dimensions. This structure is passed in when EssSmDbOtlRestruct() is called. An outline change is composed of a set of dimension changes and a set of alias table changes. Dimension changes are passed as a linked list pointed to by pDimChg. Each item in the linked list represents the changes made to the dimension; it also has a root pointer pMemberChange which points to a linked list of member changes.

Alias table changes are passed as a linked list pointed to by pAliasTableChg. Each item in the linked list represents the changes in an alias table. Currently, only Add, and Delete operations are supported. The following highlights the alias table change operations:

When an alias table is deleted, changed records show an alias table deletion. There is no change record for any alias which is deleted along with the alias table. Alias changes are recorded as member updates. Alias changes are reflected regardless of the status of the alias table, that is, the alias table does not have to be "active".

Renaming an alias table is interpreted as deleting an alias table with the old name and adding an alias table with the new name. Aliases in the renamed alias table are new aliases.

**ESS_PARTOTL_CHG_FILE_T**

Specifies metadata change files.
typedef struct ESS_PARTOTL_CHG_FILE_T
{
    ESS_USHORT_T  usFileNum;
    ESS_PSTR_T    ppszFileName;
} ESS_PARTOTL_CHG_FILE_T, *ESS_PPARTOTL_CHG_FILE_T, **ESS_PPPARTOTL_CHG_FILE_T;

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_USHORT_T</td>
<td>usFileNum</td>
<td>Number of meta change files.</td>
</tr>
<tr>
<td>ESS_PSTR_T</td>
<td>ppszFileName</td>
<td>Array of meta change file names.</td>
</tr>
</tbody>
</table>

**ESS_PARTOTL_DIM_ATTRIB_API_T**

Specifies the attributes of the specified dimension.

typedef struct ESS_PARTOTL_DIM_ATTRIB_API_T
{
    ESS_USHORT_T                     usDimType;
    ESS_USHORT_T                     usDimTag;
    ESS_ULONG_T                      ulOldDimNo;
    ESS_ULONG_T                      ulNewDimNo;
    ESS_ULONG_T                      ulNamedLevNum;
    ESS_PARTOTL_NAMED_GENLEV_API_T  *pNamedLev;
    ESS_ULONG_T                      ulNamedGenNum;
    ESS_PARTOTL_NAMED_GENLEV_API_T  *pNamedGen;
    ESS_STR_T                        pszBasememberName;
    ESS_STR_T                        pszOldName;
    ESS_STR_T                        pszNewName;
} ESS_PARTOTL_DIM_ATTRIB_API_T, *ESS_PPARTOTL_DIM_ATTRIB_API_T,
**ESS_PPPARTOTL_DIM_ATTRIB_API_T;
<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_STR_T</td>
<td>pszOldName</td>
<td>The old dimension name.</td>
</tr>
<tr>
<td>ESS_STR_T</td>
<td>pszNewName</td>
<td>The new dimension names pszOldName and pszNewName are used only for rename. Note that a dimension rename implies renaming both the dimension and the top-most member in this dimension.</td>
</tr>
</tbody>
</table>

### Dimension Type Constants (usDimType)

```c
#define ESS_DIMTYPE_DENSE  0
#define ESS_DIMTYPE_SPARSE 1
```

### Dimension Tag Constants (usDimTag)

```c
#define ESS_TTYPE_NONE          0
#define ESS_TTYPE_CCATEGORY     1       /* Accounts - currency ACCOUNTS tag */
#define ESS_TTYPE_CNAME         2       /* Country - currency COUNTRY tag */
#define ESS_TTYPE_CTIME         3       /* Time - currency TIME tag */
#define ESS_TTYPE_CTYPE         4       /* Type - currency TYPE tag */
#define ESS_TTYPE_CPARTITION    5       /* Currency Partition tag */
#define ESS_TTYPE_ATTRIBUTE     6       /* Attribute tag */
#define ESS_TTYPE_ATTRCALC      7       /* Attribute calc tag(Internal) */
```

### ESS_PARTOTL_DIMASSOCCHG_API_T

Contains information on the attribute dimension name and level as well as the dimension association change type.

```c
typedef struct ESS_PARTOTL_DIMASSOCCHG_API_T
{
    ESS_SHORT_T                           usDimAssocChgType;
    ESS_CHAR_T                           *pszAttrDimName;
    ESS_SHORT_T                           usLevel;
    struct ess_partotl_dimassocchg_api_t *pNext;
} ESS_PARTOTL_DIMASSOCCHG_API_T;
```

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_SHORT_T</td>
<td>usDimAssocChgType</td>
<td>Dimension association change type</td>
</tr>
<tr>
<td>ESS_CHAR_T</td>
<td>pszAttrDimName</td>
<td>Attribute dimension name</td>
</tr>
<tr>
<td>ESS_SHORT_T</td>
<td>usLevel</td>
<td>Dimension association level</td>
</tr>
<tr>
<td>ESS_PARTOTL_DIMASSOCCHG_API_T</td>
<td>pNext</td>
<td>Pointer to the next structure</td>
</tr>
</tbody>
</table>

### ESS_PARTOTL_DIMCHG_API_T

Specifies a change to the outline, specifically a change to a dimension.
typedef struct ESS_PARTOTL_DIMCHG_API_T
{
    ESS_USHORT_T                     usDimChgType;
    ESS_PARTOTL_DIM_ATTRIB_API_T     DimAttribute;
    ESS_PARTOTL_MBR_RSRVD_API_T     MemberReserved;
    ESS_ULONG_T                      ulMemberChanges;
    ESS_PPARTOTL_MBRCHG_API_T        pMemberChange;
    ESS_USHORT_T                     usAttrType;
    ESS_USHORT_T                     usDimAssocChgCnt;
    ESS_PARTOTL_DIMASSOCCHG_API_T   *pDimAssocChg;
    struct ess_partotl_dimchg_api_t *pNext;
} ESS_PARTOTL_DIMCHG_API_T, *ESS_PPARTOTL_DIMCHG_API_T, **ESS_PPPARTOTL_DIMCHG_API_T;

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_USHORT_T</td>
<td>usDimChgType</td>
<td>One of the dimension change (ESS_OTL_DIMCHG_T) constants listed below</td>
</tr>
<tr>
<td>&quot;ESS_PARTOTL_DIM_ATTRIB_API_T&quot; on page 161</td>
<td>DimAttribute</td>
<td>Dimension attributes</td>
</tr>
<tr>
<td>&quot;ESS_PARTOTL_MBR_RSRVD_API_T&quot; on page 164</td>
<td>MemberReserved</td>
<td>Reserved</td>
</tr>
<tr>
<td>ESS_ULONG_T</td>
<td>ulMemberChanges</td>
<td>Number of member changes</td>
</tr>
<tr>
<td>&quot;ESS_PARTOTL_MBRCHG_API_T&quot; on page 165</td>
<td>pMemberChange</td>
<td>Pointer to the linked list of member changes</td>
</tr>
<tr>
<td>ESS_USHORT_T</td>
<td>usAttrType</td>
<td>Attribute type</td>
</tr>
<tr>
<td>ESS_USHORT_T</td>
<td>usDimAssocChgCnt</td>
<td>Number of dimension associations</td>
</tr>
<tr>
<td>&quot;ESS_PARTOTL_DIMASSOCCHG_API_T&quot; on page 162</td>
<td>pDimAssocChg</td>
<td>Linked list of dimension associations</td>
</tr>
<tr>
<td>ESS_PARTOTL_DIMCHG_API_T</td>
<td>pNext</td>
<td>Pointer to the next dimension change</td>
</tr>
</tbody>
</table>

### Dimension Change (ESS_OTL_DIMCHG_T) Constants

The following constants are defined for the `usDimChgType` field of the ESS_PARTOTL_DIMCHG_API_T structure:

- ESS_PARTITION_OTLDM_ADD     /* Add dimensions */
- ESS_PARTITION_OTLDM_DELETE  /* Delete dimensions */
- ESS_PARTITION_OTLDM_UPDATE  /* Update dimensions */
- ESS_PARTITION_OTLDM_MOVE    /* Move dimensions */
- ESS_PARTITION_OTLDM_RENAME  /* Rename dimensions */
- ESS_PARTITION_OTLDM_MBRCHG  /* */
- ESS_PARTITION_OTLDM_ALL     /* All of the above */
**ESS_PARTOTL_MBR_RSRVD_API_T**

Specifies reserved member operations.

typedef struct ESS_PARTOTL_MBR_RSRVD_API_T
{
    ESS_BOOL_T            breject;
    ESS_PARTOTL_OSN_RELATIVES_API_T *pSrcRelatives;
    ESS_PARTOTL_OSN_RELATIVES_API_T *pDstRelatives;
    ESS_VOID_T            *unused;
} ESS_PARTOTL_MBR_RSRVD_API_T, *ESS_PPARTOTL_MBR_RSRVD_API_T,
**ESS_PPPARTOTL_MBR_RSRVD_API_T**;

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_BOOL_T</td>
<td>breject</td>
<td>TRUE rejects this record (for Outline Synchronization only).</td>
</tr>
<tr>
<td>&quot;ESS_PARTOTL_OSN_RELATIVES_API_T&quot; on page 168</td>
<td>*pSrcRelatives</td>
<td>Source parent and sibling</td>
</tr>
<tr>
<td>&quot;ESS_PARTOTL_OSN_RELATIVES_API_T&quot; on page 168</td>
<td>*pDstRelatives</td>
<td>Destination parent and sibling</td>
</tr>
<tr>
<td>ESS_VOID_T</td>
<td>*unused</td>
<td>(Unused)</td>
</tr>
</tbody>
</table>

**ESS_PARTOTL_MBRASSOCCHG_API_T**

Contains information on the attribute dimension and member name, as well as the attribute value.

typedef struct ESS_PARTOTL_MBRASSOCCHG_API_T
{
    ESS_CHAR_T                         *pszAttrDimName;
    ESS_CHAR_T                         *pszAttrMbrName;
    ESS_CHAR_T                         *pszAttrParName;
    ESS_ATTRIBUTEVALUE_T              AttrValue;
    struct ess_partotl_mbrassocchg_api_t *pNext;
} ESS_PARTOTL_MBRASSOCCHG_API_T;

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_CHAR_T</td>
<td>pszAttrDimName</td>
<td>Attribute dimension name</td>
</tr>
<tr>
<td>ESS_CHAR_T</td>
<td>pszAttrMbrName</td>
<td>Attribute member name</td>
</tr>
<tr>
<td>ESS_CHAR_T</td>
<td>pszAttrParName</td>
<td>Attribute parent name</td>
</tr>
<tr>
<td>&quot;ESS_ATTRIBUTEVALUE_T&quot; on page 118</td>
<td>AttrValue</td>
<td>Attribute value</td>
</tr>
<tr>
<td>ESS_PARTOTL_MBRASSOCCHG_API_T</td>
<td>*pNext</td>
<td>Pointer to the next structure</td>
</tr>
</tbody>
</table>

**ESS_PARTOTL_MBRATTR_API_T**

Stores member attribute information.
typedef struct ESS_PARTOTL_MBRATTR_API_T
{
    ESS_STS_T                      status;
    ESS_SHORT_T                    level;
    ESS_SHORT_T                    generation;
    ESS_CHAR_T                     *calc;
    ESS_SHORT_T                    ucal;
    ESS_USHORT_T                   atype;
    ESS_BOOL_T                     nocconvert;
    ESS_CHAR_T                     *crMbrName;
    ESS_PARTOTL_NAMECHG_API_T      *pUdaChange;
    ESS_PARTOTL_NAMECHG_API_T      *pAliasChange;
}  ESS_PARTOTL_MBRATTR_API_T,*ESS_PPARTOTL_MBRATTR_API_T,**ESS_PPPARTOTL_MBRATTR_API_T;

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_STS_T</td>
<td>status</td>
<td>Member status.</td>
</tr>
<tr>
<td>ESS_SHORT_T</td>
<td>level</td>
<td>Level number.</td>
</tr>
<tr>
<td>ESS_SHORT_T</td>
<td>generation</td>
<td>Generation.</td>
</tr>
<tr>
<td>ESS_CHAR_T</td>
<td>calc</td>
<td>Calculation equation.</td>
</tr>
<tr>
<td>ESS_SHORT_T</td>
<td>ucalc</td>
<td>Unary calculation symbol for this member.</td>
</tr>
<tr>
<td>ESS_USHORT_T</td>
<td>atype</td>
<td>A 16 bit mask for members of the dimension tagged as ACCOUNT. This is not used elsewhere. By default, they are all OFF.</td>
</tr>
<tr>
<td>ESS_BOOL_T</td>
<td>nocconvert</td>
<td>Default to FALSE, do currency conversion.</td>
</tr>
<tr>
<td>ESS_CHAR_T</td>
<td>crMbrName</td>
<td>The name of the tagged currency database member</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FOR TIME -- tagged Time Member</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FOR COUNTRY -- tagged currency Member</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FOR ACCOUNTS -- tagged category Member</td>
</tr>
<tr>
<td>&quot;ESS_PARTOTL_NAMECHG_API_T&quot; on page 167</td>
<td>pUdaChange</td>
<td>User defined attributes changes.</td>
</tr>
<tr>
<td>&quot;ESS_PARTOTL_NAMECHG_API_T&quot; on page 167</td>
<td>pAliasChange</td>
<td>Alias changes.</td>
</tr>
</tbody>
</table>

**ESS_PARTOTL_MBRCHG_API_T**

Specifies a member change operation.

typedef struct ESS_PARTOTL_MBRCHG_API_T
{
    ESS_ULONG_T                      ulOperator;
    ESS_CHAR_T                      *pszOperand1;
    ESS_CHAR_T                      *pszOperand2;
    ESS_CHAR_T                      *pszOperand3;
    ESS_CHAR_T                      *pszOperand4;
    ESS_ULONG_T                      ulOperand1;
    ESS_PARTOTL_MBRATTR_API_T       *pMemberAttribute;
    ESS_PARTOTL_MBR_RSRVD_API_T     MemberReserved;
    ESS_ULONG_T                      ulMbrAssocChgCnt;
}
ESS_PARTOTL_MBRASSOCCHG_API_T *pMbrAssocChg;
struct ess_partotl_mbrchg_api_t *pNext;
} ESS_PARTOTL_MBRCHG_API_T,*ESS_PPARTOTL_MBRCHG_API_T,**ESS_PPPARTOTL_MBRCHG_API_T;

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_ULONG_T</td>
<td>ulOperator</td>
<td>One of the member change (ESS_MBR_CHANGE_T) constants listed below</td>
</tr>
<tr>
<td>ESS_CHAR_T</td>
<td>pszOperand1</td>
<td>Alphabetic operand 1</td>
</tr>
<tr>
<td>ESS_CHAR_T</td>
<td>pszOperand2</td>
<td>Alphabetic operand 2</td>
</tr>
<tr>
<td>ESS_CHAR_T</td>
<td>pszOperand3</td>
<td>Alphabetic operand 3</td>
</tr>
<tr>
<td>ESS_CHAR_T</td>
<td>pszOperand4</td>
<td>Alphabetic operand 4</td>
</tr>
<tr>
<td>ESS_ULONG_T</td>
<td>ulOperand1</td>
<td>A bit-field operand that indicates updated attributes of the given member.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This field is only used when the member change operator is ESS_PARTITION_OTLMBR_UPDATE.</td>
</tr>
</tbody>
</table>

*pMemberAttribute The pointer to a member attribute structure. The value is null for delete and rename.

*ESS_PARTOTL_MBRASSOCCHG_API_T* on page 164

**ESS_PARTOTL_MBRASSOCCHG_API_T** on page 164

ESS_PARTITION_OTLMBR_ADD             /* Add members                     */
ESS_PARTITION_OTLMBR_DELETE          /* Delete members                  */
ESS_PARTITION_OTLMBR_RENAME          /* Rename members                  */
ESS_PARTITION_OTLMBR_MOVE            /* Move members                    */
ESS_PARTITION_OTLMBR_UPDATE          /* Update members                  */
ESS_PARTITION_OTLMBRATTR_STATUS      /* Status changes                  */
ESS_PARTITION_OTLMBRATTR_ALIAS       /* Alias changes                    */
ESS_PARTITION_OTLMBRATTR_UCALC       /* Unary calc symbol changes       */
ESS_PARTITION_OTLMBRATTR_ATYPE       /* Account type changes            */
ESS_PARTITION_OTLMBRATTR_CCURRENCY   /* Currency conversion flag        */
ESS_PARTITION_OTLMBRATTR_CRMNAME     /* Tagged currency database member */
ESS_PARTITION_OTLMBRATTR_CUDATA     /* User defined attribute changes  */
ESS_PARTITION_OTLMBRATTR_CALC        /* Calc formula changes            */
ESS_PARTITION_OTLMBRATTR_LEVEL       /* Level number changes            */
ESS_PARTITION_OTLMBRATTR_GENERATION  /* Generation number changes       */
ESS_PARTITION_OTLMBRATTR_ATTRIBUTE   /* Attribute changes                */
ESS_PARTITION_OTLMBRATTR_ALL         /* All of the above                 */

Member Change (ESS_MBR_CHANGE_T) Constants

The following constants are defined for the ulOperator field of the ESS_PARTOTL_MBRCHG_API_T structure:
**ESS_PARTOTL_NAMECHG_API_T**

Records name changes.

typedef struct ESS_PARTOTL_NAMECHG_API_T
{
    ESS_USHORT_T               usCount;
    ESS_PPARTOTL_NAMEMAP_API_T pNameMap;
} ESS_PARTOTL_NAMECHG_API_T, *ESS_PPARTOTL_NAMECHG_API_T, **ESS_PPPARTOTL_NAMECHG_API_T;

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_USHORT_T</td>
<td>usCount</td>
<td>The number of changes.</td>
</tr>
<tr>
<td>&quot;ESS_PARTOTL_NAMEMAP_API_T&quot; on page 167</td>
<td>pNameMap</td>
<td>Array of name maps.</td>
</tr>
</tbody>
</table>

---

**ESS_PARTOTL_NAMED_GENLEV_API_T**

Specifies a name for a level or generation.

typedef struct ESS_PARTOTL_NAMED_GENLEV_API_T
{
    ESS_USHORT_T                           usOperator;
    ESS_SHORT_T                            sGenLev;
    ESS_STR_T                              pszName;
    struct ess_partotl_named_genlev_api_t *pNext;
} ESS_PARTOTL_NAMED_GENLEV_API_T, *ESS_PPARTOTL_NAMED_GENLEV_API_T, **ESS_PPPARTOTL_NAMED_GENLEV_API_T;

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_USHORT_T</td>
<td>usOperator</td>
<td>One of the Name Operation Type constants listed below.</td>
</tr>
<tr>
<td>ESS_SHORT_T</td>
<td>sGenLev</td>
<td>Generation or Level number.</td>
</tr>
<tr>
<td>ESS_STR_T</td>
<td>pszName</td>
<td>Generation or Level name.</td>
</tr>
<tr>
<td>ESS_PARTOTL_NAMED_GENLEV_API_T</td>
<td>pNext</td>
<td>Pointer to the next structure.</td>
</tr>
</tbody>
</table>

**Name Operation Type Constants**

#define ESS_NAME_ADD    0x01
#define ESS_NAME_DELETE 0x02
#define ESS_NAME_UPDATE 0x04

---

**ESS_PARTOTL_NAMEMAP_API_T**

Charts name changes.

typedef struct ESS_PARTOTL_NAMEMAP_API_T
{
    ESS_USHORT_T               usOperator;
    ESS_CHAR_T                 *name;
    ESS_CHAR_T                 *name2;
}
struct ess_partotl_namemap_api_t  *pNext;

}  ESS_PARTOTL_NAMEMAP_API_T, *ESS_PPARTOTL_NAMEMAP_API_T,
**ESS_PPPARTOTL_NAMEMAP_API_T;

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_USHORT_T</td>
<td>usOperator</td>
<td>One of the Name Operation Type constants listed below.</td>
</tr>
<tr>
<td>ESS_CHAR_T</td>
<td>name</td>
<td>Name of uda, for alias changes, the alias table name.</td>
</tr>
<tr>
<td>ESS_CHAR_T</td>
<td>name2</td>
<td>Not used for uda changes, for alias changes, use the alias name.</td>
</tr>
<tr>
<td>ESS_PARTOTL_NAMEMAP_API_T</td>
<td>pNext</td>
<td>Pointer to the next structure.</td>
</tr>
</tbody>
</table>

Name Operation Type Constants

#define ESB_NAME_ADD 0x01
#define ESB_NAME_DELETE 0x02
#define ESB_NAME_UPDATE 0x04

**ESS_PARTOTL_OSN_RELATIVES_API_T**

Contains the names of the member, its parent, and its siblings.

typedef struct ESS_PARTOTL_OSN_RELATIVES_API_T
{
    ESS_UCHAR_T        statuses[ESS_PARTOTL_OSN_NUM_RELATIVES];
    ESS_PCHAR_T        names[ESS_PARTOTL_OSN_NUM_RELATIVES];
    ESS_ATTRIBUTEVALUE_T values[ESS_PARTOTL_OSN_NUM_RELATIVES];
} ESS_PARTOTL_OSN_RELATIVES_API_T;

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_UCHAR_T</td>
<td>statuses</td>
<td>An array containing the status of each relative</td>
</tr>
<tr>
<td>ESS_PCHAR_T</td>
<td>names</td>
<td>An array containing the name of each relative</td>
</tr>
<tr>
<td>&quot;ESS_ATTRIBUTEVALUE_T&quot; on page 118</td>
<td>values</td>
<td>An array containing the attribute value structure for each relative</td>
</tr>
</tbody>
</table>

Constants for ESS_PARTOTL_OSN_RELATIVES_API_T

typedef enum ESS_PARTOTL_OSN_REL_TYPE_API_T (Indices for the statuses, names and values arrays)
{
    ESS_PARTOTL_OSN_MEMBER
    ESS_PARTOTL_OSN_PARENT
    ESS_PARTOTL_OSN_LSIBLING
    ESS_PARTOTL_OSN_RSIBLING
    ESS_PARTOTL_OSN_REGION_PARENT
    ESS_PARTOTL_OSN_LEVEL_REGION_LSIBLING
    ESS_PARTOTL_OSN_LEVEL_REGION_RSIBLING
    ESS_PARTOTL_OSN_GENER_REGION_LSIBLING
    ESS_PARTOTL_OSN_GENER_REGION_RSIBLING
    ESS_PARTOTL_OSN_RESERVED1

168  C Main API Declarations
typedef enum ESS_PARTOTL_OSN_REL_TYPE_API_T (Values for statuses) {
    ESS_PARTOTL_OSN_REL_NONE
    ESS_PARTOTL_OSN_REL_SAME_AS_ADJACENT  /* The name of the region sibling is the same as the name of the sibling. */
    ESS_PARTOTL_OSN_REL_SHARED
    ESS_PARTOTL_OSN_REL_REAL
} ESS_PARTOTL_QUERY_T

Queries metadata changes.

typedef struct ESS_PARTOTL_QUERY_T
{
    ESS_PART_CONNECT_INFO_T HostDatabase;
    ESS_USHORT_T usOperationType;
    ESS_USHORT_T, usDataDirectionType;
    ESS_PARTOTL_QRY_FILTER_T MetaFilter;
} ESS_PARTOTL_QUERY_T, *ESS_PPARTOTL_QUERY_T, **ESS_PPPARTOTL_QUERY_T;

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_USHORT_T</td>
<td>usOperationType</td>
<td>One of the Operation Type constants listed below.</td>
</tr>
<tr>
<td>ESS_USHORT_T</td>
<td>usDataDirectionType</td>
<td>One of the Direction Type constants listed below.</td>
</tr>
<tr>
<td>ESS_PARTOTL_QRY_FILTER_T</td>
<td>MetaFilter</td>
<td>Criteria to further define names.</td>
</tr>
</tbody>
</table>

Operation Type Constants

#define ESS_PARTITION_OP_REPLICATED 0x0001
#define ESS_PARTITION_OP_LINKED 0x0002
#define ESS_PARTITION_OP_TRANSPARENT 0x0004
#define ESS_PARTITION_OP_ALLTYPES (ESS_PARTITION_OP_REPLICATED | ESS_PARTITION_OP_LINKED | ESS_PARTITION_OP_TRANSPARENT)

Direction Type Constants

#define ESS_PARTITION_DATA_SOURCE 0x0001
#define ESS_PARTITION_DATA_TARGET 0x0002
# ESS_PARTOTL_QRY_FILTER_T

Further defines the metadata retrieval criteria.

```c
typedef struct ESS_PARTOTL_QRY_FILTER_T
{
    ESS_TIME_T         TimeStamp;
    ESS_ULONG_T        ulDimFilter;
    ESS_ULONG_T        ulMbrFilter;
    ESS_ULONG_T        ulMbrAttrFilter;
} ESS_PARTOTL_QRY_FILTER_T, *ESS_PPARTOTL_QRY_FILTER_T, **ESS_PPPARTOTL_QRY_FILTER_T;
```

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_TIME_T</td>
<td>TimeStamp</td>
<td>Query meta change happens after this time.</td>
</tr>
<tr>
<td>ESS_ULONG_T</td>
<td>ulDimFilter</td>
<td>Bitfield to select dimension changes.</td>
</tr>
<tr>
<td>ESS_ULONG_T</td>
<td>ulMbrFilter</td>
<td>Bitfield to select member changes.</td>
</tr>
<tr>
<td>ESS_ULONG_T</td>
<td>ulMbrAttrFilter</td>
<td>Bitfield to select member attribute changes.</td>
</tr>
</tbody>
</table>

## Member Attribute Change Constants

```c
#define ESS_PARTITION_OTLMBRATTR_STATUS    0x0001  /* status changes */
#define ESS_PARTITION_OTLMBRATTR_ALIAS      0x0002  /* alias changes */
#define ESS_PARTITION_OTLMBRATTR_UCALC      0x0004  /* unary calc symbol changes */
#define ESS_PARTITION_OTLMBRATTR_ATYPE      0x0008  /* account type changes */
#define ESS_PARTITION_OTLMBRATTR_CCONVERT   0x0010  /* currency conversion flag */
#define ESS_PARTITION_OTLMBRATTR_CRMBRNAME  0x0020  /* tagged currency db member */
#define ESS_PARTITION_OTLMBRATTR_UDA        0x0040  /* user defined attribute changes */
#define ESS_PARTITION_OTLMBRATTR_CALC       0x0080  /* calc formula changes */
#define ESS_PARTITION_OTLMBRATTR_LEVEL      0x0100  /* level number changes */
#define ESS_PARTITION_OTLMBRATTR_GENERATION 0x0200  /* generation number changes */
#define ESS_PARTITION_OTLMBRATTR_ALL        (ESS_PARTITION_OTLMBRATTR_STATUS |
                                          ESS_PARTITION_OTLMBRATTR_ALIAS |
                                          ESS_PARTITION_OTLMBRATTR_UCALC |
                                          ESS_PARTITION_OTLMBRATTR_ATYPE |
                                          ESS_PARTITION_OTLMBRATTR_CCONVERT |
                                          ESS_MBRATTR_CRMBR_NAME |
                                          ESS_PARTITION_OTLMBRATTR_UDA |
                                          ESS_PARTITION_OTLMBRATTR_CALC |
                                          ESS_PARTITION_OTLMBRATTR_LEVEL |
                                          ESS_PARTITION_OTLMBRATTR_GENERATION)
```

```c
#define ESS_ALLCHG       (ESS_PARTITION_OTLMBR_ALL | ESS_DIMCHG_ALL)
```

# ESS_PARTOTL_READ_T

Reads metadata changes.

```c
typedef struct ESS_PARTOTL_READ_T
{
    ESS_PPARTOTL_CHANGE_API_T  pOtlChg;
    ESS_TIME_T                 SourceTime;
} ESS_PARTOTL_READ_T, *ESS_PPARTOTL_READ_T, **ESS_PPPARTOTL_READ_T;
```
<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;ESS_PARTOTL_CHANGE_API_T&quot; on page 160</td>
<td>pOtlChg</td>
<td>Meta change records.</td>
</tr>
<tr>
<td>ESS_TIME_T</td>
<td>SourceTime</td>
<td>Time when source outline is changed.</td>
</tr>
</tbody>
</table>

**ESS_PARTOTL_SELECT_APPLY_T**

Applies metadata changes.

typedef struct ESS_PARTOTL_SELECT_APPLY_T
{
    ESS_STR_T                  pszFileName;
    ESS_PPARTOTL_CHANGE_API_T  pOtlChg;
    ESS_TIME_T                 SourceTime;
} ESS_PARTOTL_SELECT_APPLY_T, *ESS_PPARTOTL_SELECT_APPLY_T, **ESS_PPPARTOTL_SELECT_APPLY_T;

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_STR_T</td>
<td>pszFileName</td>
<td>Outline change file name.</td>
</tr>
<tr>
<td>&quot;ESS_PARTOTL_CHANGE_API_T&quot; on page 160</td>
<td>pOtlChg</td>
<td>Outline change records (from EssPartitionReadOtlChangeFile).</td>
</tr>
<tr>
<td>ESS_TIME_T</td>
<td>SourceTime</td>
<td>Timestamp from outline change source.</td>
</tr>
</tbody>
</table>

**ESS_PARTOTL_SELECT_CHG_T**

Queries metadata.

typedef struct ESS_PARTOTL_SELECT_CHG_T
{
    ESS_STR_T                pszFileName;
    ESS_PARTOTL_QRY_FILTER_T QueryFilter;
} ESS_PARTOTL_SELECT_CHG_T, *ESS_PPARTOTL_SELECT_CHG_T, **ESS_PPPARTOTL_SELECT_CHG_T;

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_STR_T</td>
<td>pszFileName</td>
<td>Meta change file name.</td>
</tr>
<tr>
<td>&quot;ESS_PARTOTL_QRY_FILTER_T&quot; on page 170</td>
<td>QueryFilter</td>
<td>Only reads records which satisfy the criteria.</td>
</tr>
</tbody>
</table>

**ESS_PARTSLCT_T**

Queries shared partitions for a given site.

typedef struct ESS_PARTSLCT_T
{
    ESS_USHORT_T       usOperationTypes;
    ESS_USHORT_T       usDirectionTypes;
}
ESS_USHORT_T usOperationTypes;

typedef union ESS_PPARTSLCT_VALIDATE_T,
    *ESS_PPARTSLCT_VALIDATE_T, **ESS_PPPARTSLCT_VALIDATE_T;

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_USHORT_T</td>
<td>usOperationTypes</td>
<td>One of the Operation Type constants listed below.</td>
</tr>
<tr>
<td>ESS_USHORT_T</td>
<td>usDirectionTypes</td>
<td>One of the Direction constants listed below.</td>
</tr>
</tbody>
</table>

**Operation Type Constants**

- 
  - 
  - defined ESS_PARTITION_OP_REPLICATED 0x0001
  - defined ESS_PARTITION_OP_LINKED 0x0002
  - defined ESS_PARTITION_OP_TRANSPARENT 0x0004
  - defined ESS_PARTITION_OP_ALLTYPES (ESS_PARTITION_OP_REPLICATED |
    ESS_PARTITION_OP_LINKED |
    ESS_PARTITION_OP_TRANSPARENT)

**Direction Constants**

- 
  - defined ESS_PARTITION_DATA_SOURCE 0x0001
  - defined ESS_PARTITION_DATA_TARGET 0x0002
  - defined ESS_PARTITION_DATA_BOTH (ESS_PARTITION_DATA_SOURCE | |
    ESS_PARTITION_DATA_TARGET)

**ESS_PARTSLCT_VALIDATE_T**

Specifies a partition to verify.

typedef struct ESS_PARTSLCT_VALIDATE_T
{
    ESS_USHORT_T     usLoc;
    ESS_STR_T        pszFileName;
    ESS_PART_DEFINED_T Part;
} ESS_PARTSLCT_VALIDATE_T, *ESS_PPARTSLCT_VALIDATE_T, **ESS_PPPARTSLCT_VALIDATE_T;

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_USHORT_T</td>
<td>usLoc</td>
<td>Either ESS_FILE_CLIENT or ESS_FILE_SERVER</td>
</tr>
<tr>
<td>ESS_STR_T</td>
<td>pszFileName</td>
<td>Partition definition file name.</td>
</tr>
<tr>
<td>&quot;ESS_PART_DEFINED_T&quot; on page 153</td>
<td>Partition</td>
<td>Partition to verify.</td>
</tr>
</tbody>
</table>

**ESS_PERF_ALLOC_ARG_T**

This structure contains information about where errors occur for allocations or custom calculations.

typedef enum ESS_PERF_ALLOC_ARG_T
{
    ESS_PERF_ALLOC_ARG_NA, 0,
    ESS_PERF_ALLOC_ARG_POV, 1,
    ESS_PERF_ALLOC_ARG_AMOUNT, 2,

172  C Main API Declarations
This structure returns information about warnings and errors returned by the allocations functions. This information is used by the calling function to determine which argument has an error and on which line number and token it occurs. Only some warnings or errors will generate an error structure. The messageNumber indicates which structure goes with which message. If more than one message has the same number, then the corresponding error structures (if any) will be in the same order in which the messages were given.

typedef struct ESS_PERF_ALLOC_ERROR_T
{
    struct ESS_PERF_ALLOC_ERROR_T *nextError;
    ESS_ULONG_T messageNumber;
    ESS_PERF_ALLOC_ARG_T argument;
    ESS_ULONG_T lineNumber;
    ESS_CHAR_T token[8192];
} ESS_PERF_ALLOC_ERROR_T;

### ESS_PERF_ALLOC_ERROR_T

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_PERF_ALLOC_ERROR_T</td>
<td>nextError</td>
<td>Pointer to the next error structure, if any</td>
</tr>
<tr>
<td>ESS_ULONG_T</td>
<td>messageNumber</td>
<td>The number of the corresponding error or warning message</td>
</tr>
<tr>
<td>ESS_PERF_ALLOC_ARG_T</td>
<td>argument</td>
<td>Indicates which parameter contains the error</td>
</tr>
<tr>
<td>ESS_ULONG_T</td>
<td>lineNumber</td>
<td>Indicates which line of the argument contains the error. If zero, this is not applicable.</td>
</tr>
</tbody>
</table>
Data Type | Field | Description
--- | --- | ---
| ESS_CHAR_T | token | Indicates which part of the argument contains a parsing error; empty if not applicable

**ESS_PERF_ALLOC_T**

This structure stores information to be used for performing allocations.

typedef struct ESS_PERF_ALLOC_T
{
    ESS_STR_T pov;
    ESS_STR_T amount;
    ESS_STR_T amountContext;
    ESS_STR_T amountTimeSpan;
    ESS_STR_T target;
    ESS_STR_T targetTimeSpan;
    ESS_ALLOCATION_TARGETTIMESPAN_OPTION targetTimeSpanOption;
    ESS_STR_T offset;
    ESS_STR_T debitMember;
    ESS_STR_T creditMember;
    ESS_STR_T range;
    ESS_STR_T excludedRange;
    ESS_STR_T basis;
    ESS_STR_T basisTimeSpan;
    ESS_ALLOCATION_BASISTIMESPAN_OPTION basisTimeSpanOption;
    ESS_ALLOCATION_METHOD_OPTION allocationMethod;
    ESS_ULONG_T spreadSkipOption;
    ESS_ALLOCATION_ZEROAMT_OPTION zeroAmountOption;
    ESS_ALLOCATION_ZEROBASIS_OPTION zeroBasisOption;
    ESS_ALLOCATION_NEGBASIS_OPTION negativeBasisOption;
    ESS_ALLOCATION_ROUND_OPTION roundMethod;
    ESS_STR_T roundDigits;
    ESS_STR_T roundToLocation;
    ESS_ULONG64_T groupId;
    ESS_ULONG64_T ruleID;
} ESS_PERF_ALLOC_T;
<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_ALLOCATION_</td>
<td>targetTimeSpanOption</td>
<td><strong>Optional:</strong> Specifies how values are allocated to <code>targetTimeSpan</code> members:</td>
</tr>
<tr>
<td>TARGETTIMESPAN_</td>
<td></td>
<td>- ESS_ASO_ALLOCATION_TIMESPAN_DIVIDEMT (divide)</td>
</tr>
<tr>
<td>OPTION_T</td>
<td></td>
<td>- ESS_ASO_ALLOCATION_TIMESPAN_REPEATAMT (repeat)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Ignored if empty</td>
</tr>
<tr>
<td>ESS_STR_T</td>
<td>offset</td>
<td><strong>Optional:</strong> MDX tuple expression specifying location for offsetting entries</td>
</tr>
<tr>
<td></td>
<td>debitMember</td>
<td><strong>Optional:</strong> MDX member expression specifying where positive result values should be written. If empty, debit/credit processing is not performed.</td>
</tr>
<tr>
<td></td>
<td>creditMember</td>
<td><strong>Optional:</strong> MDX member expression specifying where negative result values should be written. If empty, debit/credit processing is not performed.</td>
</tr>
<tr>
<td></td>
<td>range</td>
<td>MDX set expression specifying database region for allocation</td>
</tr>
<tr>
<td></td>
<td>excludedRange</td>
<td><strong>Optional:</strong> MDX set expression specifying a subset of <code>range</code>; a region included in the allocation but not written to</td>
</tr>
<tr>
<td></td>
<td>basis</td>
<td>MDX tuple expression specifying the basis location. If <code>allocationMethod = ESS_ASO_ALLOCATION_METHOD_SPREAD</code> and <code>spreadSkipOptions = 0</code>, then <code>basis</code> must be empty.</td>
</tr>
<tr>
<td></td>
<td>basisTimeSpan</td>
<td><strong>Optional:</strong> MDX set expression specifying time periods to be considered with <code>basis</code>. With <code>basisTimeSpanOption</code>, determines basis for allocation.</td>
</tr>
<tr>
<td>ESS_ALLOCATION_</td>
<td>basisTimeSpanOption</td>
<td><strong>Optional:</strong> Specifies how basis is computed across time periods from the following options:</td>
</tr>
<tr>
<td>BASISTIMESPAN_</td>
<td></td>
<td>- ESS_ASO_ALLOCATION_TIMESPAN_SPLITBASIS—Process basis value for each time period individually</td>
</tr>
<tr>
<td>OPTION_T</td>
<td></td>
<td>- ESS_ASO_ALLOCATION_TIMESPAN_COMBINEBASIS—Sum basis value across time periods in <code>basisTimeSpan</code> and use the combined basis for allocation</td>
</tr>
<tr>
<td></td>
<td>allocationMethod</td>
<td>Allocation method:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_ASO_ALLOCATION_METHOD_SHARE—allocate proportional to basis values</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_ASO_ALLOCATION_METHOD_SPREAD—allocate evenly across the target region</td>
</tr>
<tr>
<td>Data Type</td>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>---------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>ESS_ULONG_T</td>
<td>spreadSkipOption</td>
<td>Optional:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● If <code>allocationMethod</code> = ESS_ALLOCATION_METHOD_SHARE, then this value equals 0.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● If <code>allocationMethod</code> = ESS_ALLOCATION_METHOD_SPREAD, specifies which basis values should be skipped. Select one or more of the following bitwise arguments:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>○ ESS_ALLOCATION_SPREAD_SKIPMISSING—Excludes all cells in <code>allocationRange</code> for which the basis member is #missing</td>
</tr>
<tr>
<td></td>
<td></td>
<td>○ ESS_ALLOCATION_SPREAD_SKIPZERO—Excludes all cells in <code>allocationRange</code> for which the basisMbr is zero</td>
</tr>
<tr>
<td></td>
<td></td>
<td>○ ESS_ALLOCATION_SPREAD_SKIPNEGATIVE—Excludes all cells in <code>allocationRange</code> for which the basisMbr is negative</td>
</tr>
<tr>
<td></td>
<td></td>
<td>These arguments can be combined bitwise; for example ESS_ALLOCATION_SPREAD_SKIPZERO</td>
</tr>
<tr>
<td>ESS_ALLOCATION_ZEROAMT_OPTION_T</td>
<td>zeroAmountOption</td>
<td>Specifies what to do when an <code>amount</code> value is zero or #MISSING:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESS_ALLOCATION_ZEROAMT_DEFAULT—Allocate zero values</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESS_ALLOCATION_ZEROAMT_NEXTAMT—Skip to the next <code>amount</code> value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESS_ALLOCATION_ZEROAMT_ABORT—Cancel the entire allocation</td>
</tr>
<tr>
<td>ESS_ALLOCATION_ZEROBASIS_OPTION_T</td>
<td>zeroBasisOption</td>
<td>● If <code>allocationMethod</code> = ESS_ALLOCATION_METHOD_SHARE—Tells Essbase what to do when the aggregate sum of <code>basis</code> is zero</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● If <code>allocationMethod</code> = ESS_ALLOCATION_METHOD_SPREAD—Tells Essbase what to do when all <code>basis</code> values have been skipped</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Specify an option:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESS_ALLOCATION_ZEROBASIS_NEXTAMT—Skip to the next <code>amount</code> value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESS_ALLOCATION_ZEROBASIS_ABORT—Cancel the allocation</td>
</tr>
<tr>
<td>ESS_ALLOCATION_NEGBASIS_OPTION_T</td>
<td>negativeBasisOption</td>
<td>Tells Essbase what to do when a negative <code>basis</code> value is encountered:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESS_ALLOCATION_NEGBASIS_DEFAULT—Calculate as normal</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESS_ALLOCATION_NEGBASIS_NEXTAMT—Skip to next <code>amount</code> value. No data is allocated for the current <code>amount</code> value.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESS_ALLOCATION_NEGBASIS_ABORT—Cancel the allocation; no data is written.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The following values are only valid when <code>allocationMethod</code> = ESS_ALLOCATION_METHOD_SHARE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESS_ALLOCATION_NEGBASIS_ABS—Use the absolute value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESS_ALLOCATION_NEGBASIS_MISSING—Treat the basis as #missing</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESS_ALLOCATION_NEGBASIS_ZERO—Treat the basis value as zero</td>
</tr>
</tbody>
</table>
### ESS_ALLOCATION_ROUND_OPTION_T

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_ALLOCATION_ROUND_OPTION_T</td>
<td>roundMethod</td>
<td>Rounding method for allocated values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_ASO_ALLOCATION_ROUND_NONE – Perform no rounding</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_ASO_ALLOCATION_ROUND_DISCARDERRORS – Round, discarding rounding errors</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_ASO_ALLOCATION_ROUND_ERRORSTOHIGHEST – Round, adding rounding errors to the target cell with the greatest allocated value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_ASO_ALLOCATION_ROUND_ERRORSTOLOWEST – Round, adding rounding errors to the target cell with the lowest allocated value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_ASO_ALLOCATION_ROUND_ERRORSTOLOCATION – Round, adding rounding errors to <code>roundToLocation</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_STR_T</td>
<td>roundDigits</td>
<td>Must be empty if <code>roundMethod</code> = ESS_ASO_ALLOCATION_ROUND_NONE. Must be specified as a MDX numeric value or tuple expression. Value must be a whole number between 100 and -100.</td>
</tr>
<tr>
<td>ESS_STR_T</td>
<td>roundToLocation</td>
<td>Optional: If <code>roundMethod</code> = ESS_ASO_ALLOCATION_ROUND_ERRORSTOLOCATION, this is an MDX tuple expression specifying a location within <code>range</code>; empty otherwise</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_ULONG64_T</td>
<td>groupId</td>
<td>Internal use only. Always enter 0.</td>
</tr>
<tr>
<td>ESS_ULONG64_T</td>
<td>ruleID</td>
<td>Internal use only. Always enter 0.</td>
</tr>
</tbody>
</table>

### ESS_PERF_CUSTCALC_T

This structure stores information to be used for performing custom calculations with aggregate storage databases.

For complete information about writing and executing custom calculation scripts, see “Performing Custom Calculations and Allocations on Aggregate Storage Databases” in the Oracle Essbase Database Administrator’s Guide.

typedef struct ESS_PERF_CUSTCALC_T
{
    ESS_STR_T     pov;
    ESS_STR_T     script;
    ESS_STR_T     target;
    ESS_STR_T     debitMember;
    ESS_STR_T     creditMember;
    ESS_STR_T     offset;
    ESS_STR_T     sourceRegion;
    ESS_ULONG64_T  groupId;
    ESS_ULONG64_T  ruleID;
} ESS_PERF_CUSTCALC_T;

### ESS_STR_T

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_STR_T</td>
<td>pov</td>
<td>MDX set expression specifying script execution area within the database</td>
</tr>
<tr>
<td>Data Type</td>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ESS_STR_T</td>
<td>script</td>
<td>Contents of the custom calculation script. Should include multiple assignments of the form <code>Target := Formula;</code> where <code>Target</code> is an MDX tuple expression and <code>Formula</code> is an MDX numeric value expression. The script can contain only MDX tuple expressions and arithmetic operators (+, -, *, /). MDX functions are not supported.</td>
</tr>
<tr>
<td>ESS_STR_T</td>
<td>target</td>
<td>Optional: MDX tuple expression specifying location, in combination with <code>pov</code> and the left-hand side of the assignment statements in <code>script</code>, to which calculation results will be written. If dimensions overlap, the order for resolving conflicts is assignment statements first, then <code>target</code>, then <code>pov</code>.</td>
</tr>
<tr>
<td>ESS_STR_T</td>
<td>debitMember</td>
<td>Optional: MDX member expression specifying the debit member. Positive results are stored here. If empty, debit/credit processing is not performed.</td>
</tr>
<tr>
<td>ESS_STR_T</td>
<td>creditMember</td>
<td>Optional: MDX member expression specifying the credit member. Negative results will be stored here. If empty, debit/credit processing is not performed.</td>
</tr>
<tr>
<td>ESS_STR_T</td>
<td>offset</td>
<td>Optional: MDX tuple expression specifying the location for offset entries, if any, to be written.</td>
</tr>
<tr>
<td>ESS_STR_T</td>
<td>sourceRegion</td>
<td>MDX set expression indicating the database region referred to by the right-hand sides of the formulas in the script.</td>
</tr>
<tr>
<td>ESS_ULONG64_T</td>
<td>groupID</td>
<td>Internal use only. Always enter 0.</td>
</tr>
<tr>
<td>ESS_ULONG64_T</td>
<td>ruleID</td>
<td>Internal use only. Always enter 0.</td>
</tr>
</tbody>
</table>

## ESS_PROCSTATE_T

When you perform asynchronous operations (for example, a calculation), this structure is returned from calls to `EssGetProcessState()`. This lets the caller determine the status of the asynchronous operation.

**Note:** In this release of the C API, the `State` field is the only field implemented; all other fields are reserved for future use.

typedef struct ESS_PROCSTATE_T
{
    ESS_USHORT_T Action;
    ESS_USHORT_T State;
    ESS_USHORT_T Reserved1;
    ESS_ULONG_T  Reserved2;
    ESS_ULONG_T  Reserved3;
} ESS_PROCSTATE_T, *ESS_PPROCSTATE_T;

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_USHORT_T</td>
<td>Action</td>
<td>Current process action (not used)</td>
</tr>
</tbody>
</table>
### ESS_RATEINFO_T

This currency partition rate information structure is used by `EssGetCurrencyRateInfo()`. The fields in this structure cannot be modified by the API.

```c
typedef struct ESS_RATEINFO_T {
    ESS_MBRNAME_T MbrName;
    ESS_MBRNAME_T RateMbr[ESS_CRDB_MAXDIMNUM];
} ESS_RATEINFO_T, *ESS_PRATEINFO_T, **ESS_PPRATEINFO_T;
```

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_MBRNAME_T</td>
<td>MbrName</td>
<td>Member name</td>
</tr>
<tr>
<td>ESS_MBRNAME_T</td>
<td>RateMbr</td>
<td>Array of rate member names</td>
</tr>
</tbody>
</table>

### ESS_REQUESTINFO_T

Contains information that can be used to display information about, or terminate, sessions and requests. A session is the time between login and logout for a user connected to Essbase Server. A request is a query sent to Essbase by a user or by another process; for example, starting an application, or restructuring a database outline. Each session can process only one request at a time; therefore, sessions and requests have a one-to-one relationship.

```c
typedef struct ESS_REQUESTINFO_T {
    ESS_LOGINID_T LoginId;                user login identification tag
    ESS_USERNAME_T UserName;              user name
    ESS_SVRNAME_T LoginSourceMachine;     Login machine name
    ESS_APPNAME_T AppName;                connected application
    ESS_DBNAME_T DbName;                  connected database
    ESS_USHORT_T DbRequestCode;           Request code
    ESS_DESC_T RequestString;             Request string
    ESS_TIME_T TimeStarted;               time started (in seconds)
    ESS_REQ_STATE_T State;               current process state
} ESS_REQUESTINFO_T, *ESS_PREQUESTINFO_T, **ESS_PPREQUESTINFO_T;
```
<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_LOGINID_T</td>
<td>LoginId</td>
<td>A unique number assigned to the user when the user logs in.</td>
</tr>
<tr>
<td>ESS_USERNAME_T</td>
<td>UserName</td>
<td>The name of the requesting user.</td>
</tr>
<tr>
<td>ESS_SVRNAME_T</td>
<td>LoginSourceMachine</td>
<td>Server name from which the session or request is being made</td>
</tr>
<tr>
<td>ESS_APPNAME_T</td>
<td>AppName</td>
<td>The active application (if any) for the session or request</td>
</tr>
<tr>
<td>ESS_DBNAME_T</td>
<td>DbName</td>
<td>The active database (if any) for the session or request</td>
</tr>
<tr>
<td>ESS_USHORT_T</td>
<td>DbRequestCode</td>
<td>A positive integer representing an active session. Example: 774896669</td>
</tr>
<tr>
<td>ESS_DESC_T</td>
<td>RequestString</td>
<td>A string representing the type of request. For possible values, see Request Types below.</td>
</tr>
<tr>
<td>ESS_TIME_T</td>
<td>TimeStarted</td>
<td>how long the session or request has been in progress (in seconds)</td>
</tr>
</tbody>
</table>

Request Types
- Process xref request
- xref test
- Restructure
- GetCurrencyDb
- SetCurrencyType
- Export
- SQLImport
- SQLRetrieve
- Report
- SQLConnect
- SQLDatabases
- Calculate
- SetDefaultCalcScript
- ListCalcFunc
- VerifyFormula
- LoadAlias
- ListAliases
- DumpAlias
- BuildDimFile
- GetMbrInfo
- TestDriver
- GetSmStats
- OtlQueryMbrs
- OtlQueryAttrib
- CheckAttribute
- List location aliases
- ClearData
- SetCurrencyDb
- GetCurrencyType
- ParExport
- Import
- CancelUpdate
- SpreadsheetOperation
- SQLListDsn
- SQLTables
- ParseCalcScript
- GetDefaultCalcScript
- VerifyJavaSpec
- ListUdfs
- RemoveAlias
- SetAlias
- BuildDimStart
- GetDSInfo
- GetMbrCalc
- GetDimInfo
- PerfCommand
- OtlQueryMbrs
- OtlGetUpdateTime
- PutReplicatedCells
- Create location alias
- Validate
- GetStats
- SetCurrencyType
- GetCurrencyRate
- DataLoad
ESS_REQUESTINFOEX_T

Contains information that can be used to display information about, or terminate, sessions and requests. A session is the time between login and logout for a user connected to Essbase Server. A request is a query sent to Essbase by a user or by another process; for example, starting an application, or restructuring a database outline. Each session can process only one request at a time; therefore, sessions and requests have a one-to-one relationship. This structure is similar to ESS_REQUESTINFO_T, with the addition of the ProviderName and connparam fields.

typedef struct ESS_REQUESTINFOEX_T
{
    ESS_LOGINID_T         LoginId;
    ESS_USERNAME_T        UserName;
    ESS_USERNAME_T        ProviderName;
    ESS_CONNPARAM_T       connparam;
    ESS_SVRNAME_T         LoginSourceMachine;
    ESS_APPNAME_T         AppName;
    ESS_DBNAME_T          DbName;
    ESS_USHORT_T          DbRequestCode;
    ESS_DESC_T            RequestString;
}
```c

ESS_TIME_T    TimeStarted;
ESS_REQ_STATE_T State;
} ESS_REQUESTINFOEX_T, *ESS_PREQUESTINFOEX_T, **ESS_PPREQUESTINFOEX_T;
```

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_LOGINID_T</td>
<td>LoginId</td>
<td>A unique number assigned to the user when the user logs in</td>
</tr>
<tr>
<td>ESS_USERNAME_T</td>
<td>UserName</td>
<td>Name of the requesting user</td>
</tr>
<tr>
<td>ESS_USERNAME_T</td>
<td>ProviderName</td>
<td>Name of the user directory. Example: @Native Directory</td>
</tr>
<tr>
<td>ESS_CONNPARAM_T</td>
<td>connparam</td>
<td>Unique identity attribute identifying a user or group in a directory. Example: native://nvid=f0ed2a6d7fb07688:5a342200:1265973105c:-7f46?USER</td>
</tr>
<tr>
<td>ESS_SVRNAME_T</td>
<td>LoginSourceMachine</td>
<td>Server name from which the session or request is being made</td>
</tr>
<tr>
<td>ESS_APPNAME_T</td>
<td>AppName</td>
<td>Active application (if any) for the session or request</td>
</tr>
<tr>
<td>ESS_DBNAME_T</td>
<td>DbName</td>
<td>Active database (if any) for the session or request</td>
</tr>
<tr>
<td>ESS_USHORT_T</td>
<td>DbRequestCode</td>
<td>A positive integer representing an active session. Example: 774896669</td>
</tr>
<tr>
<td>ESS_DESC_T</td>
<td>RequestString</td>
<td>A string representing the type of request. For possible values, see &quot;ESS_REQUESTINFOEX_T&quot; on page 182.</td>
</tr>
<tr>
<td>ESS_TIME_T</td>
<td>TimeStarted</td>
<td>How long the session or request has been in progress (in seconds)</td>
</tr>
<tr>
<td>ESS_REQ_STATE_T</td>
<td>State;</td>
<td>State of the current session or request: whether it is processing, terminating, or terminated.</td>
</tr>
</tbody>
</table>

**Request Types**

- Process xref request
- xref test
- Restructure
- GetCurrencyDb
- SetCurrencyType
- Export
- SQLImport
- SQLRetrieve
- Report
- SQLConnect
- SQLDatabases
- Calculate
- SetDefaultCalcScript
- ListCalcFunc
- VerifyFormula
- LoadAlias
- ListAliases
- DumpAlias
- BuildDimFile
- GetMbrInfo
- TestDriver
- GetSmStats
- OtlQueryMbrs
- OtlQueryAttrib
- CheckAttribute
- List location aliases
- ClearData
- SetCurrencyDb
- GetCurrencyType
- ParExport
- Import
- CancelUpdate
- SpreadsheetOperation
- SQLListDsn
- SQLTables
- ParseCalcScript
- GetDefaultCalcScript
- VerifyJavaSpec
- ListUdfs
- RemoveAlias
- SetAlias
- BuildDimStart
- GetDSInfo
- GetMbrCalc
- GetDimInfo
- PerfCommand
- OtlQueryMbrs
- OtlGetUpdateTime
- PutReplicatedCells
Create location alias
Validate
GetStats
SetCurrencyType
GetCurrencyRate
DataLoad
StreamDataload
ClearUserLocks
SpreadsheetCellOperation
SQLColumns
SQLGetDsn
RunDefaultCalcScript
CalcStats
UpdateCdfCdm
UdfInfo
ClearAliases
GetAlias
BuildDimension
GetSelectedMbrInfo
CheckMbrName
GetAttributeNameSpecs
GetOtlInfo
OtlQueryUDAs
GetAttrInfo
GetReplicatedCells
Delete location alias

ESS_REQ_STATE_T

Used by ESS_REQUESTINFO_T. This structure returns information about the state of the current session or request. Fields in this structure cannot be modified using the API.

typedef ESS_USHORT_T ESS_REQ_STATE_T;
#define ESS_REQ_IN_PROGRESS 0
#define ESS_REQ_TERMINATING 1
#define ESS_REQ_TERMINATED 2
<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_USHORT_T</td>
<td>ESS_REQ_IN_PROGRESS (0)</td>
<td>The current session or request is processing.</td>
</tr>
<tr>
<td>ESS_USHORT_T</td>
<td>ESS_REQ_TERMINATING (1)</td>
<td>The current session or request is terminating.</td>
</tr>
<tr>
<td>ESS_USHORT_T</td>
<td>ESS_REQ_TERMINATED (2)</td>
<td>The current session or request is terminated.</td>
</tr>
</tbody>
</table>

**ESS_SECURITY_MODE_T**

Used by `EssGetEssbaseSecurityMode`. This data type returns information about the security mode of Essbase Server.

```c
typedef ESS_USHORT_T, ESS_SECURITY_MODE_T;
#define ESS_NATIVE_SECURITY             1
#define ESS_SS_SECURITY         2
```

**ESS_SEQID_T**

Contains an array of sequence ids.

```c
typedef struct ESS_SEQID_T
{
    ESS_ULONG_T, seq_id_start;
    ESS_ULONG_T, seq_id_upper_start;
    ESS_ULONG_T, seq_id_end;
    ESS_ULONG_T, seq_id_upper_end;
} ESS_SEQID_T;
```

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_ULONG_T</td>
<td>seq_id_start</td>
<td>Start of range</td>
</tr>
<tr>
<td>ESS_ULONG_T</td>
<td>seq_id_upper_start</td>
<td>Upper start of range</td>
</tr>
<tr>
<td>ESS_ULONG_T</td>
<td>seq_id_end</td>
<td>End of range</td>
</tr>
<tr>
<td>ESS_ULONG_T</td>
<td>seq_id_upper_end</td>
<td>Upper end of range</td>
</tr>
</tbody>
</table>

**ESS_TIMERECORD_T**

```c
typedef struct ESS_TIMERECORD_T
{
    ESS_TIME_T   TimeValue;
    ESS_USHORT_T Seconds;
    ESS_USHORT_T Minutes;
    ESS_USHORT_T Hours;
    ESS_USHORT_T Day;
    ESS_USHORT_T Month;
    ESS_USHORT_T Year;
    ESS_USHORT_T Weekday;
} ESS_TIMERECORD_T, *ESS_PTIMERECORD_T;
```
Used in the “ESS_DBRQINFO_T” on page 126 structure. The times expressed in this structure are usually server times. The fields are:

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_TIME_T</td>
<td>TimeValue</td>
<td>Time value in seconds after 1/1/70</td>
</tr>
<tr>
<td>ESS_USHORT_T</td>
<td>Seconds</td>
<td>Seconds after the minute. Values: 0-59.</td>
</tr>
<tr>
<td>ESS_USHORT_T</td>
<td>Minutes</td>
<td>Minutes after the hour. Values: 0-59.</td>
</tr>
<tr>
<td>ESS_USHORT_T</td>
<td>Hours</td>
<td>Hours since midnight. Values: 0-23.</td>
</tr>
<tr>
<td>ESS_USHORT_T</td>
<td>Year</td>
<td>Years since 1900.</td>
</tr>
<tr>
<td>ESS_USHORT_T</td>
<td>Weekday</td>
<td>Days since Sunday. Values: 0-6. Sunday = 0.</td>
</tr>
</tbody>
</table>

**ESS_TRANSACTION_ENTRY_T**

Contains

```c
typedef struct ess_transaction_entry_t
{
    ESS_ULONG_T seq_id;
    ESS_ULONG_T seq_id_upper;
    ESS_TIME_T time_start;
    ESS_TIME_T time_end;
    ESS_USERNAME_T username;
    ESS_UCHAR_T type;
    ESS_UCHAR_T state;
    ESS_CHAR_T reserved1;
    ESS_TRANSACTION_REQSPECIFIC_T reqSpecDat;
} ESS_TRANSACTION_ENTRY_T
```

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_ULONG_T</td>
<td>seq_id</td>
<td>Sequence ID</td>
</tr>
<tr>
<td>ESS_ULONG_T</td>
<td>seq_id_upper</td>
<td>Sequence ID upper for future</td>
</tr>
<tr>
<td>ESS_TIME_T</td>
<td>time_start</td>
<td>Operation start time</td>
</tr>
<tr>
<td>ESS_TIME_T</td>
<td>time_end</td>
<td>Operation end time</td>
</tr>
<tr>
<td>ESS_USERNAME_T</td>
<td>username</td>
<td>Executing user</td>
</tr>
<tr>
<td>ESS_UCHAR_T</td>
<td>type</td>
<td>Record type</td>
</tr>
<tr>
<td>ESS_UCHAR_T</td>
<td>state</td>
<td>Do not use this field and client</td>
</tr>
<tr>
<td>ESS_CHAR_T</td>
<td>reserved1</td>
<td>For future expansion</td>
</tr>
<tr>
<td>Data Type</td>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>-----------</td>
<td>------------------------------------</td>
</tr>
<tr>
<td>ESS_TRANSACTION_REQSPECIFIC_T</td>
<td>reqSpecDat</td>
<td>Request specific data</td>
</tr>
</tbody>
</table>

**ESS_TRANSACTION_REPLAY_INP_T**

Contains information on transaction replays.

typedef struct ESS_TRANSACTION_REPLAY_INP_T {
    ESS_UCHAR_T, InpType;
    ESS_UCHAR_T, reserved1;
    ESS_UCHAR_T, reserved2;
    ESS_UCHAR_T, reserved3;
    union
    {
        ESS_TIME32_T, InpTime, value;
        ESS_ULONG_T, num_seq_id_range, value;
    } value;
} ESS_TRANSACTION_REPLAY_INP_T;

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_UCHAR_T</td>
<td>InpType</td>
<td>is it time based or sequence id</td>
</tr>
<tr>
<td>ESS_UCHAR_T</td>
<td>reserved1</td>
<td>reserved</td>
</tr>
<tr>
<td>ESS_UCHAR_T</td>
<td>reserved2</td>
<td>reserved</td>
</tr>
<tr>
<td>ESS_UCHAR_T</td>
<td>reserved3</td>
<td>reserved</td>
</tr>
<tr>
<td>ESS_TIME32_T</td>
<td>InpTime</td>
<td>A union variable for the following values:</td>
</tr>
<tr>
<td>ESS_ULONG_T</td>
<td>num_seq_id_range</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**ESS_TRANSACTION_REQSPECIFIC_T**

Contains information.

typedef struct ess_transaction_reqspecific_t {
    ESS_UCHAR_T, ucReqType;
    ESS_UCHAR_T, reserved1;
    ESS_UCHAR_T, reserved2;
    ESS_UCHAR_T, reserved3;
    union
    {
        ESS_FILENAME_T, calcname, value;
        ESS_LOG_DATALOAD_T, dataload_info, value;
        ESS_LOG_DIMBLD_T, dimbld_info, value;
        ESS_FILENAME_T, tmpotlfilename, value;
    } value;
} ESS_TRANSACTION_REQSPECIFIC_T;

188  C Main API Declarations
### ESS_UCHAR_T, ESS_FILENAME_T

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_UCHAR_T</td>
<td>ucReqType</td>
<td>Request type</td>
</tr>
<tr>
<td>ESS_UCHAR_T</td>
<td>reserved1</td>
<td>Reserved</td>
</tr>
<tr>
<td>ESS_UCHAR_T</td>
<td>reserved2</td>
<td>Reserved</td>
</tr>
<tr>
<td>ESS_UCHAR_T</td>
<td>reserved3</td>
<td>Reserved</td>
</tr>
<tr>
<td>ESS_FILENAME_T</td>
<td>calcname</td>
<td>A union variable for the following values:</td>
</tr>
<tr>
<td>ESS_LOG_DATLOAD_T</td>
<td>dataload_info</td>
<td>• Calc file name</td>
</tr>
<tr>
<td>ESS_LOG_DIMBLD_T</td>
<td>dimbld_info</td>
<td>• Data load details</td>
</tr>
<tr>
<td>ESS_FILENAME_T</td>
<td>tmpotlfilename</td>
<td>• Build load details</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Temporary outline file name</td>
</tr>
</tbody>
</table>

### ESS_USERAPP_T, ESS_GROUPAPP_T

Contains access privilege information for a user or group and a specific application. The Access field is the only field in this structure that can be modified using the API. The fields are:

```c
typedef struct ESS_USERAPP_T {
    ESS_USERNAME_T  UserName;
    ESS_APPNAME_T   AppName;
    ESS_ACCESS_T    Access;
    ESS_ACCESS_T    MaxAccess;
} ESS_USERAPP_T, *ESS_PUSERAPP_T, **ESS_PPUSERAPP_T,
ESS_GROUPAPP_T, *ESS_PGROUPAPP_T, **ESS_PPGROUPAPP_T;
```

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_USERNAME_T</td>
<td>UserName</td>
<td>The user or group name</td>
</tr>
<tr>
<td>ESS_APPNAME_T</td>
<td>AppName</td>
<td>The application name</td>
</tr>
<tr>
<td>ESS_ACCESS_T</td>
<td>Access</td>
<td>The assigned access privilege to the application for the user or group. This field can take any combination of the following bit values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESS_PRIV_NONE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESS_PRIV_APPLOAD</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESS_PRIV_APPDESIGN</td>
</tr>
<tr>
<td>ESS_ACCESS_T</td>
<td>MaxAccess</td>
<td>The maximum access privilege to the application for the user or group from all sources</td>
</tr>
</tbody>
</table>

### ESS_USERAPPEX_T, ESS_GROUPAPPEX_T

Contains access privilege information for a user or group and a specific application. This structure is similar to ESS_USERAPP_T, ESS_GROUPAPP_T, with the addition of the ProviderName, Type, and connparam fields.
typedef struct ESS_USERAPPEX_T
{
    ESS_USERNAME_T  UserName;
    ESS_USERNAME_T  ProviderName;
    ESS_CONNPARAM_T connparam;
    ESS_USHORT_T    Type;
    ESS_APPNAME_T   AppName;
    ESS_ACCESS_T    Access;
    ESS_ACCESS_T    MaxAccess;
} ESS_USERAPPEX_T, *ESS_PUSERAPPEX_T, **ESS_PPUSERAPPEX_T;

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_USERNAME_T</td>
<td>UserName</td>
<td>The user or group name</td>
</tr>
<tr>
<td>ESS_USERNAME_T</td>
<td>ProviderName</td>
<td>Name of the user directory. Example: @Native Directory</td>
</tr>
<tr>
<td>ESS_CONNPARAM_T</td>
<td>connparam</td>
<td>Unique identity attribute identifying user or group in a directory. Example: native://nvid=f0ed2a67fb07688:5a342200:1265973105c:-7f46?USER</td>
</tr>
<tr>
<td>ESS_USHORT_T</td>
<td>Type</td>
<td>Type of the structure. This field can contain the following values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESS_TYPE_USER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESS_TYPE_GROUP</td>
</tr>
<tr>
<td>ESS_APPNAME_T</td>
<td>AppName</td>
<td>The application name</td>
</tr>
<tr>
<td>ESS_ACCESS_T</td>
<td>Access</td>
<td>The assigned access privilege to the application for the user or group. This field can take any combination of the following bit values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESS_PRIV_NONE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESS_PRIV_APPLOAD</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESS_PRIV_APPDESIGN</td>
</tr>
<tr>
<td>ESS_ACCESS_T</td>
<td>MaxAccess</td>
<td>The maximum access privilege to the application for the user or group from all sources</td>
</tr>
</tbody>
</table>

**ESS_USERDB_T, ESS_GROUPDB_T**

Contains access privilege information for a user or group and a specific database. The Access and Filter fields are the only fields in this structure that can be modified using the API. The fields are:

typedef struct ESS_USERDB_T
{
    ESS_USERNAME_T  UserName;
    ESS_APPNAME_T   AppName;
    ESS_DBNAME_T    DbName;
    ESS_ACCESS_T    Access;
    ESS_ACCESS_T    MaxAccess;
    ESS_FTRNAME_T   FilterName;
} ESS_USERDB_T, *ESS_PUSERDB_T, **ESS_PPUSERDB_T;

ESS_GROUPDB_T, *ESS_PGROUPDB_T, **ESS_PPGROUPDB_T;
### Data Type | Field | Description
---|---|---
ESS_USERNAME_T | UserName | The user or group name.
ESS_APPNAME_T |AppName | The application name.
ESS_DBNAME_T |DbName | The database name.
ESS_ACCESS_T | Access | The assigned access privilege to the database for the user or group. Access privileges are set through the Administrative Services interface. This field can take any combination of the following bit values:
  - ESS_PRIV_NONE
  - ESS_PRIV_READ
  - ESS_PRIV_WRITE
  - ESS_PRIV_CALC
  - ESS_PRIV_DBLOAD
  - ESS_PRIV_DBDESIGN

These values are a subset of the “Bitmask Data Types (C)” on page 96.

ESS_ACCESS_T | MaxAccess | The maximum access privilege to the database for the user or group from all sources. Access privileges are set through the Administrative Services interface.

ESS_FTRNAME_T | FilterName | The name of the assigned database filter, if any. If none, the first byte is NULL.

### ESS_USERDBEX_T, ESS_GROUPDBEX_T
Contains access privilege information for a user or group and a specific database. This structure is similar to ESS_USERDB_T, ESS_GROUPDB_T, with the addition of the ProviderName, connparam and Type fields.

typedef struct ESS_USERDBEX_T
{
    ESS_USERNAME_T  UserName;
    ESS_USERNAME_T  ProviderName;
    ESS_CONNPARAM_T connparam;
    ESS_USHORT_T    Type;
    ESS_APPNAME_T   AppName;
    ESS_DBNAME_T    DbName;
    ESS_ACCESS_T    Access;
    ESS_ACCESS_T    MaxAccess;
    ESS_FTRNAME_T   FilterName;
} ESS_USERDBEX_T, *ESS_PUSERDBEX_T, **ESS_PPUSERDBEX_T,

ESS_GROUPDBEX_T, *ESS_PGROUPDBEX_T, **ESS_PPGROUPDBEX_T;

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_USERNAME_T</td>
<td>UserName</td>
<td>The user or group name</td>
</tr>
<tr>
<td>ESS_USERNAME_T</td>
<td>ProviderName</td>
<td>Name of the user directory. Example: @Native Directory</td>
</tr>
<tr>
<td>Data Type</td>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ESS_CONNPARAM_T</td>
<td>connparam</td>
<td>Unique identity attribute identifying user or group in a directory. Example: native://nvid=f0ed2a6d7fb07688:5a342200:1265973105c:-7f46?USER</td>
</tr>
<tr>
<td>ESS_USHORT_T</td>
<td>Type</td>
<td>Type of the structure. This field can contain the following value:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESS_TYPE_USER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESS_TYPE_GROUP</td>
</tr>
<tr>
<td>ESS_APPNAME_T</td>
<td>AppName</td>
<td>The application name</td>
</tr>
<tr>
<td>ESS_DBNAME_T</td>
<td>DbName</td>
<td>The database name</td>
</tr>
<tr>
<td>ESS_ACCESS_T</td>
<td>Access</td>
<td>The assigned access privilege to the database for the user or group. Access privileges are set through the Administrative Services interface. This field can take any combination of the following bit values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESS_PRIV_NONE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESS_PRIV_READ</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESS_PRIV_WRITE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESS_PRIV_CALC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESS_PRIV_DBLOAD</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESS_PRIV_DBDESIGN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>These values are a subset of the &quot;Bitmask Data Types (C)&quot; on page 96.</td>
</tr>
<tr>
<td>ESS_ACCESS_T</td>
<td>MaxAccess</td>
<td>The maximum access privilege to the database for the user or group from all sources.</td>
</tr>
<tr>
<td>ESS_FTRNAME_T</td>
<td>FilterName</td>
<td>The name of the assigned database filter, if any. If none, the first byte is NULL.</td>
</tr>
</tbody>
</table>

**ESS_USERINFO_T, ESS_GROUPINFO_T**

Stores information about a user or group. Some of the fields are specific to users and cannot be used for groups. The Access, Expiration, and PwdChgNow fields are the only fields in this structure that can be modified using the API. The fields are:

**Note:** Refer also to the locale-specific extended User Info structure, “ESS_USERINFOEX_T” on page 195.

```c
typedef struct ESS_USERINFO_T
{
    /* The items below are 4.X and above */
    ESS_USERNAME_T    Name;
    ESS_APPNAME_T     AppName;
    ESS_DBNAME_T      DbName;
    ESS_BOOL_T        Login;
    ESS_USHORT_T      Type;
    ESS_ACCESS_T      Access;
    ESS_ACCESS_T      MaxAccess;
}...```
ESS_DATE_T      Expiration;
ESS_TIME_T      LastLogin;
ESS_TIME_T      DbConnectTime;
ESS_USHORT_T    FailCount;
ESS_LOGINID_T   LoginId;

/* The items below are 5.X and above */
ESS_DESC_T      Description;
ESS_EMAIL_T     EMailID;
ESS_BOOL_T      LockedOut;
ESS_BOOL_T      PwdChgNow;

} ESS_USERINFO_T, *ESS_USERINFO_T, **ESS_PUSERINFO_T,
ESS_GROUPINFO_T, *ESS_PGROUPINFO_T, **ESS_PPGROUPINFO_T;

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_USERNAME_T</td>
<td>Name</td>
<td>User or group name</td>
</tr>
<tr>
<td>ESS_APPNAME_T</td>
<td>AppName</td>
<td>Name of currently connected application (if applicable)</td>
</tr>
<tr>
<td>ESS_DBNAME_T</td>
<td>DbName</td>
<td>Name of the currently connected database (if applicable)</td>
</tr>
<tr>
<td>ESS_BOOL_T</td>
<td>Login</td>
<td>Flag to indicate login status (users only)</td>
</tr>
</tbody>
</table>
| ESS_USHORT_T   |Type             | Type of the structure (user or group). This field can contain the following values:
|                |                 | - ESS_TYPE_USER
|                |                 | - ESS_TYPE_GROUP
| ESS_ACCESS_T   |Access           | User or group assigned default access privileges. Values: any combination of the following bit values:
|                |                 | - ESS_ACCESS_SUPER /* Supervisor, all bits set */
|                |                 | - ESS_PRIV_APPCREATE /* App create/delete privilege */
|                |                 | - ESS_PRIV_USERCREATE /* user create/delete privilege */
| ESS_ACCESS_T   |MaxAccess        | User's maximum access privileges (users only, including individual access and access levels due to group membership. |
| ESS_DATE_T     |Expiration       | User's password expiration date.                                            |
| ESS_TIME_T     |LastLogin        | Date of user's last successful login stated as Greenwich Mean Time (users only). |
| ESS_TIME_T     |DbConnectTime    | Local (server) time of database connection. Read-only. Cannot be set by EssSetUser. |
| ESS_USHORT_T   |FailCount        | Count of the failed login attempts since the last successful login (users only). |
| ESS_LOGINID_T  |LoginId          | The user login identification tag (users only).                             |
| ESS_DESC_T     |Description      | User/group description                                                      |
| ESS_EMAIL_T    |EMailID          | User/group email address                                                    |
| ESS_BOOL_T     |LockedOut        | Flag that user is locked out.                                               |
| ESS_BOOL_T     |PwdChgNow        | Flag that user must change password.                                        |
ESS_USERINFOID_T, ESS_GROUPINFOID_T

Stores information about a user or group. This structure is similar to ESS_USERINFOEX_T, with the addition of the ProviderName and connparam fields.

typedef struct ESS_USERINFOID_T
{
    ESS_USERNAME_T     Name;
    ESS_USERNAME_T     ProviderName;
    ESS_PASSWORD_T     Password;
    ESS_APPNAME_T      AppName;
    ESS_DBNAME_T       DbName;
    ESS_BOOL_T         Login;
    ESS_Ushort_T       Type;
    ESS_ACCESS_T       Access;
    ESS_ACCESS_T       MaxAccess;
    ESS_DATE_T         Expiration;
    ESS_TIME_T         LastLogin;
    ESS_TIME_T         DbConnectTime;
    ESS_Ushort_T       FailCount;
    ESS_LOGINID_T      LoginId;
    ESS_DESC_T         Description;
    ESS_EMAIL_T        EMailID;
    ESS_BOOL_T         LockedOut;
    ESS_BOOL_T         PwdChgNow;
    ESS_PROTOCOL_T     protocol;
    ESS_CONNPARAM_T    connparam;
} ESS_USERINFOID_T, *ESS_PUSERINFOID_T, **ESS_PPUSERINFOID_T,
ESS_GROUPINFOID_T, *ESS_PGROUPINFOID_T, **ESS_PPGROUPINFOID_T;

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_USERNAME_T</td>
<td>Name</td>
<td>User name</td>
</tr>
<tr>
<td>ESS_USERNAME_T</td>
<td>ProviderName</td>
<td>Name of the user directory. Example: @Native Directory</td>
</tr>
<tr>
<td>ESS_PASSWORD_T</td>
<td>Password</td>
<td>Password of externally authenticated user. This is used only when setting an externally authenticated user to the Essbase authenticated mechanisms. This password is ignored in other situations, including retrieving information from the server on the externally authenticated user.</td>
</tr>
<tr>
<td>ESS_APPNAME_T</td>
<td>AppName</td>
<td>Name of the currently connected application (if applicable)</td>
</tr>
<tr>
<td>ESS_DBNAME_T</td>
<td>DbName</td>
<td>Name of the currently connected database (if applicable)</td>
</tr>
<tr>
<td>ESS_BOOL_T</td>
<td>Login</td>
<td>Flag to indicate login status</td>
</tr>
<tr>
<td>ESS_Ushort_T</td>
<td>Type</td>
<td>Type of the structure. This field can contain the following value:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESS_TYPE_USER</td>
</tr>
<tr>
<td>ESS_ACCESS_T</td>
<td>Access</td>
<td>User assigned default access privileges. Values can be any combination of the following bit values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESS_ACCESS_SUPER /* Supervisor, all bits set */</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESS_PRIV_APPCREATE /* App create/delete privilege */</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESS_PRIV_USERCREATE /* user create/delete privilege */</td>
</tr>
<tr>
<td>Data Type</td>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ESS_ACCESS_T</td>
<td>MaxAccess</td>
<td>User's maximum access privileges (including individual access and access levels due to group membership)</td>
</tr>
<tr>
<td>ESS_DATE_T</td>
<td>Expiration</td>
<td>User's password expiration date</td>
</tr>
<tr>
<td>ESS_TIME_T</td>
<td>LastLogin</td>
<td>Date of user's last successful login stated as Greenwich Mean Time</td>
</tr>
<tr>
<td>ESS_TIME_T</td>
<td>DbConnectTime</td>
<td>Local (server) time of database connection. Read-only. Cannot be set by EssSetUser.</td>
</tr>
<tr>
<td>ESS_USHORT_T</td>
<td>FailCount</td>
<td>Count of the failed login attempts since the last successful login</td>
</tr>
<tr>
<td>ESS_LOGINID_T</td>
<td>LoginId</td>
<td>User login identification tag</td>
</tr>
<tr>
<td>ESS_DESC_T</td>
<td>Description</td>
<td>User description</td>
</tr>
<tr>
<td>ESS_EMAIL_T</td>
<td>EMailID</td>
<td>User email address</td>
</tr>
<tr>
<td>ESS_BOOL_T</td>
<td>LockedOut</td>
<td>Flag indicating that the user is locked out</td>
</tr>
<tr>
<td>ESS_BOOL_T</td>
<td>PwdChgNow</td>
<td>Flag indicating that the user must change the password</td>
</tr>
<tr>
<td>ESS_PROTOCOL_T</td>
<td>protocol</td>
<td>External authentication protocol.</td>
</tr>
<tr>
<td>ESS_CONNPARAM_T</td>
<td>connparam</td>
<td>Unique identity attribute identifying a user or group in a directory. Example: native://nvid=f0ed2a6d7fb07688:5a342200:1265973105c:-7f46?USER</td>
</tr>
</tbody>
</table>

**ESS_USERINFOEX_T**

Stores information about a user or group. Some of the fields are specific to users and cannot be used for groups. The Access, Expiration, and PwdChgNow fields are the only fields in this structure that can be modified using the API.

This extended User Info structure is slightly different from the standard ESS_USERINFO_T structure used by EssGetUser (see “ESS_USERINFO_T, ESS_GROUPINFO_T” on page 192). This extended structure is used by EssGetUserEx.

The fields are:

```
typedef struct ESS_USERINFOEX_T
{
    ESS_USERNAME_T     Name;
    ESS_PASSWORD_T     Password;
    ESS_APPNAME_T      AppName;
    ESS_DBNAME_T       DbName;
    ESS_BOOL_T         Login;
    ESS_USHORT_T       Type;
    ESS_ACCESS_T       Access;
    ESS_ACCESS_T       MaxAccess;
    ESS_DATE_T         Expiration;
    ESS_TIME_T         LastLogin;
    ESS_TIME_T         DbConnectTime;
    ESS_USHORT_T       FailCount;
} USERINFOEX;
```
<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_USERNAME_T</td>
<td>Name, Externally authenticated User name.</td>
</tr>
<tr>
<td>ESS_PASSWORD_T</td>
<td>Password, Password of externally authenticated user.</td>
</tr>
<tr>
<td>ESS_APPNAME_T</td>
<td>AppName, Name of currently connected application (if applicable).</td>
</tr>
<tr>
<td>ESS_DBNAME_T</td>
<td>DbName, Name of the currently connected database (if applicable).</td>
</tr>
<tr>
<td>ESS_BOOL_T</td>
<td>Login, Flag to indicate login status (users only).</td>
</tr>
<tr>
<td>ESS_USHORT_T</td>
<td>Type, Type of the structure (user or group).</td>
</tr>
<tr>
<td>ESS_ACCESS_T</td>
<td>Access, User or group assigned default access privileges.</td>
</tr>
<tr>
<td>ESS_MAXACCESS_T</td>
<td>MaxAccess, User's maximum access privileges (users only).</td>
</tr>
<tr>
<td>ESS_DATE_T</td>
<td>Expiration, User's password expiration date.</td>
</tr>
<tr>
<td>ESS_TIME_T</td>
<td>LastLogin, Date of user's last successful login stated as Greenwich Mean Time</td>
</tr>
<tr>
<td>ESS_TIME_T</td>
<td>DbConnectTime, Local (server) time of database connection. Read-only.</td>
</tr>
<tr>
<td>ESS_USHORT_T</td>
<td>FailCount, Count of the failed login attempts since the last successful login</td>
</tr>
<tr>
<td>ESS_LOGINID_T</td>
<td>LoginId, The user login identification tag (users only).</td>
</tr>
<tr>
<td>ESS_DESC_T</td>
<td>Description, User/group description.</td>
</tr>
<tr>
<td>ESS_EMAIL_T</td>
<td>EMailID, User/group email address.</td>
</tr>
<tr>
<td>ESS_BOOL_T</td>
<td>LockedOut, Flag that user is locked out.</td>
</tr>
<tr>
<td>ESS_BOOL_T</td>
<td>PwdChgNow, Flag that user must change password.</td>
</tr>
<tr>
<td>ESS_PROTOCOL_T</td>
<td>protocol, External authentication protocol.</td>
</tr>
</tbody>
</table>
### ESS_VARIABLE_T

**ESS_VARIABLE_T** is the primary substitution variable datatype. It identifies the substitution variable's value and name, as well as the Essbase database, application, and server where the variable is defined.

The Server name is optional, but recommended. If not included, the current server is the default. TheAppName is optional. TheDbName is optional, but if it exists, then theAppName member is required. TheVarName is required. TheVarValue is required.

```c
typedef struct ESS_VARIABLE_T {
    ESS_SVRNAME_T  Server;
    ESS_APPNAME_T  AppName;
    ESS_DBNAME_T   DbName;
    ESS_MBRNAME_T  VarName;
    ESS_CHAR_T     VarValue[ESS_VARVALUELEN];
} ESS_VARIABLE_T, *ESS_PVARIABLE_T, **ESS_PPVARIABLE_T;
```

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_SVRNAME_T</td>
<td>Server</td>
<td>Name of server where variable is defined (optional)</td>
</tr>
<tr>
<td>ESS_APPNAME_T</td>
<td>AppName</td>
<td>Name of application to restrict variable to</td>
</tr>
<tr>
<td>ESS_DBNAME_T</td>
<td>DbName</td>
<td>Name of database to restrict variable to. If used, it requires that application be set.</td>
</tr>
<tr>
<td>ESS_MBRNAME_T</td>
<td>VarName</td>
<td>Name of substitution variable.</td>
</tr>
<tr>
<td>ESS_CHAR_T</td>
<td>VarValue[256]</td>
<td>Value of substitution variable.</td>
</tr>
</tbody>
</table>
C Main API Functions

In This Chapter

C Main API Function Categories ................................................................. 199
C Main API Function Reference ................................................................. 215

C Main API Function Categories

Subtopics

- C Main API Alias Table Functions
- C Main API Application Functions
- C Main API Attributes Functions
- C Main API Database Functions
- C Main API Database Member Functions
- C Main API Drill-through Functions
- C Main API File Functions
- C Main API Group Administration Functions
- C Main API Initialization and Login Functions
- C Main API LRO Functions
- C Main API Location Aliases Functions
- C Main API Memory Allocation Functions
- C Main API Miscellaneous Functions
- C Main API Object Functions
- C Main API Partition Functions
- C Main API Performance Stats Functions
- C Main API Reporting, Updating, and Calculation Functions
- C Main API Security Filter Functions
- C Main API Substitution Variables Functions
- C Main API User Administration Functions
- C Main API User and Group Identity Functions
- C Main API Shared Services Functions
- C Main API Unicode Mode Functions

C Main API Alias Table Functions

Alias table functions manage database alias tables.
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EssListAliases</td>
<td>Lists all the alias tables in the active database.</td>
</tr>
<tr>
<td>EssLoadAlias</td>
<td>Loads an alias table for the active database from a structured text file</td>
</tr>
<tr>
<td>EssGetAlias</td>
<td>Gets the active alias table name from the active database for a user.</td>
</tr>
<tr>
<td>EssSetAlias</td>
<td>Sets the active alias table in the active database for a user.</td>
</tr>
<tr>
<td>EssDisplayAlias</td>
<td>Dumps contents of alias table in active database.</td>
</tr>
<tr>
<td>EssRemoveAlias</td>
<td>Removes an alias table from the active database.</td>
</tr>
<tr>
<td>EssClearAliases</td>
<td>Clears all alias tables for the active database.</td>
</tr>
</tbody>
</table>

**C Main API Application Functions**

The application functions can create new applications, and modify, copy, get information about and otherwise manage existing applications.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EssGetActive</td>
<td>Gets the names of the caller's current active application and database.</td>
</tr>
<tr>
<td>EssSetActive</td>
<td>Sets the callers active application and database.</td>
</tr>
<tr>
<td>EssClearActive</td>
<td>Clears the user's current active application and database.</td>
</tr>
<tr>
<td>EssListApplications</td>
<td>Lists all applications which are accessible to the caller.</td>
</tr>
<tr>
<td>EssConvertApplicationtoUnicode</td>
<td>Converts a non Unicode mode application to a Unicode mode application.</td>
</tr>
<tr>
<td>EssCreateApplication</td>
<td>Creates a new application, either on the client or the server.</td>
</tr>
<tr>
<td>EssCreateApplicationEx</td>
<td>Creates a new application with the option of setting the application type: Unicode- or non-Unicode mode.</td>
</tr>
<tr>
<td>EssCreateStorageTypedApplicationEx</td>
<td>Creates a new application with options for setting the data storage mode (block or aggregate) and application type (Unicode- or non-Unicode mode).</td>
</tr>
<tr>
<td>EssDeleteApplication</td>
<td>Deletes an existing application, either on the client or the server.</td>
</tr>
<tr>
<td>EssRenameApplication</td>
<td>Renames an existing application, either on the client or the server.</td>
</tr>
<tr>
<td>EssCopyApplication</td>
<td>Copies an existing application, either on the client or the server, to a new application, including all associated databases and objects.</td>
</tr>
<tr>
<td>EssGetApplicationInfoEx</td>
<td>Gets information from one or more applications</td>
</tr>
<tr>
<td>EssGetApplicationState</td>
<td>Gets an application state structure, which contains user-configurable parameters for the application.</td>
</tr>
<tr>
<td>EssSetApplicationState</td>
<td>Sets user-configurable parameters for the application using the application's state structure.</td>
</tr>
</tbody>
</table>
### C Main API Attributes Functions

These C Main functions are for attributes.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EssCheckAttributes</td>
<td>Returns the attribute type for given attribute dimensions, base dimensions, attribute members, and base members</td>
</tr>
<tr>
<td>EssFreeStructure</td>
<td>Frees memory dynamically allocated for string type attribute information</td>
</tr>
<tr>
<td>EssGetAssociatedAttributesInfo</td>
<td>Returns the attribute members associated with a given base member</td>
</tr>
<tr>
<td>EssGetAttributeInfo</td>
<td>Returns attribute information for a given attribute member or dimension</td>
</tr>
<tr>
<td>EssGetAttributeSpecifications</td>
<td>Retrieves attribute specifications for the outline</td>
</tr>
</tbody>
</table>

See C Outline API “C Outline API Attributes Functions” on page 714.

### C Main API Database Functions

Database functions carry out database management tasks, and retrieve and modify database information structures.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EssBeginDataLoad</td>
<td>Starts sending an update specification to the active database.</td>
</tr>
<tr>
<td>EssBeginDataLoadASO</td>
<td>Starts a data load on an aggregate storage database.</td>
</tr>
<tr>
<td>EssClearDatabase</td>
<td>Clears all loaded data in the active database.</td>
</tr>
<tr>
<td>EssCommitDatabase</td>
<td>Forces all data blocks in the active database to be written to disk.</td>
</tr>
<tr>
<td>EssCopyDatabase</td>
<td>Copies an existing database, either on the client or the server, to a new database, including all associated databases and objects.</td>
</tr>
<tr>
<td>EssCreateDatabase</td>
<td>Creates a new database within an application, on client or server.</td>
</tr>
<tr>
<td>EssDeleteDatabase</td>
<td>Deletes an existing database from an application, on client or server.</td>
</tr>
<tr>
<td>EssEndDataLoad</td>
<td>Marks the end of an update specification being sent to the active database.</td>
</tr>
</tbody>
</table>
### Function Description

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>EssGetCurrencyRateInfo</code></td>
<td>Gets a list of structures containing rate information for all members of the tagged currency partition dimension in the active database outline.</td>
</tr>
<tr>
<td><code>EssGetDatabaseInfo</code></td>
<td>Gets a database's information structure, which contains non user-configurable parameters for the database.</td>
</tr>
<tr>
<td><code>EssGetDatabaseInfoEx</code></td>
<td>Gets information for one or more databases.</td>
</tr>
<tr>
<td><code>EssGetDatabaseNote</code></td>
<td>Gets a database's note-of-the-day message.</td>
</tr>
<tr>
<td><code>EssGetDatabaseState</code></td>
<td>Gets a database's state structure, which contains user-configurable parameters for the database.</td>
</tr>
<tr>
<td><code>EssGetDatabaseStats</code></td>
<td>Gets the active database's Stats structure, which contains statistical information about the database.</td>
</tr>
<tr>
<td><code>EssListCurrencyDatabases</code></td>
<td>Lists all currency databases within a specific application that are accessible to the caller.</td>
</tr>
<tr>
<td><code>EssListDatabases</code></td>
<td>Lists all databases that are accessible to the caller, either within a specific application, or on an entire server.</td>
</tr>
<tr>
<td><code>EssListExistingLoadBuffers</code></td>
<td>Returns the list of structures that describe existing data load buffers for an aggregate storage database.</td>
</tr>
<tr>
<td><code>EssLoadBufferInit</code></td>
<td>Creates a temporary data load buffer.</td>
</tr>
<tr>
<td><code>EssLoadBufferTerm</code></td>
<td>Destroys the temporary data-load memory buffer(s) allocated by oadBufferInit.</td>
</tr>
<tr>
<td><code>EssLoadDatabase</code></td>
<td>Starts a database.</td>
</tr>
<tr>
<td><code>EssMergeDatabaseData</code></td>
<td>Merges two or more data slices into a single data slice.</td>
</tr>
<tr>
<td><code>EssRenameDatabase</code></td>
<td>Renames a database on client or server.</td>
</tr>
<tr>
<td><code>EssSetDatabaseNote</code></td>
<td>Sets a database's note-of-the-day message.</td>
</tr>
<tr>
<td><code>EssSetDatabaseState</code></td>
<td>Sets user-configurable parameters for the database using the database's state structure.</td>
</tr>
<tr>
<td><code>EssUnloadDatabase</code></td>
<td>Stops a database within an application on the server.</td>
</tr>
<tr>
<td><code>EssValidateDB</code></td>
<td>Checks the database for data integrity.</td>
</tr>
</tbody>
</table>

---

**C Main API Database Member Functions**

These functions obtain information about database members and build database dimensions.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>EssQueryDatabaseMembers</code></td>
<td>Performs a report-style query to list a selection of database member information.</td>
</tr>
<tr>
<td><code>EssCheckMemberName</code></td>
<td>Checks if a string is a valid member name within the active database outline.</td>
</tr>
<tr>
<td><code>EssGetMemberInfo</code></td>
<td>Gets a structure containing information about a specific member in the active database outline.</td>
</tr>
</tbody>
</table>
### C Main API Drill-through Functions

The following Drill-through functions manage drill-through URLs for drilling through to information hosted on Oracle ERP and EPM applications.

- EssCreateDrillThruURL
- EssDeleteDrillThruURL
- EssGetCellDrillThruReports
- EssGetDrillThruURL
- EssListDrillThruURLs
- EssUpdateDrillThruURL

The following Drill-through functions retrieve data from connected relational databases.

See “Drill-Through Constant and Structure Definitions” on page 110.

Refer to these Drill-Through functions in the Grid API:

- EssGDTConnect
- EssGDTEndDrillThrough
- EssGDTExecuteReport
- EssGDTGetData
- EssGDTGetHeader
- EssGDTGetInfo
- EssGDTListReports
- EssGDTRequestDrillThrough
- EssGDTSetInfo

**Note:** In future releases, the C Main API Drill-Through functions below will be deprecated and replaced by the corresponding Grid API functions. Programs should use the Grid API functions listed above.
C Main API File Functions

File functions enable an application to use predefined report scripts, data files and calculation scripts against the active database. There are also functions for importing and exporting data to and from both text and binary files.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EssArchiveBegin</td>
<td>Prepares database for archive by setting READ-ONLY status.</td>
</tr>
<tr>
<td>EssArchiveEnd</td>
<td>After archive, returns database status to READ-WRITE.</td>
</tr>
<tr>
<td>EssCalcFile</td>
<td>Executes a calc script against the active database from a file.</td>
</tr>
<tr>
<td>EssExport</td>
<td>Exports data from the current database to a text file.</td>
</tr>
<tr>
<td>EssImport</td>
<td>Imports data from text files and other sources to the current database.</td>
</tr>
<tr>
<td>EssImportASO</td>
<td>Imports data from different sources to an aggregate storage database.</td>
</tr>
<tr>
<td>EssListDbFiles</td>
<td>Retrieves information on specified index and data files</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>EssReportFile</td>
<td>Sends a report specification to the active database from a file.</td>
</tr>
<tr>
<td>EssSetDefaultCalcFile</td>
<td>Sets the default calc script for the active database from a calc script file.</td>
</tr>
<tr>
<td>EssUpdateFile</td>
<td>Sends an update specification to the active database from a file.</td>
</tr>
<tr>
<td>EssUpdateFileEx</td>
<td>Sends an update specification to the active database from a file, capturing any data load errors.</td>
</tr>
<tr>
<td>EssUpdateFileASO</td>
<td>Sends an update specification to the active aggregate storage database from a file.</td>
</tr>
<tr>
<td>EssUpdateFileASOEx</td>
<td>Sends an update specification to the active aggregate storage database from a file, capturing any data load errors.</td>
</tr>
<tr>
<td>EssUpdateFileUTF8ASO</td>
<td>Sends an update specification to the active aggregate storage database from a UTF-8-encoded file.</td>
</tr>
<tr>
<td>EssUpdateFileUTF8ASOEx</td>
<td>Sends an update specification to the active aggregate storage database from a UTF-8-encoded file, capturing any data load errors.</td>
</tr>
<tr>
<td>EssUpdateFileUtf8Ex</td>
<td>Sends an update specification to the active database from a UTF-8-encoded file, capturing any data load errors.</td>
</tr>
<tr>
<td>EssDisplayTriggers</td>
<td>Returns a list of all triggers associated with a database.</td>
</tr>
<tr>
<td>EssMdxTrig</td>
<td>Manipulates triggers based on the operations contained in an MDX language file.</td>
</tr>
<tr>
<td>EssListSpoolFiles</td>
<td>Returns a list of all the spool files associated with a database.</td>
</tr>
<tr>
<td>EssGetSpoolFile</td>
<td>Returns a specific spool file associated with a database.</td>
</tr>
<tr>
<td>EssDeleteAllSplFiles</td>
<td>Deletes all the spool files associated with a database.</td>
</tr>
<tr>
<td>EssDeleteSplFile</td>
<td>Deletes a specific spool file.</td>
</tr>
</tbody>
</table>

**C Main API Group Administration Functions**

These functions create groups, set and modify group attributes, and obtain information about existing groups.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EssListGroups</td>
<td>Lists all groups who have access to a particular Essbase Server.</td>
</tr>
<tr>
<td>EssCreateGroup</td>
<td>Creates a new group.</td>
</tr>
<tr>
<td>EssDeleteGroup</td>
<td>Deletes an existing group.</td>
</tr>
<tr>
<td>EssRenameGroup</td>
<td>Renames an existing group.</td>
</tr>
<tr>
<td>EssGetGroup</td>
<td>Gets a group information structure, which contains security information for the group.</td>
</tr>
<tr>
<td>EssSetGroup</td>
<td>Sets a group information structure.</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>EssGetGroupList</td>
<td>Gets the list of users who are members of a group (or the list of groups to which a user belongs).</td>
</tr>
<tr>
<td>EssSetGroupList</td>
<td>Sets list of users who are members of group.</td>
</tr>
<tr>
<td>EssAddToGroup</td>
<td>Adds user to a list of group members.</td>
</tr>
<tr>
<td>EssDeleteFromGroup</td>
<td>Removes a user from a list of group members.</td>
</tr>
</tbody>
</table>

### C Main API Initialization and Login Functions

These functions initialize the API, and log in and out of the Essbase Server. They also obtain version information, and enable an application to create and delete local contexts.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EssAutoLogin</td>
<td>Displays a dialog box which allows the user to log in to an Essbase Server, and optionally selects an active application and database.</td>
</tr>
<tr>
<td>EssCreateLocalContext</td>
<td>Creates a local API context for use in local API operations.</td>
</tr>
<tr>
<td>EssDeleteLocalContext</td>
<td>Releases a local context previously created by EssCreateLocalContext().</td>
</tr>
<tr>
<td>EssGetAPIVersion</td>
<td>Gets the full version number of the connected API client module.</td>
</tr>
<tr>
<td>EssGetVersion</td>
<td>Gets the full version number of the connected Essbase Server.</td>
</tr>
<tr>
<td>EssInit</td>
<td>Initializes the API and message database.</td>
</tr>
<tr>
<td>EssLogin</td>
<td>Logs a user in to the Essbase Server.</td>
</tr>
<tr>
<td>EssLoginAs</td>
<td>Logs in to the Essbase Server as another user.</td>
</tr>
<tr>
<td>EssLoginEx</td>
<td>Logs in to the Essbase Server using an authentication token.</td>
</tr>
<tr>
<td>EssLoginExAs</td>
<td>Logs in to the Essbase Server as another user, using an authentication token.</td>
</tr>
<tr>
<td>EssLoginSetPassword</td>
<td>Logs in a user, and changes the password.</td>
</tr>
<tr>
<td>EssLogout</td>
<td>Logs a user out from an Essbase Server.</td>
</tr>
<tr>
<td>EssLogoutUser</td>
<td>Allows a Supervisor or Application Designer to disconnect another user from an Essbase Server.</td>
</tr>
<tr>
<td>EssLogSize</td>
<td>Returns the size of the Essbase Server log file (essbase.log), or of the application log file (appname.log).</td>
</tr>
<tr>
<td>EssShutdownServer</td>
<td>Allows a Supervisor to remotely stop the Essbase Server.</td>
</tr>
<tr>
<td>EssTerm</td>
<td>Terminates the API and releases all system resources used by the API.</td>
</tr>
<tr>
<td>EssValidateHCtx</td>
<td>Validates a specific API context handle (hCtx).</td>
</tr>
<tr>
<td>EssWriteToFileLog</td>
<td>Writes a message to the Essbase Server log file (essbase.log), or to the application log file (appname.log).</td>
</tr>
</tbody>
</table>
## C Main API LRO Functions

These functions create, retrieve and delete LROs and return information about them.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EssLROAddObject</td>
<td>Links a reporting object to a data cell in an Essbase database.</td>
</tr>
<tr>
<td>EssLRODeleteCellObjects</td>
<td>Deletes all objects linked to a given data cell in an Essbase database.</td>
</tr>
<tr>
<td>EssLRODeleteObject</td>
<td>Deletes a specific object linked to a data cell in an Essbase database.</td>
</tr>
<tr>
<td>EssLROGetCatalog</td>
<td>Retrieves a list of LRO catalog entries for a given data cell in an Essbase database.</td>
</tr>
<tr>
<td>EssLROGetCatalogBatch</td>
<td>Retrieves a list of LRO catalog entries for multiple data cells in an Essbase database.</td>
</tr>
<tr>
<td>EssLROGetObject</td>
<td>Retrieves an object linked to a data cell in an Essbase database.</td>
</tr>
<tr>
<td>EssLROListObjects</td>
<td>Retrieves a list of all objects linked to cells in the active database for a given user name and/or modification date.</td>
</tr>
<tr>
<td>EssLROPurgeObjects</td>
<td>Deletes all objects linked to cells in the active database for a given user name and/or modification date.</td>
</tr>
<tr>
<td>EssLROUpdateObject</td>
<td>Stores an updated version of an LRO on the server.</td>
</tr>
</tbody>
</table>

## C Main API Location Aliases Functions

These functions create, delete and list location aliases.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EssCreateLocationAlias</td>
<td>Maps an alias name to the host name, application name, database name, user login name, and user password</td>
</tr>
<tr>
<td>EssDeleteLocationAlias</td>
<td>Deletes an existing location alias</td>
</tr>
<tr>
<td>EssGetLocationAliasList</td>
<td>Returns all location aliases and the names to which the location aliases are mapped</td>
</tr>
</tbody>
</table>

## C Main API Memory Allocation Functions

These functions manage memory for an application by allocating, reallocating and freeing blocks of memory.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EssAlloc</td>
<td>Allocates a block of memory, using the defined memory allocation scheme.</td>
</tr>
<tr>
<td>EssRealloc</td>
<td>Reallocates a previously-allocated block of memory.</td>
</tr>
<tr>
<td>EssFree</td>
<td>Frees a previously allocated block of memory, using the defined memory allocation scheme.</td>
</tr>
</tbody>
</table>
C Main API Miscellaneous Functions

These functions manage asynchronous processes, obtain state information, handle log files, and retrieve messages.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EssGetProcessState</td>
<td>Gets the current state of an asynchronous process, such as a calculate or data import.</td>
</tr>
<tr>
<td>EssCancelProcess</td>
<td>Cancels an asynchronous process which has not yet completed.</td>
</tr>
<tr>
<td>EssGetLogFile</td>
<td>Copies all or part of an application log file from the server to the client.</td>
</tr>
<tr>
<td>EssDeleteLogFile</td>
<td>Deletes an application log file on the server.</td>
</tr>
<tr>
<td>EssGetGlobalState</td>
<td>Gets the server global state structure which contains parameters for system administration.</td>
</tr>
<tr>
<td>EssSetGlobalState</td>
<td>Sets the server global state structure which contains parameters for system administration.</td>
</tr>
<tr>
<td>EssSetPath</td>
<td>Sets the ESSBASEPATH environment variable for the current process.</td>
</tr>
</tbody>
</table>

C Main API Object Functions

These functions create, delete, move and copy objects. They also retrieve and display object information and control access to objects.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EssGetLocalPath</td>
<td>Gets the full local file for an object file on the client.</td>
</tr>
<tr>
<td>EssListObjects</td>
<td>Lists all objects of types specified.</td>
</tr>
<tr>
<td>EssGetObjectInfo</td>
<td>Gets information about a specified object.</td>
</tr>
<tr>
<td>EssGetObject</td>
<td>Copies an object from the server to a local file, and optionally locks it.</td>
</tr>
<tr>
<td>EssPutObject</td>
<td>Copies an object from a local file to the server, and optionally unlocks it.</td>
</tr>
<tr>
<td>EssLockObject</td>
<td>Locks an object on the server to prevent other users from updating it.</td>
</tr>
<tr>
<td>EssUnlockObject</td>
<td>Unlocks a locked object on the server.</td>
</tr>
<tr>
<td>EssCreateObject</td>
<td>Creates a new object.</td>
</tr>
<tr>
<td>EssDeleteObject</td>
<td>Deletes an existing object.</td>
</tr>
<tr>
<td>EssRenameObject</td>
<td>Renames an existing object.</td>
</tr>
<tr>
<td>EssCopyObject</td>
<td>Copies an object.</td>
</tr>
</tbody>
</table>

C Main API Partition Functions

These functions manage partition operations on a database.
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EssPartitionApplyOtlChangeFile</td>
<td>Tells the server to apply metadata changes to files.</td>
</tr>
<tr>
<td>EssPartitionApplyOtlChangeFileEx</td>
<td>Tells the server to apply metadata changes to files.</td>
</tr>
<tr>
<td>EssPartitionApplyOtlChangeRecs</td>
<td>Tells the server to apply metadata changes to records.</td>
</tr>
<tr>
<td>EssPartitionCloseDefFile</td>
<td>Closes the shared partition definition file.</td>
</tr>
<tr>
<td>EssPartitionFreeDefCtx</td>
<td>Frees memory dynamically allocated under shared partition context structures.</td>
</tr>
<tr>
<td>EssPartitionFreeOtlChanges</td>
<td>Frees up memory allocated by the ReadMetaChange routine.</td>
</tr>
<tr>
<td>EssPartitionGetAreaCellCount</td>
<td>Returns the number of cells in the specified slice string.</td>
</tr>
<tr>
<td>EssPartitionGetList</td>
<td>Returns a list of the partition partition definitions in which the currently selected database participates.</td>
</tr>
<tr>
<td>EssPartitionGetOtlChanges</td>
<td>Pulls meta data changes from a given source.</td>
</tr>
<tr>
<td>EssPartitionGetReplCells</td>
<td>Replicates all data cells that are identified in the replication partition from the source database to the selected target database.</td>
</tr>
<tr>
<td>EssPartitionNewDefFile</td>
<td>Creates and opens a new shared partition, definition file based upon input parameters supplied.</td>
</tr>
<tr>
<td>EssPartitionOpenDefFile</td>
<td>Opens an existing shared partition definition file.</td>
</tr>
<tr>
<td>EssPartitionPurgeOtlChangeFile</td>
<td>Purges meta changes made previous to the time specified with the TimeStamp parameter.</td>
</tr>
<tr>
<td>EssPartitionPutReplCells</td>
<td>Replicates all data cells that are identified in the replication partition from the selected source database to the target database.</td>
</tr>
<tr>
<td>EssPartitionReadDefFile</td>
<td>Replicates all data cells that are identified in the replication partition from the selected source database to the target database.</td>
</tr>
<tr>
<td>EssPartitionReadOtlChangeFile</td>
<td>Reads meta changes from a file into memory.</td>
</tr>
<tr>
<td>EssPartitionReplaceDefFile</td>
<td>Tells the server that a new shared partition file has been sent, which replaces any existing file for this database.</td>
</tr>
<tr>
<td>EssPartitionResetOtlChangeTime</td>
<td>Takes two partitions, one source and one destination. It takes the &quot;last meta change&quot; time from the source partition and assigns it as the &quot;last meta change&quot; time of the destination partition.</td>
</tr>
<tr>
<td>EssPartitionValidateDefinition</td>
<td>Performs full validation of the specified partition definition; that is, validates the source and target parts of one partition definition. Useful during creation of a new or modification of an existing partition definition.</td>
</tr>
<tr>
<td>EssPartitionValidateLocal</td>
<td>Performs partial validation of all partition definitions on the specified server. Useful to ascertain the validity of partition definitions after metadata changes; for example, after database restructuring.</td>
</tr>
<tr>
<td>EssPartitionWriteDefFile</td>
<td>Writes the current memory version of the shared partition definition file to disk.</td>
</tr>
</tbody>
</table>
# C Main API Performance Stats Functions

These functions provide I/O performance statistics on threads, databases and applications.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EssDumpPerfStats</td>
<td>Provides a pointer to the character array that contains performance statistics tables</td>
</tr>
<tr>
<td>EssGetStatBufSize</td>
<td>Provides a pointer to the size of the buffer needed for the performance statistics tables</td>
</tr>
<tr>
<td>EssResetPerfStats</td>
<td>Resets values in the performance statistics tables to zero</td>
</tr>
</tbody>
</table>

# C Main API Reporting, Updating, and Calculation Functions

These functions carry out reporting (retrieving data), updating (loading data) and calculation (aggregating data) tasks against the active database.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EssReport</td>
<td>Sends a report specification to the active database as a single string.</td>
</tr>
<tr>
<td>EssBeginReport</td>
<td>Starts sending a report specification to the active database.</td>
</tr>
<tr>
<td>EssEndReport</td>
<td>Marks the end of a report specification being sent to the active database.</td>
</tr>
<tr>
<td>EssUpdate</td>
<td>Sends an update to the active database as a single string.</td>
</tr>
<tr>
<td>EssUpdateFileASO</td>
<td>Sends an update specification to the active aggregate storage database from a file.</td>
</tr>
<tr>
<td>EssUpdateFileUTF8ASO</td>
<td>Sends an update specification to the active aggregate storage database from a UTF-8-encoded file.</td>
</tr>
<tr>
<td>EssBeginUpdate</td>
<td>Starts sending an update specification to the active database.</td>
</tr>
<tr>
<td>EssEndUpdate</td>
<td>Marks the end of an update specification being sent to the active database.</td>
</tr>
<tr>
<td>EssGetString</td>
<td>Gets a string of data from the active database.</td>
</tr>
<tr>
<td>EssSendString</td>
<td>Sends a string of data to the active database.</td>
</tr>
<tr>
<td>EssCalc</td>
<td>Sends and optionally executes a calc script against the active database as a single string.</td>
</tr>
<tr>
<td>EssBeginCalc</td>
<td>Starts sending a calc script and optionally executes it against the active database.</td>
</tr>
<tr>
<td>EssEndCalc</td>
<td>Marks the end of a calc script being sent to the active database.</td>
</tr>
<tr>
<td>EssDefaultCalc</td>
<td>Executes the default calculation for the active database.</td>
</tr>
<tr>
<td>EssGetDefaultCalc</td>
<td>Gets the default calc script for the active database.</td>
</tr>
<tr>
<td>EssListCalcFunctions</td>
<td>Lists all available calculator functions.</td>
</tr>
<tr>
<td>EssSetDefaultCalc</td>
<td>Sets the default calc script for a database.</td>
</tr>
</tbody>
</table>
C Main API Security Filter Functions

Security filter functions create filters, set filter contents, assign filters to user groups, display filter lists for data bases, and obtain other data about security filters.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EssListFilters</td>
<td>Lists all filters for a database.</td>
</tr>
<tr>
<td>EssGetFilter</td>
<td>Starts getting the contents of a filter.</td>
</tr>
<tr>
<td>EssGetFilterRow</td>
<td>Gets the next row of a filter.</td>
</tr>
<tr>
<td>EssCreateFilter</td>
<td>Creates a filter. Starts setting the contents of the filter.</td>
</tr>
<tr>
<td>EssSetFilter</td>
<td>Creates or replaces a filter. Starts setting the contents of the filter.</td>
</tr>
<tr>
<td>EssSetFilterRow</td>
<td>Gets the next row of a filter.</td>
</tr>
<tr>
<td>EssGetFilterList</td>
<td>Gets the list of users who are assigned a filter.</td>
</tr>
<tr>
<td>EssSetFilterList</td>
<td>Sets the list of users who are assigned a filter.</td>
</tr>
<tr>
<td>EssDeleteFilter</td>
<td>Deletes an existing filter.</td>
</tr>
<tr>
<td>EssRenameFilter</td>
<td>Renames an existing filter.</td>
</tr>
<tr>
<td>EssCopyFilter</td>
<td>Copies an existing filter.</td>
</tr>
<tr>
<td>EssVerifyFilter</td>
<td>Verifies the syntax of a series of filter row strings against a specified database.</td>
</tr>
<tr>
<td>EssVerifyFilterRow</td>
<td>Verifies the syntax of a single filter row string against a specified database.</td>
</tr>
</tbody>
</table>

C Main API Substitution Variables Functions

These functions create, retrieve and delete substitution variables and return information about them.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EssCreateVariable</td>
<td>This function creates a new substitution variable or modifies an existing substitution variable if the variable name already exists with the identical server, application, and database values.</td>
</tr>
<tr>
<td>EssDeleteVariable</td>
<td>This function deletes a substitution variable.</td>
</tr>
<tr>
<td>EssGetVariable</td>
<td>This function retrieves the value of a substitution variable.</td>
</tr>
<tr>
<td>EssListVariables</td>
<td>This function lists all substitution variables that conform to the input criteria.</td>
</tr>
</tbody>
</table>
C Main API User Administration Functions

User administration functions create users, assign their passwords, and their access to databases, applications, and calc scripts. Functions are also provided to retrieve information about user capabilities.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EssListUsers</td>
<td>Lists all users who have access to a particular Essbase Server.</td>
</tr>
<tr>
<td>EssCreateUser</td>
<td>Creates a new user.</td>
</tr>
<tr>
<td>EssDeleteUser</td>
<td>Deletes an existing user.</td>
</tr>
<tr>
<td>EssRenameUser</td>
<td>Renames an existing user.</td>
</tr>
<tr>
<td>EssGetUser</td>
<td>Gets a user information structure, which contains security information for user.</td>
</tr>
<tr>
<td>EssSetUser</td>
<td>Sets a user information structure that contains security information for user.</td>
</tr>
<tr>
<td>EssResetUser</td>
<td>Resets the user's security structure to its initial state.</td>
</tr>
<tr>
<td>EssSetPassword</td>
<td>Sets a user's password, erasing the existing password.</td>
</tr>
<tr>
<td>EssGetApplicationAccess</td>
<td>Gets a list of user application access structures, which contain information about user access to applications.</td>
</tr>
<tr>
<td>EssSetApplicationAccess</td>
<td>Sets a list of user application access structures.</td>
</tr>
<tr>
<td>EssGetDatabaseAccess</td>
<td>Gets a list of user database access structures.</td>
</tr>
<tr>
<td>EssSetDatabaseAccess</td>
<td>Sets a list of user database access structures.</td>
</tr>
<tr>
<td>EssGetCalcList</td>
<td>Gets the list of calc scripts objects accessible to the user.</td>
</tr>
<tr>
<td>EssSetCalcList</td>
<td>Sets the list of calc script objects which are available to a user.</td>
</tr>
<tr>
<td>EssListConnections</td>
<td>Lists all users who are connected to the current application and database.</td>
</tr>
<tr>
<td>EssListLogins</td>
<td>Lists information about currently connected users.</td>
</tr>
<tr>
<td>EssListRequests</td>
<td>Lists information about current Essbase user sessions or requests.</td>
</tr>
<tr>
<td>EssKillRequest</td>
<td>Terminates all or specific Essbase user sessions or requests.</td>
</tr>
<tr>
<td>EssListLocks</td>
<td>Lists all users who are connected to a specific application and database.</td>
</tr>
<tr>
<td>EssRemoveLocks</td>
<td>Removes all data block locks held by a user on a database.</td>
</tr>
</tbody>
</table>

C Main API User and Group Identity Functions

User and group identity functions are enhanced to enable the specification of user directories and unique identity attributes to identify users and groups that are hosted in a directory.
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EssAddToGroupEx</td>
<td>Adds a user to the specified group. Similar to EssAddToGroup, but can accept a user directory specification or unique identity attribute.</td>
</tr>
<tr>
<td>EssCreateExtGroup</td>
<td>Creates a group in the external user directory.</td>
</tr>
<tr>
<td>EssDeleteFromGroupEx</td>
<td>Removes a user from a group. Similar to EssDeleteFromGroup, but can accept a user directory specification or unique identity attribute.</td>
</tr>
<tr>
<td>EssDeleteGroupEx</td>
<td>Deletes an existing group. Similar to EssDeleteGroup, but can accept a user directory specification or unique identity attribute.</td>
</tr>
<tr>
<td>EssDeleteUserEx</td>
<td>Deletes a user. Similar to EssDeleteUser, but can accept a user directory specification or unique identity attribute.</td>
</tr>
<tr>
<td>EssGetApplicationAccessEx</td>
<td>Gets a list of user or group application access structures, which contain information about user or group access to applications. Similar to EssGetApplicationAccess, but can accept a user directory specification or unique identity attribute.</td>
</tr>
<tr>
<td>EssGetDatabaseAccessEx</td>
<td>Gets a list of user database access structures, which contain information about user access to databases. Similar to EssGetDatabaseAccess, but can accept a user directory specification or unique identity attribute.</td>
</tr>
<tr>
<td>EssGetGroupInfoEx</td>
<td>Gets a group information structure, which contains security information for the group. Similar to EssGetGroup, but can accept a user directory specification or unique identity attribute.</td>
</tr>
<tr>
<td>EssGetGroupListEx</td>
<td>Gets the list of users who are members of a group or the list of groups to which the user belongs. Similar to EssGetGroupList, but can accept a user directory specification or unique identity attribute.</td>
</tr>
<tr>
<td>EssGetUserInfoEx</td>
<td>Gets a user information structure, which contains security information for the user. Similar to EssGetUser, but can accept a user directory specification or unique identity attribute.</td>
</tr>
<tr>
<td>EssKillRequestEx</td>
<td>Terminates specific user sessions or requests. Similar to EssKillRequest, but the input structure can include user directories and unique identity attributes.</td>
</tr>
<tr>
<td>EssListConnectionsEx</td>
<td>Lists all users who are connected to the currently logged in server or application. Similar to EssListConnections, but includes users hosted in a user directory.</td>
</tr>
<tr>
<td>EssListGroupsInfoEx</td>
<td>Lists all groups who have access to a particular Essbase Server, application or database. Similar to EssListGroups, but the group list structure can include user directories and unique identity attributes.</td>
</tr>
<tr>
<td>EssListLocksEx</td>
<td>Lists all users who are connected to a specific application and database, together with a count of data blocks which they currently have locked. Similar to EssListLocks, but includes users hosted in a user directory.</td>
</tr>
<tr>
<td>EssListLoginsEx</td>
<td>Returns the list of log in instances in the current session. Similar to EssListLogins, but includes users hosted in a user directory.</td>
</tr>
<tr>
<td>EssListRequestsEx</td>
<td>Returns information about active sessions and requests. Similar to EssListRequests, but includes users hosted in a user directory.</td>
</tr>
<tr>
<td>EssListUsersInfoEx</td>
<td>Lists all users who have access to a particular Essbase Server, application or database. Similar to EssListUsers, but the user list structure can include user directories and unique identity attributes.</td>
</tr>
</tbody>
</table>
### Function Description

**EssSetApplicationAccessEx**
Sets a list of user application access structures, which contain information about user access to applications. Similar to EssSetApplicationAccess, but the input structure can include user directories and unique identity attributes.

**EssSetCalcListEx**
Sets the calculation list accessible to the specified user or group. Similar to EssSetCalcList, but includes users and groups hosted in a user directory.

**EssSetDatabaseAccessEx**
Sets a list of user database access structures, which contain information about user access to databases. Similar to EssSetDatabaseAccess, but the input structure can include user directories and unique identity attributes.

**EssSetFilterListEx**
Sets the list of groups or users that are assigned to a filter. The count parameter controls the number of groups or users assigned to the filter. A count of zero removes all the groups or users from the list.

**EssSetGroupListEx**
Sets the list of users who are members of a group. Similar to EssSetGroupList, but can accept a user directory specification or unique identity attribute.

### C Main API Shared Services Functions

Shared Services User Management enables centralized management of user access rights and accessibility to applications created under various projects of different products.

The following functions help you migrate Essbase to Shared Services mode. After migration, you can manage users, groups, and applications in Shared Services mode instead of in Essbase native mode.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>EssSetSSSecurityMode</strong></td>
<td>Migrates Essbase Server and any existing users and groups to Oracle Hyperion Enterprise Performance Management System security mode.</td>
</tr>
<tr>
<td><strong>EssSetUserToSS</strong></td>
<td>Migrates a user to EPM System security mode.</td>
</tr>
<tr>
<td><strong>EssSetGroupToSS</strong></td>
<td>Migrates a group to EPM System security mode.</td>
</tr>
<tr>
<td><strong>EssSetUsersToSS</strong></td>
<td>Migrates all users to EPM System security mode.</td>
</tr>
<tr>
<td><strong>EssSetGroupToSS</strong></td>
<td>Migrates all groups to EPM System security mode.</td>
</tr>
<tr>
<td><strong>EssGetEssbaseSecurityMode</strong></td>
<td>Displays the type of security in use.</td>
</tr>
<tr>
<td><strong>EssListSSMigrFailedUsers</strong></td>
<td>Displays users that did not successfully migrate to Shared Services.</td>
</tr>
<tr>
<td><strong>EssListSSMigrFailedGroups</strong></td>
<td>Displays groups that did not successfully migrate to Shared Services.</td>
</tr>
<tr>
<td><strong>EssReRegisterApplication</strong></td>
<td>Re-establishes one or all Essbase applications as Shared Services applications.</td>
</tr>
<tr>
<td><strong>EssSetEasLocation</strong></td>
<td>Set or change the Essbase Administration Server location that will be registered with Shared Services upon application creation or migration.</td>
</tr>
</tbody>
</table>
C Main API Unicode Mode Functions

Essbase Server allows the creation of Unicode mode applications, or migration of non-Unicode mode applications to Unicode mode, only when it is in Unicode mode.

The following functions help you work with the Essbase Server and applications in Unicode mode.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EssSetServerMode</td>
<td>Sets the mode of Essbase Server to be Unicode or non-Unicode.</td>
</tr>
<tr>
<td>EssGetServerMode</td>
<td>Indicates whether the Essbase Server is in Unicode mode or non-Unicode mode.</td>
</tr>
<tr>
<td>EssCreateApplicationEx</td>
<td>Creates a Unicode mode application.</td>
</tr>
<tr>
<td>EssConvertApplicationtoUnicode</td>
<td>Converts a non Unicode mode application to a Unicode mode application.</td>
</tr>
<tr>
<td>EssGetApplicationInfo and EssGetApplicationInfoEx</td>
<td>Returns application information, including locale information.</td>
</tr>
<tr>
<td>EssUpdateFileUTF8ASO</td>
<td>Sends an update specification to the active aggregate storage database from a UTF-8-encoded file.</td>
</tr>
<tr>
<td>EssUpdateFileUTF8ASOEx</td>
<td>Sends an update specification to the active aggregate storage database from a UTF-8-encoded file, capturing any data load errors.</td>
</tr>
<tr>
<td>EssUpdateFileUtf8Ex</td>
<td>Sends an update specification to the active database from a UTF-8-encoded file, capturing any data load errors.</td>
</tr>
</tbody>
</table>

C Main API Function Reference

Consult the Contents pane for the alphabetical list of C Main API functions.

EssAddToGroup

Adds a user to the list of group members.

**Syntax**

```
ESS_FUNC_M EssAddToGroup (hCtx, GroupName, UserName);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>GroupName</td>
<td>ESS_STR_T</td>
<td>Group name.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UserName</td>
<td>ESS_STR_T</td>
<td>Name of user to add to group list.</td>
</tr>
</tbody>
</table>

**Notes**

- This function also adds the user to the list of group members and the group to the user's own list of groups.

**Return Value**

None.

**Access**

This function requires the caller to have Create/Delete User privilege (ESS_PRIV_USERCREATE) for the logged in server.

**Example**

```c
ESS_FUNC_M
ESS_AddUser (ESS_HCTX_T hCtx)
{
    ESS_FUNC_M sts = ESS_STS_NOERR;
    ESS_STR_T GroupName;
    ESS_STR_T UserName;
    GroupName = "PowerUsers";
    UserName = "Jim Smith";

    sts = EssAddToGroup (hCtx, GroupName, UserName);

    return (sts);
}
```

**See Also**

- EssAddToGroupEx
- EssDeleteFromGroup
- EssGetGroupList
- EssListGroups
- EssSetGroupList

### EssAddToGroupEx

Adds a user to the specified group. Similar to EssAddToGroup, but can accept a user directory specification or unique identity attribute for GroupId or UserId.

**Syntax**

```c
ESS_FUNC_M EssAddToGroupEx (hCtx, GroupId, UserId, bUsingIdentity);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle (input).</td>
</tr>
<tr>
<td>Parameter</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>GroupId</td>
<td>ESS_STR_T</td>
<td>Group name or identity (input). Can be specified as groupname@provider or as a unique identity attribute.</td>
</tr>
<tr>
<td>bIsGroupId</td>
<td>ESS.Bool_T</td>
<td>Input. Indicates if GroupId is a name or an identity. If TRUE, GroupId is an identity.</td>
</tr>
<tr>
<td>UserId</td>
<td>ESS_STR_T</td>
<td>Name of user to add to group (input). Can be specified as username@provider or as a unique identity attribute.</td>
</tr>
<tr>
<td>bUsingIdentity</td>
<td>ESS.Bool_T</td>
<td>Input. Indicates if UserID is a name or an identity. If TRUE, UserID is an identity.</td>
</tr>
</tbody>
</table>

**Notes**

- The API can accept an identity or a group name. The group name can be specified as `groupname@provider`.

**Return Value**

None.

**Access**

This function requires the caller to have Create/Delete User privilege (ESS_PRIV_USERCREATE) for the logged in server.

**Example**

```c
void DisplayUserList(ESS_USHORT_T count, ESS_PSTR_T UserList)
{
    ESS_USHORT_T i;

    for (i = 0; i < count; i++)
    {
        if (UserList [i])
            printf ("%s\n", UserList[i]);
    }
}

ESS_FUNC_M ESS_AddUser (ESS_HCTX_T    hCtx)
{
    ESS_STS_T sts  = ESS_STS_NOERR;
    ESS_STR_T groupId, userId;
    ESS_BOOL_T bGroupId, bUserId;
    ESS_USHORT_T type;
    ESS_USHORT_T count;
    ESS_BOOL_T bUsingIdentity;
    ESS_PSTR_T pUserList;

    groupId = "IDRegularGroup@ldap";
    bGroupId = ESS_FALSE;
    userId = "IDUser6";
    bUserId = ESS_FALSE;
    sts = EssAddToGroupEx(hCtx, groupId, bGroupId, userId, bUserId);
    printf("EssAddToGroupEx sts: %ld\n", sts);
```
if(!sts)
{
    sts = EssGetGroupListEx(hCtx, groupId, bisIdentity, type, &count, &bUsingIdentity, &pUserList);
    printf("EssGetGroupListEx sts: %ld\n", sts);
    if(!sts)
    {
        if(pUserList)
        {
            printf ("\n---User/Group list for %s:\n", groupId);
            DisplayUserList(count, pUserList);
        }
        else
            printf ("\nUser list is empty\n");
    }
}
return (sts);

See Also

- EssDeleteFromGroupEx
- EssGetGroupListEx
- EssListGroupsInfoEx

**EssAlloc**

Allocates a block of memory, using the defined memory allocation scheme.

**Syntax**

```c
ESS_FUNC_M EssAlloc (hInstance, Size, ppBlock);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>ESS_HINST_T</td>
<td>API instance handle.</td>
</tr>
<tr>
<td>Size</td>
<td>ESS_SIZE_T</td>
<td>Size of memory block to allocate.</td>
</tr>
<tr>
<td>ppBlock</td>
<td>ESS_PPVOID_T</td>
<td>Address of pointer to receive allocated memory block.</td>
</tr>
</tbody>
</table>

**Notes**

- This function allocates memory using the user-supplied memory management function passed to the EssInit() function. If no such functions are supplied, the default memory allocation function (dependent on the platform) will be used.
- Memory allocated using this function should always be reallocated or freed by using the EssRealloc() and EssFree() functions respectively.
- It is generally not advisable to allocate a block of zero size, as the effects of such an allocation are platform- and compiler-dependent.
**Return Value**

Returns a pointer to the allocated memory block in `ppBlock`.

**Access**

This function requires no special privileges.

**Example**

```c
ESS_FUNC_M ESS_GetAppActive (ESS_HCTX_T hCtx,
    ESS_HINST_T hInst)
{
    ESS_FUNC_M     sts = ESS_STS_NOERR;
    ESS_STR_T     pDbName;
    ESS_STR_T     pAppName;
    ESS_ACCESS_T  Access;

    if ((sts = EssAlloc (hInst, 80, (ESS_PPVOID_T)&pAppName)) == 0)
    {
        if ((sts = EssAlloc (hInst, 80, (ESS_PPVOID_T)&pDbName)) == 0)
        {
            if ((sts = EssGetActive (hCtx, &pAppName, &pDbName, &Access)) == 0)
            {
                if (pAppName)
                {
                    if (*pAppName)
                        printf (“Current active application is [%s]\r\n”, pAppName);
                    else
                        printf (“No active Application is set\r\n”);
                    printf (“\r\n”);
                }
                EssFree (hInst, pDbName);
            }
            EssFree (hInst, pAppName);
        }
        return (sts);
    }
}
```

**See Also**

- `EssFree`
- `EssInit`
- `EssRealloc`

**EssArchive**

No longer in use.

This function is retained for compatibility with earlier versions of Essbase only. For current Essbase archiving, see `EssArchiveBegin()` and `EssArchiveEnd()`. This function now returns the error message `ESS_STS_OBSOLETE`.

**See Also**

- `EssRestore`
EssArchiveBegin

Prepares the server for archiving by changing server mode to Read-Only.

Syntax

```c
ESS_FUNC_M EssArchiveBegin (hCtx,AppName,DbName,FileName);
```

Parameter | Data Type | Description
---|---|---

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle</td>
</tr>
<tr>
<td>AppName</td>
<td>ESS_STR_T</td>
<td>Name of application to archive</td>
</tr>
<tr>
<td>DbName</td>
<td>ESS_STR_T</td>
<td>Name of database to archive</td>
</tr>
<tr>
<td>FileName</td>
<td>ESS_STR_T</td>
<td>Name of file to contain archive information</td>
</tr>
</tbody>
</table>

Notes

- This function changes server mode to Read-Only. This mode allows the database administrator to back up all the files on the server and prevents writing to the files during the backup. The database files to back up are listed in the app\db directory specified by the FileName parameter.
- Any existing information in the specified file is overwritten by the archived data.

Return Value

None.

Access

The caller must have at least read access (ESS_PRIV_READ) to the database, and must select it as the active database using EssSetActive().

Example

```c
    ESS_FUNC_M
    ESSArchiveBegin(ESS_HCTX_T hCtx)
    {
        ESS_FUNC_M       sts = ESS_STS_NOERR;
        ESS_STR_T    AppName;
        ESS_STR_T    DbName;
        ESS_STR_T    FileName;
        AppName  = "Sample";
        DbName   = "Basic";
        FileName = "Test.arc";

        /* Begin Archive */
        sts = EssArchiveBegin(hCtx,AppName,DbName,FileName);
    }
```
return (sts);
}

See Also

- EssArchiveEnd
- EssRestore

**EssArchiveDatabase**

Creates an archive of a database in a specified backup file.

**Syntax**

```
ESS_FUNC_M EssArchiveDatabase (hCtx, AppName, DbName, BackupFileName, OptionsFileName, bOverWrite);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>Login context</td>
</tr>
<tr>
<td>AppName</td>
<td>ESS_STR_T</td>
<td>Application Name</td>
</tr>
<tr>
<td>DbName</td>
<td>ESS_STR_T</td>
<td>Database Name</td>
</tr>
<tr>
<td>BackupFileName</td>
<td>ESS_STR_T</td>
<td>Full path to the backup file in which to archive data. Specify the full path, for example:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>c:\hyperion\Test.arc</td>
</tr>
<tr>
<td>OptionsFileName</td>
<td>ESS_FILENAME_T</td>
<td>Reserved for the future.</td>
</tr>
<tr>
<td>bOverWrite</td>
<td>ESS_BOOL_T</td>
<td>Boolean:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_TRUE—overwrite existing backup file</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_FALSE—do not overwrite</td>
</tr>
</tbody>
</table>

**Note:** Works only at the database level. The AppName parameter specifies an Application in order to access the database residing within.

**Note:** For this release, use an empty string.

**Return Value**

Returns:

- 0—if successful
- Error number—if unsuccessful

**Access**

The caller must have Essbase Administrator access to the database.

**Example**

```c
void RestoreDB()
{
```
ESS_FUNC_M        sts = ESS_STS_NOERR;
ESS_STR_T            AppName = "Backup";
ESS_STR_T            DbName = "Basic";
ESS_STR_T            BackupFileName = "F:\testArea\ArchiveAndRestore\TempBackup.arc";
ESS_STR_T            optionsFileName = "";
ESS_BOOL_T        bOverWrite;
ESS_BOOL_T        bForceDiffName;
ESS_USHORT_T        count;
ESS_PDISKVOLUME_REPLACE_T replaceVol;

printf("\nArchive DB:\n");
bOverWrite = ESS_TRUE;
sts = EssArchiveDatabase(hCtx, AppName, DbName,
                        BackupFileName, optionsFileName, bOverWrite);

printf("EssArchiveDatabase sts: %ld\n",sts);
sts = EssUnloadApplication(hCtx, AppName);
printf("\nEssUnloadApplication sts: %ld\n",sts);

printf("\nCase with no volume replacement:\n");
bForceDiffName = ESS_FALSE;
count = 0;
replaceVol = ESS_NULL;
sts = EssRestoreDatabase (hCtx, AppName, DbName,
                        BackupFileName, bForceDiffName, count, replaceVol);

printf("EssRestoreDatabase sts: %ld\n",sts);

printf("\nCase with a replacement volume (index and page files to a different
volume):\n");
bForceDiffName = ESS_FALSE;
count = 1;
if (count) {
    sts = EssAlloc(hInst, count * sizeof(ESS_DISKVOLUME_REPLACE_T),
                   (ESS_PPVOID_T)&replaceVol);
    memset(replaceVol, 0, count * sizeof(ESS_DISKVOLUME_REPLACE_T));
}
strcpy(replaceVol->szPartition_Src, "C");
strcpy(replaceVol->szPartition_Dest, "F");

sts = EssUnloadApplication(hCtx, AppName);
printf("\nEssUnloadApplication sts: %ld\n",sts);

sts = EssRestoreDatabase (hCtx, AppName, DbName,
                        BackupFileName, bForceDiffName, count, replaceVol);
printf("EssRestoreDatabase sts: %ld\n",sts);
if (replaceVol)
    EssFree(hInst, replaceVol);
EssArchiveEnd

Restores the server to "read-write" mode after archiving is complete.

Syntax

```c
ESS_FUNC_M EssArchiveEnd (hCtx, AppName, DbName);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>AppName</td>
<td>ESS_STR_T</td>
<td>Name of archived application.</td>
</tr>
<tr>
<td>DbName</td>
<td>ESS_STR_T</td>
<td>Name of archived database.</td>
</tr>
</tbody>
</table>

Notes

- After calling `EssArchiveBegin()`, a call to `EssArchiveEnd()` is required to restore Read-Write mode.

Return Value

None.

Access

The caller must have at least read access (ESS_PRIV_READ) to the database, and must select it as the active database using `EssSetActive()`.

Example

```c
ESS_FUNC_M
ESS_ArchiveEnd(ESS_HCTX_T hCtx)
{
    ESS_FUNC_M      sts = ESS_STS_NOERR;
    ESS_STR_T    AppName;
    ESS_STR_T    DbName;
    AppName  = "Sample";
    DbName   = "Basic";

    /* End Archive */
    sts = EssArchiveEnd(hCtx, AppName, DbName);
    return (sts);
}
```

See Also

- `EssArchiveBegin`
- `EssRestore`
**EssAsyncBuildDim**

Issues an asynchronous dimension build request.

If you use asynchronous data loads and dimension builds, you can query for the following information during the process:

- The state of dimension build/data load process: whether it is in progress, in the final stages, or completed
- The stage of the dimension build/data load process: whether opening the data source, reading the outline, building dimensions, verifying an outline, or writing an outline
- The number of data records processed and rejected so far
- The name and location of the error file
- The data records processed and rejected so far

**Syntax**

```c
ESS_FUNC_M EssAsyncBuildDim(hCtx, RulesObj, DataObj, MbrUser, bOverwrite, usBuildOption, szTmpOtlFile)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>Essbase API context handle.</td>
</tr>
<tr>
<td>RulesObj</td>
<td>ESS_POBJDEF_T</td>
<td>Pointer to rules file object definition structure.</td>
</tr>
<tr>
<td>DataObj</td>
<td>ESS_POBJDEF_T</td>
<td>Pointer to data file object definition structure.</td>
</tr>
<tr>
<td>MbrUser</td>
<td>ESS_PMBRUSER_T</td>
<td>SQL user structure (if data source is SQL database). A NULL SQL user structure indicates a non SQL data source.</td>
</tr>
<tr>
<td>bOverwrite</td>
<td>ESS_BOOL_T</td>
<td>Indicates overwrite or append error message to the error file.</td>
</tr>
<tr>
<td>usBuildOption</td>
<td>ESS_USHORT_T</td>
<td>Valid values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_INCDIMBUILD_BUILD</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Build members only.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_INCDIMBUILD_VERIFY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Build members and verify the outline.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_INCDIMBUILD_SAVEOTL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Build members and save the outline to a temp outline file.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_INCDIMBUILD_ALL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Build members, verify the outline, and restructure.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_INCDIMBUILD_ABORT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Abort the build process.</td>
</tr>
<tr>
<td>szTmpOtlFile</td>
<td>ESS_STR_T</td>
<td>The temporary outline file name. No extension or path is needed. Essbase creates a temporary outline file in the app/\db directory, with an extension of .otb, if the resulting outline in this round of dimension build has outline verification errors.</td>
</tr>
</tbody>
</table>

224  C Main API Functions
Notes
This function returns an error if the data object is located on the client. The network connection between client and server remains active even if an error is returned.

You must call `EssCloseAsyncProc` to close the connection; otherwise, the server request handler blocks further requests from the same login session.

Return Value
Returns zero if successful; error code if unsuccessful.

Example

```c
void ESS_AsyncBuildDim()
{
    ESS_STS_T sts = 0;
    ESS_OBJDEF_T Rules;
    ESS_OBJDEF_T Data;
    ESS_MBRUSER_T pMbrUser;
    ESS_BOOL_T bOverwrite;
    ESS_USHORT_T usBuildOption;
    ESS_STR_T szTmpOtlFile;
    ESS_STR_T bldDimErrFile;
    ESS_STR_T asyncProcErrLog;
    ESS_BLDDL_STATE_T procState;
    ESS_BOOL_T errFileOverWrite;

    szAppName = "Sample";
    szDbName = "Basic";
    ESS_setActive();
    AddMember("800");

    sts = EssBeginIncrementalBuildDim(hCtx);
    printf("EssBeginIncrementalBuildDim sts: %ld\n",sts);

    memset(&Rules,0,sizeof(ESS_OBJDEF_T));
    memset(&Data,0,sizeof(ESS_OBJDEF_T));
    Rules.hCtx = hCtx;
    Rules.FileName = "apgeibl";
    Rules.AppName = szAppName;
    Rules.DbName = szDbName;
    Rules.ObjType = ESS_OBJTYPE_RULES;
    Data.hCtx = hCtx;
    Data.AppName = szAppName;
    Data.DbName = szDbName;
    Data.ObjType = ESS_OBJTYPE_TEXT;
    Data.FileName = "apgeibl1";

    pMbrUser = ESS_NULL;
    bOverwrite = ESS_TRUE;
    usBuildOption = ESS_INCDIMBUILD_BUILD;
    szTmpOtlFile = "asyncBldTmp";
    sts = EssAsyncBuildDim(hCtx, &Rules, &Data, pMbrUser, bOverwrite, usBuildOption,
                           szTmpOtlFile);
    printf("EssAsyncBuildDim sts: %ld\n",sts);
}```
sts = EssGetAsyncProcLog (hCtx, "\AsyncProc.log", ESS_TRUE);
printf("EssGetAsyncProcLog sts: %ld\n",sts);

sts = EssGetAsyncProcState(hCtx, &procState);
printf("EssGetAsyncProcState sts: %ld\n",sts);
if(!sts)
{
    do
    {
        DisplayProcessStateInfo(procState);
        if(procState.ilProcessStatus)
        {
            sts = EssCancelAsyncProc(hCtx, asyncProcErrLog, errFileOverWrite);
            printf("EssCancelAsyncProc sts: %ld\n",sts);
        }
        else
        {
            sts = EssGetAsyncProcState(hCtx, &procState);
            printf("EssGetAsyncProcState sts: %ld\n",sts);
        }
    }
    while(procState.usProcessState != ESS_BLDDL_STATE_DONE);

    if(!procState.ilProcessStatus)
    {
        sts = EssCloseAsyncProc(hCtx, &procState);
        printf("EssCloseAsyncProc sts: %ld\n",sts);
    }
}

bldDimErrFile = "F:\testArea\mainapi\BldDim.err";
st = EssEndIncrementalBuildDim(hCtx, ESS_DOR_ALLDATA, szTmpOtlFile, bldDimErrFile,
ESS_FALSE);
printf("EssEndIncrementalBuildDim sts: %ld\n",sts);

See Also

- EssAsyncImport
- EssGetAsyncProcLog
- EssGetAsyncProcState
- EssCancelAsyncProc
- EssCloseAsyncProc

**EssAsyncImport**

Issues an asynchronous data load request.

If you use asynchronous data loads and dimension builds, you can query for the following information during the process:

- The state of dimension build/data load process: whether it is in progress, in the final stages, or completed
The stage of the dimension build/data load process: whether opening the data source, reading the outline, building dimensions, verifying an outline, or writing an outline

The number of data records processed and rejected so far

The name and location of the error file

The data records processed and rejected so far

Syntax

```
ESS_FUNC_M EssAsyncImport (hCtx, pRules, pData, pMbrUser, abortOnError);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>pRules</td>
<td>ESS_POBJDEF_T</td>
<td>Pointer to the rules file object definition structure.</td>
</tr>
<tr>
<td>pData</td>
<td>ESS_POBJDEF_T</td>
<td>Pointer to the data file object definition structure.</td>
</tr>
<tr>
<td>pMbrUser</td>
<td>ESS_PMBRUSER_T</td>
<td>Pointer to the SQL user structure (if data source is a SQL database). A NULL SQL user structure indicates a non SQL data source.</td>
</tr>
<tr>
<td>abortOnError</td>
<td>ESS_USHORT_T</td>
<td>If TRUE, import stops on the first error. Otherwise, it continues.</td>
</tr>
</tbody>
</table>

Notes

This function returns an error if the data object is located on the client. The network connection between client and server remains active even if an error is returned.

You must call `EssCloseAsyncProc` to close the connection; otherwise, the server request handler blocks further requests from the same login session.

Return Value

Returns zero if successful. Otherwise, returns an error code.

Access

This function requires the caller to have database designer privilege for the specified database (ESS_PRIV_DBDESIGN).

Example

```
ESS_AsyncImport();
{
    ESS_STS_T sts = ESS_STS_NOERR;
    ESS_SHORT_T isAbortOnError;
    ESS_OBJDEF_T Rules;
    ESS_OBJDEF_T Data;
    ESS_PMBRUSER_T pUser;
    ESS_STR_T errorName;
    ESS_BLDDL_STATE_T procState;
    ESS_BOOL_T errFileOverWrite;

    szAppName = "Sample";
    szDbName = "Basic";
```
ESS_SetActive();

memset(&Rules,0,sizeof(ESS_OBJDEF_T));
memset(&Data,0,sizeof(ESS_OBJDEF_T));
Rules.hCtx = hCtx;
Rules.FileName = "Act1";
Rules.AppName = szAppName;
Rules.DbName = szDbName;
Rules.ObjType = ESS_OBJTYPE_RULES;
Data.hCtx = hCtx;
Data.FileName = "Act1";
Data.AppName = szAppName;
Data.DbName = szDbName;
Data.ObjType = ESS_OBJTYPE_TEXT;

errorName = ".\asyncProcess.err";
errFileOverWrite = ESS_TRUE;
isAbortOnError = ESS_TRUE;
pUser = ESS_NULL; /* NULL equals a non-SQL data source */
sts = EssAsyncImport(hCtx, &Rules, &Data, pUser, isAbortOnError);
printf("EssAsyncImport sts: %ld\n",sts);

sts = EssGetAsyncProcState(hCtx, &procState);
printf("EssGetAsyncProcState sts: %ld\n",sts);
if(!sts)
{
    do
    {
        DisplyProcesStateInfo(procState);
        if(procState.ilProcessStatus)
        {
            sts = EssCancelAsyncProc(hCtx, errorName, errFileOverWrite);
            printf("EssCancelAsyncProc sts: %ld\n",sts);
        }
        else
        {
            sts = EssGetAsyncProcState(hCtx, &procState);
            printf("EssGetAsyncProcState sts: %ld\n",sts);
        }
    }while(procState.usProcessState != ESS_BLD_DC_STATE_DONE);

    if(!procState.ilProcessStatus)
    {
        sts = EssCloseAsyncProc(hCtx, &procState);
        printf("EssCloseAsyncProc sts: %ld\n",sts);
    }
}
}

See Also

- EssAsyncBuildDim
- EssGetAsyncProcLog
- EssGetAsyncProcState
- EssCancelAsyncProc
- EssCloseAsyncProc

228 C Main API Functions
EssAsyncImportASO

Issues an asynchronous data load request on an aggregate storage database.

If you use asynchronous data loads and dimension builds, you can query for the following information during the process:

- The state of dimension build/data load process: whether it is in progress, in the final stages, or completed
- The stage of the dimension build/data load process: whether opening the data source, reading the outline, building dimensions, verifying an outline, or writing an outline
- The number of data records processed and rejected so far
- The name and location of the error file
- The data records processed and rejected so far

Syntax

```
ESS_FUNC_M EssAsyncImportASO (hCtx, pRules, pData, pUser, usAbortOnError, ulBufferId);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>pRules</td>
<td>ESS_POBJDEF_T</td>
<td>Pointer to the rules file object definition structure.</td>
</tr>
<tr>
<td>pData</td>
<td>ESS_POBJDEF_T</td>
<td>Pointer to the data file object definition structure.</td>
</tr>
<tr>
<td>pUser</td>
<td>ESS_PMBRUSER_T</td>
<td>Pointer to the SQL user structure (if data source is a SQL database). A NULL SQL user structure indicates a non SQL data source.</td>
</tr>
<tr>
<td>usAbortOnError</td>
<td>ESS_USHORT_T</td>
<td>If TRUE, import stops on the first error. Otherwise, it continues.</td>
</tr>
<tr>
<td>ulBufferID</td>
<td>ESS_ULONG_T</td>
<td>ID of a data load buffer (a number between 1 and 999,999). To destroy a buffer before a data load is complete, you must use the same ulBufferId number that was used to initialize the buffer.</td>
</tr>
</tbody>
</table>

Notes

This function returns an error if the data object is located on the client. The network connection between client and server remains active even if an error is returned.

You must call EssCloseAsyncProc to close the connection; otherwise, the server request handler blocks further requests from the same login session.

Return Value

Returns zero if successful. Otherwise, returns an error code.

Access

This function requires the caller to have database designer privilege for the specified database (ESS_PRIV_DBDESIGN).
```c
void ESS_AsyncImportASO()
{
    ESS_STS_T sts = ESS_STS_NOERR;
    ESS_SHORT_T isAbortOnError;
    ESS_OBJDEF_T Rules;
    ESS_OBJDEF_T Data;
    ESS_PMBRERR_T pMbrErr = NULL;
    ESS_PMBRUSER_T pMbrUser = NULL;
    ESS_ULONG_T ulBufferId;
    ESS_ULONG_T ulDuplicateAggregationMethod;
    ESS_ULONG_T ulOptionsFlags;
    ESS_ULONG_T ulSize;
    ESS_ULONG_T ulBufferCnt;
    ESS_ULONG_T ulCommitType;
    ESS_ULONG_T ulActionType;
    ESS_ULONG_T ulOptions;
    ESS_ULONG_T ulBufferIdAry[1];
    ESS_STR_T errorName;
    ESS_BLDDL_STATE_T procState;
    ESS_BOOL_T errFileOverWrite;

    szAppName = "ASOSamp";
    szDbName = "Sample";
    ESS_SetActive();

    errorName = ".\asyncProcess.err";
    errFileOverWrite = ESS_TRUE;
    ulDuplicateAggregationMethod = ESS_ASO_DATA_LOAD_BUFFER_DUPLICATES_ADD;
    ulOptionsFlags = ESS_ASO_DATA_LOAD_BUFFER_IGNORE_MISSING_VALUES;
    ulSize = 1;
    ulBufferId = 100;
    sts = EssLoadBufferInit(hCtx, szAppName, szDbName, ulBufferId,
                           ulDuplicateAggregationMethod,
                           ulOptionsFlags, ulSize);
    printf("EssLoadBufferInit sts: %ld\n", sts);
    if(!sts)
    {
        /* Server object */
        Rules.hCtx     = hCtx;
        Rules.AppName  = szAppName;
        Rules.DbName   = szDbName;
        Rules.ObjType  = ESS_OBJTYPE_RULES;
        Rules.FileName = "Dataload";
        Data.hCtx      = hCtx;
        Data.AppName   = szAppName;
        Data.DbName    = szDbName;
        Data.ObjType   = ESS_OBJTYPE_TEXT;
        Data.FileName  = "Dataload";

        isAbortOnError = ESS_TRUE;
        sts = EssAsyncImportASO (hCtx, &Rules, &Data, pMbrUser, isAbortOnError,
                                 ulBufferId);
        printf("EssAsyncImportASO sts: %ld\n",sts);
        if(!sts)
        {
```
sts = EssGetAsyncProcState(hCtx, &procState);
printf("EssGetAsyncProcState sts: %ld\n",sts);
if(!sts)
{
    do
    {
        DisplayProcStateInfo(procState);
        if(procState.ilProcessStatus)
        {
            sts = EssCancelAsyncProc(hCtx, errorName, errFileOverWrite);
            printf("EssCancelAsyncProc sts: %ld\n",sts);
        }
        else
        {
            sts = EssGetAsyncProcState(hCtx, &procState);
            printf("EssGetAsyncProcState sts: %ld\n",sts);
        }
    }while(procState.usProcessState != ESS_BLDDL_STATE_DONE);
}
sts = EssCloseAsyncProc(hCtx, &procState);
printf("EssCloseAsyncProc sts: %ld\n",sts);

ulBufferCnt = 1;
ulBufferIdAry[0] = ulBufferId;
ulCommitType = ESS_ASO_DATA_LOAD_BUFFER_STORE_DATA;
ulActionType = ESS_ASO_DATA_LOAD_BUFFER_COMMIT;
printf("\nIncrement to main slice:\n");
ulOptions = ESS_ASO_DATA_LOAD_INCR_TO_MAIN_SLICE;
sts = EssLoadBufferTerm(hCtx, szAppName, szDbName, ulBufferCnt,
ulBufferIdAry, ulCommitType, ulActionType, ulOptions);
printf("EssLoadBufferTerm sts: %ld\n",sts);

See Also

- EssAsyncBuildDim
- EssAsyncImport
- EssGetAsyncProcLog
- EssGetAsyncProcState
- EssCancelAsyncProc
- EssCloseAsyncProc

EssAutoLogin

Displays a dialog box that allows the user to log in to an Essbase Server, and optionally select an active application and database.

Syntax

ESS_FUNC_M EssAutoLogin (hInstance, Server, UserName,
Password,AppName, DbName, Options, pAccess, phCtx);
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>ESS_HINST_T</td>
<td>API instance handle</td>
</tr>
<tr>
<td>Server</td>
<td>ESS_SVRNAME_T</td>
<td>Network server name string</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The server name can be expressed as a URL representing the APS servlet endpoint with the Essbase failover cluster name; for example:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><a href="http://myhost:13080/aps/Essbase?clustername=Essbase-Cluster1">http://myhost:13080/aps/Essbase?clustername=Essbase-Cluster1</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td>For secure mode (SSL), the URL syntax is</td>
</tr>
<tr>
<td></td>
<td></td>
<td>http[s]://host:port/aps/Essbase?ClusterName=logicalName&amp;SecureMODE=yesORno</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For example,</td>
</tr>
<tr>
<td>UserName</td>
<td>ESS_USERNAME_T</td>
<td>User name string</td>
</tr>
<tr>
<td>Password</td>
<td>ESS_PASSWORD_T</td>
<td>Password string</td>
</tr>
<tr>
<td>AppName</td>
<td>ESS_APPNAME_T</td>
<td>Application name</td>
</tr>
<tr>
<td>DbName</td>
<td>ESS_DBNAME_T</td>
<td>Database name</td>
</tr>
<tr>
<td>Options</td>
<td>ESS_USHORT_T</td>
<td>Options flag. Values:</td>
</tr>
<tr>
<td></td>
<td>AUTO_NODIALOG</td>
<td>Attempts to log the user in without displaying the dialog, using the default settings (from the above arguments).</td>
</tr>
<tr>
<td></td>
<td>AUTO_NOSELECT</td>
<td>Allows the user to log in without selecting an application and database (lower part of the dialog is not displayed).</td>
</tr>
<tr>
<td></td>
<td>AUTO_NODIALOG</td>
<td>AUTO_NOSELECT: AUTO_DEFAULT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>You can use both AUTO_NODIALOG and AUTO_NOSELECT with an OR operator (</td>
</tr>
<tr>
<td>pAccess</td>
<td>ESS_PACCESS_T</td>
<td>Address of variable to receive database access level.</td>
</tr>
<tr>
<td>phCtx</td>
<td>ESS_PHCTX_T</td>
<td>Address of variable to receive Essbase context handle. Set to ESS_INVALID_HCTX unless you are reusing an existing (valid) context handle to log in again.</td>
</tr>
</tbody>
</table>

**Notes**

- The dialog box is automatically managed by the function, and provides features in the login dialog to change the user password, display the database note message, etc., and so provides a standardized and powerful login screen for all applications using the API.
- Use this function instead of the EssLogin function if you are programming in a Windows environment.
- The function should be called after executing a successful call to EssInit, and prior to making any other API calls which require a context handle argument.
This function is supported only in Windows environments. It is not supported in UNIX environments.

The string arguments Server, UserName, Password, AppName or DbName may optionally be NULL. If any of them are not NULL, the buffers they point to are updated when the function returns the actual values selected by the user from the dialog box. If any of the passed in arguments point to valid strings, they will be used as the default displayed values in the dialog. The buffers for these arguments must be large enough to contain any possible return value, not just the values passed in.

If the login is successful, the server and user names are automatically stored (in the file ESSBASE.INI) and are used as the defaults the next time this function is called (unless those arguments are specified in subsequent calls). The names of all servers which have been successfully connected to are also stored and displayed.

The auto login dialog box is a child window of the current active window (the window that has the focus). Therefore avoid destroying the active window or changing focus while the auto login dialog is displayed.

This function returns a value of ESS_STS_CANCEL if the user presses the Cancel button or the Esc key in the dialog box.

In Windows environments, if the end user clicks the Help button, the Essbase System Login help topic shipped with the Oracle Essbase Spreadsheet Add-in User’s Guide online help is opened. You can redirect the Help button to point to a different help file by specifying a different help file name in the ESS_INIT_T structure.

**Return Value**

If successful, returns an Essbase context handle in phCtx, which can be passed as an argument in subsequent calls to other API functions. Also returns the user’s access level to the selected application and database (if selected) in pAccess.

**Access**

Before calling this function, you must first initialize the API and obtain a valid instance handle by calling the EssInit function.

**See Also**

- EssInit
- EssListDatabases
- EssLogin
- EssLogout
- EssSetActive

**EssBeginCalc**

Starts sending a calc script and optionally executes it against the active database.

**Syntax**

```c
ESS_FUNC_M EssBeginCalc (hCtx, Calculate);
```
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>Calculate</td>
<td>ESS_BOOL_T</td>
<td>Controls calculation of the calc script. If TRUE, the calc script is executed.</td>
</tr>
</tbody>
</table>

**Notes**

- This call must be followed by successive calls to **EssSendString()** to send the calc script, and finally by a call to **EssEndCalc()**.
- The Calc Script must be less than 64 KB long in total.
- The calculation can either be initiated, or the calc script can just be verified and any errors returned.
- If the calc script is successfully sent and the calculation is started, it will continue on the server as an asynchronous process after the return from this call. After calling **EssEndCalc()**, the caller must check at regular intervals to see if the process has completed by calling **EssGetProcessState()** until it returns ESS_STATE_DONE.
- If the Calculate flag is set to FALSE, the database merely performs a syntax check of the calc script.
- Unicode clients using the C Main API to communicate with Unicode-enabled Essbase applications must send the UTF-8 encoded Unicode byte order mark (BOM) in the text stream immediately after calling this function. For an example, see “Specifying the Byte Order Encoding” on page 75.

**Return Value**

None.

**Access**

This function requires the caller to have calc privilege (ESS_PRIV_CALC) to the active database.

**Example**

```c
ESS_FUNC_M
ESS_Calc (ESS_HCTX_T hCtx)
{
    ESS_FUNC_M st = ESS_STS_NOERR;
    ESS_STR_T Script;
    ESS_PROCSTATE_T pState;
    Script = "CALC ALL;";

    st = EssBeginCalc (hCtx, ESS_TRUE);
    if (!st)
        st = EssSendString (hCtx, Script);
    if (!st)
        st = EssEndCalc (hCtx);
    if (!st)
    {
        st = EssGetProcessState (hCtx, &pState);
        while(!st && (pState.State != ESS_STATE_DONE))
            st = EssGetProcessState (hCtx, &pState);
    }
```

234  C Main API Functions
See Also
- EssCalc
- EssCalcFile
- EssDefaultCalc
- EssEndCalc
- EssGetDefaultCalc
- EssGetProcessState
- EssSendString
- EssSetDefaultCalc

**EssBeginDataload**

Starts sending an update specification to the active database, and can unlock any data blocks locked for update. The update data can either be stored in the database, or just verified and any errors returned.

**Syntax**

```c
ESS_STS_T EssBeginDataload (hCtx, Store, Unlock, abortOnError, pRules);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx;</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>Store;</td>
<td>ESS_BOOL_T</td>
<td>Controls storage of data. If TRUE, data is stored in the server; if FALSE, no data is stored.</td>
</tr>
<tr>
<td>Unlock;</td>
<td>ESS_BOOL_T</td>
<td>Controls unlocking of data blocks. If TRUE, all relevant blocks which are locked will be unlocked (after data is stored, if necessary). If FALSE, no blocks are unlocked.</td>
</tr>
<tr>
<td>abortOnError;</td>
<td>ESS_BOOL_T</td>
<td>If TRUE, data load stops on the first error. Otherwise, data load continues.</td>
</tr>
<tr>
<td>pRules;</td>
<td>“ESS_OBJDEF_T” on page 151</td>
<td>Pointer to the rules file object definition structure.</td>
</tr>
</tbody>
</table>

**Notes**

- **EssBeginDataload()** must be followed by at least one call to **EssSendString()** to send the update specification, and then a call to **EssEndDataload()**.
- Each string passed to **EssSendString()** following **EssBeginDataload()** must be terminated with a carriage return/linefeed character sequence (“\r\n”).
- If both the Store and Unlock flags are set to FALSE, the database merely performs a syntax check of the update specification.
- Unlike **EssBeginUpdate()**, which ignores input rows (records) after an improper input row, **EssBeginDataload()** processes the remaining input rows, and commits them if appropriate.
EssEndDataload() returns a linked list of errors in “ESS_MBRERR_T” on page 148.

Unicode clients using the C Main API to communicate with Unicode-enabled Essbase applications must send the UTF-8 encoded Unicode byte order mark (BOM) in the text stream immediately after calling this function. For an example, see “Specifying the Byte Order Encoding” on page 75.

Return Value

None.

Access

EssBeginDataload() requires the caller to have write privilege (ESS_PRIV_WRITE) to the active database.

Example

```c
ESS_STS_T     sts = ESS_STS_NOERR;
ESS_BOOL_T    Store;
ESS_BOOL_T    Unlock;
ESS_STR_T     Query1, Query2;
ESS_PMBRERR_T pMbrErr;

Store  = ESS_TRUE;
Unlock = ESS_FALSE;
Query1 = "Year Market Scenario Measures Product 12345";
Query2 = " Jan  East  Scenario Measures Coke 125";

/* Begin Update */
sts = EssBeginDataload (hCtx, Store, Unlock, ESS_FALSE, ESS_NULL);

/* Send update specification */
if(!sts)
    sts = EssSendString(hCtx, Query1);
    sts = EssSendString(hCtx, Query2);

/* End Update */
if(!sts)
    sts = EssEndDataload(hCtx, &pMbrErr);
```

See Also

- EssSendString
- EssEndDataload
- EssBeginUpdate
- EssEndUpdate
- EssUpdate
- EssImport

EssBeginDataloadASO

Starts a data load on an aggregate storage database.
Syntax

```c
ESS_FUNC_M EssBeginDataloadASO (hCtx, Store, Unlock, abortOnError, pRules, ulBufferId);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>Store</td>
<td>ESS_BOOL_T</td>
<td>Controls storage of data. If ESS_TRUE, data is stored in the server; if ESS_FALSE, no data is stored.</td>
</tr>
<tr>
<td>Unlock</td>
<td>ESS_BOOL_T</td>
<td>Not supported for aggregate storage databases. You must always pass ESS_FALSE for this parameter.</td>
</tr>
<tr>
<td>abortOnError</td>
<td>ESS_BOOL_T</td>
<td>ESS_TRUE indicates that the data load will be aborted in case of errors during the process.</td>
</tr>
<tr>
<td>pRules</td>
<td>“ESS_OBJDEF_T”</td>
<td>Pointer to the rules file object definition structure. page 151</td>
</tr>
<tr>
<td>ulBufferId</td>
<td>ESS_UINTEGER_T</td>
<td>ID of a data load buffer. To destroy a buffer before a data load is complete, you must use the same ulBufferId number that was used to initialize the buffer.</td>
</tr>
</tbody>
</table>

Notes

- **EssBeginDataloadASO()** must be followed by at least one call to **EssSendString()** to send the update specification, and then a call to **EssEndDataload()**.
- Each string passed to **EssSendString()** following **EssBeginDataloadASO()** must be terminated with a carriage return/linefeed character sequence ("\r\n").
- If the Store flag is set to FALSE, the database merely performs a syntax check of the update specification.
- Unicode clients using the C Main API to communicate with Unicode-enabled Essbase applications must send the UTF-8 encoded Unicode byte order mark (BOM) in the text stream immediately after calling this function. For an example, see "Specifying the Byte Order Encoding" on page 75.

Return Value

Returns zero if successful; otherwise, returns an error code.

Access

**EssBeginDataloadASO()** requires the caller to have write privilege (ESS_PRIV_WRITE) to the active database.

Example

```c
void TestBeginDataloadASO(ESS_HCTX_T hCtx, ESS_STR_T AppName, ESS_STR_T DbName) 
{
    ESS_STS_T       sts = ESS_STS_NOERR;
    ESS_BOOL_T      Store;
    ESS_BOOL_T      Unlock;
    ESS_BOOL_T      abortOnError;
    ESS_STR_T       loadString;
    ESS_OBJDEF_T    rulesFile;
    ESS_PMBRERR_T   pMbrErr;
```
ESS ULONG_T ulBufferId;
ESS ULONG_T ulDuplicateAggregationMethod;
ESS ULONG_T ulOptionsFlags;
ESS ULONG_T ulSize;
ESS ULONG_T ulBufferCnt;
ESS ULONG_T ulCommitType;
ESS ULONG_T ulActionType;
ESS ULONG_T ulOptions;
ESS ULONG_T ulBufferIdAry[1];

/* EssLoadBufferInit */
ulDuplicateAggregationMethod = ESS_ASO_DATA_LOAD_BUFFER_DUPLICATES_ADD;
ulOptionsFlags = ESS_ASO_DATA_LOAD_BUFFER_IGNORE_MISSING_VALUES;
ulSize = 100;
ulBufferId = 201;
sts = EssLoadBufferInit(hCtx, AppName, DbName, ulBufferId,
ulDuplicateAggregationMethod,
ulOptionsFlags, ulSize);
printf("EssLoadBufferInit sts: %ld\n", sts);

/* EssBeginDataloadASO, EssSendString, EssEndDataload */
Store = ESS_TRUE;
Unlock = ESS_FALSE;
abortOnError = ESS_FALSE;
loadString = "Mar Sale \"Curr Year\" \"Original Price\" \"017589\" \"13668\" Cash \"No Promotion\" \"1 to 13 Years\" \"Under 20,000\" \"Digital Cameras\" 111;"
sts = EssBeginDataloadASO (hCtx, Store, Unlock, abortOnError, ESS_NULL,
ulBufferId);
printf("EssBeginDataloadASO sts: %ld\n",sts);
sts = EssSendString(hCtx, loadString);
printf("EssSendString sts: %ld\n",sts);
sts = EssEndDataload(hCtx, &pMbrErr);
printf("EssEndDataload sts: %ld\n",sts);

/* EssLoadBufferTerm */
ulBufferCnt = 1;
ulBufferIdAry[0] = ulBufferId;
ulCommitType = ESS_ASO_DATA_LOAD_BUFFER_STORE_DATA;
ulActionType = ESS_ASO_DATA_LOAD_BUFFER_COMMIT;
printf("\Commit data to main slice and destroy buffer:\n");
ulOptions = ESS_ASO_DATA_LOAD_INCR_TO_MAIN_SLICE;
sts = EssLoadBufferTerm(hCtx, AppName, DbName, ulBufferCnt, ulBufferIdAry,
ulCommitType,
ulActionType, ulOptions);
printf("EssLoadBufferTerm sts: %ld\n",sts);
}

See Also
- EssLoadBufferInit
- EssSendString
- EssEndDataload
- EssLoadBufferTerm
- EssImportASO
- EssUpdateFileASO
EssBeginDataloadEx

Starts sending an update specification to the active database, and can unlock any data blocks locked for update. The update data can either be stored in the database, or just verified and any errors returned.

Syntax

```c
ESS_STS_T EssBeginDataloadEx (hCtx, Store, Unlock, abortOnError, pRules, fullMbrNames);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>Store</td>
<td>ESS_BOOL_T</td>
<td>Controls storage of data.</td>
</tr>
<tr>
<td>Unlock</td>
<td>ESS_BOOL_T</td>
<td>Controls unlocking of data blocks. If TRUE, all relevant blocks which are locked will be unlocked (after data is stored, if necessary). If FALSE, no blocks are unlocked.</td>
</tr>
<tr>
<td>abortOnError</td>
<td>ESS_BOOL_T</td>
<td>If TRUE, data load stops on the first error. Otherwise, data load continues.</td>
</tr>
<tr>
<td>pRules</td>
<td>“ESS_OBJDEF_T” on page 151</td>
<td>Pointer to the rules file object definition structure.</td>
</tr>
<tr>
<td>fullMbrNames</td>
<td>ESS_BOOL_T</td>
<td>If TRUE, the error log prints full member names for the entire record.</td>
</tr>
</tbody>
</table>

Notes

- EssBeginDataloadEx must be followed by at least one call to EssSendString to send the update specification, and then a call to EssEndDataloadEx.
- Each string passed to EssSendString following EssBeginDataloadEx must be terminated with a carriage return/linefeed character sequence (“\r\n”).
- If both the Store and Unlock flags are set to FALSE, the database merely performs a syntax check of the update specification.
- Unlike EssBeginUpdate, which ignores input rows (records) after an improper input row, EssBeginDataloadEx processes the remaining input rows, and commits them if appropriate.
- EssEndDataload returns a linked list of errors in “ESS_MBRERR_T” on page 148.
- Unicode clients using the C Main API to communicate with Unicode-enabled Essbase applications must send the UTF-8 encoded Unicode byte order mark (BOM) in the text.
stream immediately after calling this function. For an example, see “Specifying the Byte Order Encoding” on page 75.

**Return Value**

None.

**Access**

EssBeginDataloadEx() requires the caller to have write privilege (ESS_PRIV_WRITE) to the active database.

**Example**

```c
ESS_STS_T     sts = ESS_STS_NOERR;
ESS_BOOL_T    Store;
ESS_BOOL_T    Unlock;
ESS_STR_T     Query1, Query2;
ESS_PMBRERR_T pMbrErr;

Store       = ESS_TRUE;
Unlock      = ESS_FALSE;
Query1      = "Year Market Scenario Measures Product 12345";
Query2      = " Jan  East  Scenario Measures Coke  125";

/* Begin Update */
sts = EssBeginDataloadEx(hCtx, Store, Unlock, ESS_FALSE, ESS_NULL, ESS_TRUE);

/* Send update specification */
if(!sts)
    sts = EssSendString(hCtx, Query1);
    sts = EssSendString(hCtx, Query2);

/* End Update */
if(!sts)
    sts = EssEndDataload(hCtx, &pMbrErr);
```

**See Also**

- EssSendString
- EssEndDataload
- EssBeginUpdate
- EssEndUpdate
- EssUpdate
- EssImport

**EssBeginIncrementalBuildDim**

It starts the process to build members on the active database. Internally, Essbase server opens the outline of the active database and keeps it open and ready for the next steps of dimension build.

**Syntax**

```c
ESS_FUNC_M EssBeginIncrementalBuildDim(hCtx);
```
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>Essbase API context handle.</td>
</tr>
</tbody>
</table>

**Return Value**

Returns zero if successful; error code if unsuccessful.

**Example**

```c
ESS_FUNC_M
ESS_IncBuildDim( ESS_HCTX_T hCtx)
{
    ESS_STS_T     sts = 0;
    ESS_OBJDEF_T  RulesObj;
    ESS_OBJDEF_T  DataObj;
    ESS_STR_T     ErrorName;
    ESS_APPNAME_T     appname;
    ESS_DBNAME_T      dbname;

    memset(&RulesObj,0,sizeof(ESS_OBJDEF_T));
    memset(&DataObj,0,sizeof(ESS_OBJDEF_T));
    strcpy(appname, "sample");
    strcpy(dbname,"basic");

    RulesObj.hCtx     = hCtx;
    RulesObj.FileName = "genref";
    RulesObj.AppName  = appname;
    RulesObj.DbName   = dbname;
    RulesObj.ObjType  = ESS_OBJTYPE_RULES;

    DataObj.hCtx      = hCtx;
    DataObj.FileName  = "genref";
    DataObj.AppName   = appname;
    DataObj.DbName    = dbname;
    DataObj.ObjType   = ESS_OBJTYPE_TEXT;

    ErrorName         = "builddim.err";

    sts = EssBeginIncrementalBuildDim(hCtx);
    if (!sts)
        sts = EssIncrementalBuildDim(hCtx,&RulesObj,&DataObj,NULL,ErrorName,true,ESS_INCDIMBUILD_BUILD,NULL);
    if (!sts)
        sts = EssIncrementalBuildDim(hCtx,&RulesObj,&DataObj,NULL,ErrorName,true,ESS_INCDIMBUILD_VERIFY
        ,NULL);
    if (!sts)
        sts = EssIncrementalBuildDim(hCtx,&RulesObj,&DataObj,NULL,ErrorName,true,ESS_INCDIMBUILD_SAVEOT
        L,"tmpotl");

    sts = EssBeginStreamBuildDim(hCtx, &RulesObj,ESS_INCDIMBUILD_BUILD,"tmpotl");
    if (!sts)
```
sts = EssSendString(hCtx, "600    600-20    600-20-20\n"); 
if (!sts)
    sts = EssSendString(hCtx, "600    600-20    600-20-30\n"); 
if (!sts)
    sts = EssSendString(hCtx, "600    600-40    600-40-20\n"); 
sts = EssEndStreamBuildDim(hCtx, ErrorName, false); 

sts = EssEndIncrementalBuildDim(hCtx, ESS_DOR_ALLDATA, "tmpotl", ErrorName, false); 
return sts;
}

See Also

- EssIncrementalBuildDim
- EssBeginIncrementalBuildDim
- EssBeginStreamBuildDim
- EssEndIncrementalBuildDim
- EssEndStreamBuildDim

EssBeginReport

Starts sending a report specification to the active database. This call must be followed by successive calls to EssSendString() to send the report specification, and finally by a call to EssEndReport(). The report data can either be output, or the report specification can just be verified and any errors returned. Also, the corresponding data blocks in the database can optionally be locked by this call (lock for update).

Syntax

**ESS_FUNC_M** EssBeginReport (hCtx, Output, Lock);

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>Output</td>
<td>ESS_BOOL_T</td>
<td>Controls output of data. If TRUE, data is output from the server, according to the specified report. If FALSE, no data is output.</td>
</tr>
<tr>
<td>Lock</td>
<td>ESS_BOOL_T</td>
<td>Controls block locking. If TRUE, all blocks which are accessed by the report specification are locked for update. If FALSE, no blocks are locked.</td>
</tr>
</tbody>
</table>

Notes

- This function must be followed by at least one call to EssSendString(), followed by a call to EssEndReport().
- If this function causes data to be output (Output flag is TRUE), the returned data can be read by calling EssGetString().
- If this function causes blocks to be locked (Lock flag is TRUE), the caller is responsible for unlocking the locked blocks (e.g. by calling EssUpdate() with the Unlock flag set to TRUE).
- If both the Output and Lock flags are set to FALSE, the database merely performs a syntax check of the report specification.
Unicode clients using the C Main API to communicate with Unicode-enabled Essbase applications must send the UTF-8 encoded Unicode byte order mark (BOM) in the text stream immediately after calling this function. For an example, see “Specifying the Byte Order Encoding” on page 75.

**Return Value**

None.

**Access**

This function requires the caller to have read privilege (ESS_PRIV_READ) to one or more members in the active database.

**Example**

```c
ESS_FUNC_M
ESS_Report (ESS_HCTX_T   hCtx,
             ESS_HINST_T  hInst)
{
    ESS_FUNC_M       sts     = ESS_STS_NOERR;
    ESS_STR_T     rString = NULL;
    sts = EssBeginReport (hCtx, ESS_TRUE, ESS_FALSE);
    if (!sts)
        sts = EssSendString (hCtx, "<Desc Year !");
    if (!sts)
        sts = EssEndReport (hCtx);
    /***************
    * Get report *
    ***************
    if (!sts)
        sts = EssGetString (hCtx, &rString);
    while (!sts) && (rString != NULL))
    {  
        printf ("%s", rString);
        EssFree (hInst, rString);
        sts = EssGetString (hCtx, &rString);
    }
    printf ("\n\n");
    return(sts);
}
```

**See Also**

- EssBeginUpdate
- EssEndReport
- EssGetString
- EssReport
- EssReportFile
- EssSendString
**EssBeginStreamBuildDim**

Starts the dimension build process.

This function must be called before EssEndStreamBuildDim(). After calling EssBeginStreamBuildDim(), call EssSendString() to send source records to Essbase server.

**Syntax**

`ESS_FUNC_M EssBeginStreamBuildDim (hCtx, RulesObj, usBuildOption, szTmpOtlFilename)`

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>RulesObj</td>
<td>ESS_POBJDEF_T</td>
<td>Pointer to rules file object definition structure.</td>
</tr>
<tr>
<td>usBuildOption</td>
<td>ESS_USHORT_T</td>
<td>Valid values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_INCDIMBUILD_BUILD</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Build members only.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_INCDIMBUILD_VERIFY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Build members and verify the outline.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_INCDIMBUILD_SAVEOTL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Build members and save the outline to a temp outline file.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_INCDIMBUILD_ALL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Build members, verify the outline, and restructure.</td>
</tr>
<tr>
<td>szTmpOtlFilename</td>
<td>ESS_STR_T</td>
<td>The temp outline file name. Essbase creates a temporary outline file with</td>
</tr>
<tr>
<td></td>
<td></td>
<td>extension &quot;otb&quot; if the resulting outline in this round of dimension build</td>
</tr>
<tr>
<td></td>
<td></td>
<td>has outline verification errors.</td>
</tr>
</tbody>
</table>

**Notes**

Unicode clients using the C Main API to communicate with Unicode-enabled Essbase applications must send the UTF-8 encoded Unicode byte order mark (BOM) in the text stream immediately after calling this function. For an example, see "Specifying the Byte Order Encoding" on page 75.

**Return Value**

Returns zero if successful; error code if unsuccessful.

**Example**

```c
ESS_FUNC_M
ESS_IncBuildDim( ESS_HCTX_T hCtx) {
    ESS_STS_T     sts = 0;
    ESS_OBJDEF_T  RulesObj;
    ESS_OBJDEF_T  DataObj;
    ESS_STR_T     ErrorName;
    ESS_APPNAME_T appname;
    ESS_DBNAME_T  dbname;
    ...
}
```
memset(&RulesObj,0,sizeof(ESS_OBJDEF_T));
memset(&DataObj,0,sizeof(ESS_OBJDEF_T));
strcpy(appname, "sample");
strcpy(dbname,"basic");

RulesObj.hCtx     = hCtx;
RulesObj.FileName = "genref";
RulesObj.AppName  = appname;
RulesObj.DbName   = dbname;
RulesObj.ObjType  = ESS_OBJTYPE_RULES;

DataObj.hCtx      = hCtx;
DataObj.FileName  = "genref";
DataObj.AppName   = appname;
DataObj.DbName    = dbname;
DataObj.ObjType   = ESS_OBJTYPE_TEXT;

ErrorName         = "builddim.err";

sts = EssBeginIncrementalBuildDim(hCtx);

if (!sts)
  sts = EssIncrementalBuildDim(hCtx,&RulesObj,&DataObj,NULL,ErrorName,true,ESS_INCDIMBUILD_BUILD ,NULL);
  if (!sts)
    sts = EssIncrementalBuildDim(hCtx,&RulesObj,&DataObj,NULL,ErrorName,true,ESS_INCDIMBUILD_VERIFY ,NULL);
  if (!sts)
    sts = EssIncrementalBuildDim(hCtx,&RulesObj,&DataObj,NULL,ErrorName,true,ESS_INCDIMBUILD_SAVEOT L,"tmpotl");

sts = EssBeginStreamBuildDim(hCtx, &RulesObj,ESS_INCDIMBUILD_BUILD,"tmpotl");
if (!sts)
  sts = EssSendString(hCtx, "600    600-20   600-20-20\n");
if (!sts)
  sts = EssSendString(hCtx, "600    600-20   600-20-30\n");
if (!sts)
  sts = EssSendString(hCtx, "600    600-40   600-40-20\n");
sts = EssEndStreamBuildDim(hCtx,ErrorName,false);

sts = EssEndIncrementalBuildDim(hCtx,ESS_DOR_ALLDATA,"tmpotl",ErrorName,false);
return sts;
}

See Also

- EssIncrementalBuildDim
- EssBeginIncrementalBuildDim
- EssBeginStreamBuildDim
- EssEndIncrementalBuildDim
- EssEndStreamBuildDim
EssBeginUpdate

Starts sending an update specification to the active database. This call must be followed by successive calls to EssSendString() to send the update specification, and finally by a call to EssEndUpdate(). The update data can either be stored in the database, or just verified and any errors returned. Also, any data blocks locked for update can be unlocked by this call.

Syntax

```c
ESS_FUNC_M EssBeginUpdate (hCtx, Store, Unlock);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>Store</td>
<td>ESS_BOOL_T</td>
<td>Controls storage of data. If TRUE, data is stored in the server; if FALSE, no data is stored.</td>
</tr>
<tr>
<td>Unlock</td>
<td>ESS_BOOL_T</td>
<td>Controls unlocking of data blocks. If TRUE, all relevant blocks which are locked will be unlocked (after data is stored, if necessary). If FALSE, no blocks are unlocked.</td>
</tr>
</tbody>
</table>

Notes

- This function must be followed by at least one call to EssSendString(), followed by a call to EssEndUpdate().
- Each string passed to EssSendString() following this function must be terminated with a carriage return/linefeed character sequence ("\r\n").
- If both the Store and Unlock flags are set to FALSE, the database merely performs a syntax check of the update specification.
- Unicode clients using the C Main API to communicate with Unicode-enabled Essbase applications must send the UTF-8 encoded Unicode byte order mark (BOM) in the text stream immediately after calling this function. For an example, see “Specifying the Byte Order Encoding” on page 75.

Return Value

None.

Access

This function requires the caller to have write privilege (ESS_PRIV_WRITE) to the active database.

Example

```c
ESS_VOID_T Ess_BeginUpdate(ESS_HCTX_T hCtx)
{
    ESS_FUNC_M     sts = ESS_STS_NOERR;
    ESS_BOOL_T     Store;
    ESS_BOOL_T     Unlock;
    ESS_STR_T      Query;

    Store = ESS_TRUE;
    Unlock = ESS_FALSE;
}```
Query = "Year Market Scenario Measures Product 12345";

/* Begin Update */
sts = EssBeginUpdate (hCtx, Store, Unlock);

/* Send update specification */
if(!sts)
    sts = EssSendString(hCtx, Query);

/* End Update */
if(!sts)
    sts = EssEndUpdate(hCtx);
}

See Also
- EssBeginReport
- EssEndUpdate
- EssSendString
- EssUpdate
- EssUpdateFile

EssBuildDimension

Allows the addition or removal of members from the outline in the active database from a data file and rules file.

Syntax

ESS_FUNC_M EssBuildDimension (hCtx, rulesObj, dataObj, mbrUser, ErrorName);

Parameter | Data Type | Description
--- | --- | ---
hCtx | ESS_HCTX_T | API context handle.
pRulesObj | “ESS_OBJDEF_T” on page 151 | Pointer to rules file object definition structure.
pDataObj | “ESS_OBJDEF_T” on page 151 | Pointer to data file object definition structure.
pMbrUser | “ESS_MBRUSER_T” on page 148 | SQL user structure (if data source is SQL database). A NULL SQL user structure indicates a non SQL data source.
ErrorName | ESS_STR_T | Name of error output file on client.

Notes
- If MbrUser is not NULL, an SQL data source is assumed.
- See EssImport for information on importing data sources.
- The database must be the active database. See EssSetActive.

Return Value
None.
Access

This function requires the caller to have database design privilege for the specified database (ESS_PRIV_DBDESIGN).

Example

ESS_FUNC_M
ESS_BuildDim(ESS_HCTX_T hCtx)
{
    ESS_FUNC_M    sts = ESS_STS_NOERR;
    ESS_OBJDEF_T  RulesObj;
    ESS_OBJDEF_T  DataObj;
    ESS_MBRUSER_T User;
    ESS_STR_T     ErrorName;

    RulesObj.hCtx     = hCtx;
    RulesObj.FileName = "Prodmap";
    RulesObj.ObjType  = ESS_OBJTYPE_RULES;

    DataObj.hCtx      = hCtx;
    DataObj.FileName  = "Prodtabl";
    DataObj.ObjType   = ESS_OBJTYPE_TEXT;

    ErrorName         = "builddim.err";

    sts               = EssBuildDimension (hCtx, &RulesObj, &DataObj,
    NULL, ErrorName);

    return (sts);
}

C Main API Functions
See Also
- EssImport
- EssBuildDimFile
- EssBuildDimStart
- EssOtlRestructure

**EssBuildDimFile**

Builds a data file used to add or remove members from the active database outline. See EssBuildDimension for more information.

**Syntax**

```c
ESS_FUNC_M EssBuildDimFile (hCtx, RulesObj, DataObj, MbrUser, ErrorName, fOverwriteErrorFile);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>RulesObj</td>
<td>“ESS_OBJDEF_T” on page 151</td>
<td>Pointer to rules file object definition structure.</td>
</tr>
<tr>
<td>DataObj</td>
<td>“ESS_OBJDEF_T” on page 151</td>
<td>Pointer to data file object definition structure.</td>
</tr>
<tr>
<td>MbrUser</td>
<td>“ESS_MBRUSER_T” on page 148</td>
<td>SQL user structure (if data source is SQL database). NULL structure indicates a non-SQL data source.</td>
</tr>
<tr>
<td>ErrorName</td>
<td>ESS_STR_T</td>
<td>Error name output on client.</td>
</tr>
<tr>
<td>fOverwriteErrorFile</td>
<td>ESS_BOOL_T</td>
<td>A Boolean value which determines whether this function overwrites an existing file name ErrorFile.</td>
</tr>
</tbody>
</table>

**Notes**

- If `MbrUser` is not NULL, an SQL data source is assumed.
- The description of `EssImport` provides information on importing data sources.
- The database must be the active database. See the description of `EssSetActive`.
- `EssBuildDimStart` must be called prior to using `EssBuildDimFile()`.
- `EssBuildDimFile()` can be called repeatedly prior to restructuring to add members via multiple rules and/or data file to the outline.
- The database must be restructured after completion of call(s) to `EssBuildDimFile()`.
- The outline must be unlocked after restructuring.

**Return Value**

Returns a zero if successful.

**Access**

This function requires database design privilege ESS_PRIV_DBDESIGN for the specified database.
Example

ESS_FUNC_M EssBuildDimFile (ESS_HCTX_T hCtx)
{
    ESS_FUNC_M sts = ESS_STS_NOERR;
    ESS_OBJDEF_T RulesObj;
    ESS_OBJDEF_T DataObj;
    ESS_STR_T ErrorName;

    RulesObj.hCtx = hCtx;
    RulesObj.FileName = "Prodmap";
    RulesObj.ObjType = ESS_OBJTYPE_RULES;

    DataObj.hCtx = hCtx;
    DataObj.FileName = "Prodtabl";
    DataObj.ObjType = ESS_OBJTYPE_TEXT;
    ErrorName = "builddim.err";

    sts = EssBuildDimFile (hCtx, &RulesObj,
                          &DataObj, NULL, ErrorName);
    return (sts);
}

See Also

- EssImport
- EssBuildDimension
- EssBuildDimStart
- EssOtlRestructure
- EssUnlockObject

EssBuildDimStart

Starts the process to add or remove members from the active database outline.

Syntax

ESS_FUNC_M EssBuildDimStart (hCtx);

Parameter | Data Type    | Description
----------|--------------|-------------
          | ESS_HCTX_T   | API context handle.

Notes

- See the description of EssImport for information on importing data sources.
- The database must be the active database. See the description of EssSetActive.
- The outline object must be locked prior to calling EssBuildDimStart. See the description for EssLockObject.

Return Value

Returns zero if successful, otherwise returns an error code.
Access

This function requires the caller to have database design privilege for the specified database (ESS_PRIV_DBDESIGN).

Example

```c
ESS_FUNC_M Ess_BuildDimStart (ESS_HCTX_T hCtx)
{
    sts = EssBuildDimStart (hCtx);
    return (sts);
}
```

See Also

- EssImport
- EssBuildDimension
- EssBuildDimFile
- EssLockObject
- EssOtlRestructure

EssCalc

Sends a single string. This function is equivalent to making a call to EssBeginCalc(), followed by calls to EssSendString(), and finally to EssEndCalc(). The calculation can either be initiated, or the calc script can just be verified and any errors returned.

Syntax

```c
ESS_FUNC_M EssCalc (hCtx, Calculate, CalcScript);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>Calculate</td>
<td>ESS_BOOL_T</td>
<td>Controls calculation of the calc script. If TRUE, the calc script is executed and the call is asynchronous.</td>
</tr>
<tr>
<td>CalcScript</td>
<td>ESS_OBJ_T</td>
<td>The calc script, as a single string (must be less than 64 KB).</td>
</tr>
</tbody>
</table>

Notes

- The calc script string must be less than 64 KB long.
- If this function succeeds and the calculation is started, it will continue on the server as an asynchronous process after the return from this call. The caller must check at regular intervals to see if the process has completed by calling EssGetProcessState() until it returns ESS_STATE_DONE.
- This API call is asynchronous only if the Calculate parameter is TRUE. Otherwise, it is a simple synchronous request.

  During an asynchronous request, control is passed back to the program immediately, before the request completes. The set of valid requests for the current API context handle is limited.
during the time the asynchronous request is running. If you give an invalid request during that time, an error is returned. The list of valid API calls on the API context during an asynchronous operation is: EssGetProcessState, EssCancelProcess.

- If the Calculate flag is set to FALSE, the database merely performs a syntax check of the calc script, and the call is synchronous.

Return Value

None.

Access

This function requires the caller to have calc privilege (ESS_PRIV_CALC) to the active database.

Example

```c
ESS_FUNC_M
ESS_CalcLine (ESS_HCTX_T hCtx)
{
    ESS_FUNC_M sts = ESS_STS_NOERR;
    ESS_STR_T Script;
    ESS_PROCSTATE_T pState;

    Script = "CALC ALL;";
    sts = EssCalc(hCtx, ESS_TRUE, Script);
    if (!sts)
    {
        sts = EssGetProcessState (hCtx, &pState);
        while (!sts && (pState.State != ESS_STATE_DONE))
        {
            sts = EssGetProcessState (hCtx, &pState);
        }
        return(sts);
    }
}
```

See Also

- EssBeginCalc
- EssCalcFile
- EssDefaultCalc
- EssEndCalc
- EssGetDefaultCalc
- EssGetProcessState
- EssSendString
- EssSetDefaultCalc

**EssCalcFile**

Executes a calc script against the active database from a file.

**Syntax**

```c
ESS_FUNC_M EssCalcFile (hDestCtx, hSrcCtx,AppName, DbName, FileName, Calculate);
```
### Parameter Data Type Description

- **hDestCtx** ESS_HCTX_T API context handle of target database on the server.
- **hSrcCtx** ESS_HCTX_T API context handle for calc script file location. The calc script file can reside on the client or on the same server as the target database.
- **AppName** ESS_STR_T Application name for calc script file location.
- **DbName** ESS_STR_T Database name for calc script file location.
- **FileName** ESS_STR_T Name of calc script file.
- **Calculate** ESS_BOOL_T Controls calculation of the calc script. If TRUE, the calc script is executed and the call is asynchronous.

### Notes
- The calc script must not be greater than 64 KB long.
- If this function succeeds and the calculation is started, it will continue on the server as an asynchronous process after the return from this call. The caller must check at regular intervals to see if the process has completed by calling `EssGetProcessState()` until it returns ESS_STATE_DONE.
- This API call is asynchronous only if the Calculate parameter is TRUE. Otherwise, it is a simple synchronous request.
  
  During an asynchronous request, control is passed back to the program immediately, before the request completes. The set of valid requests for the current API context handle is limited during the time the asynchronous request is running. If you give an invalid request during that time, an error is returned. The list of valid API calls on the API context during an asynchronous operation is: `EssGetProcessState`, `EssCancelProcess`.

### Return Value

None.

### Access

This function requires the caller to have calc privilege (ESS_PRIV_CALC) to the active database.

### Example

```c
ESS_FUNC_M
ESS_CalcFile (ESS_HCTX_T hCtx)
{
    ESS_FUNC_M      sts = ESS_STS_NOERR;
    ESS_SHORT_T     isResponse;
    ESS_HCTX_T      hSrcCtx;
    ESS_BOOL_T      isObject = ESS_FALSE;
    ESS_STR_T       AppName;
    ESS_STR_T       DbName;
    ESS_STR_T       FileName;
    ESS_PROCSTATE_T pState;

    hSrcCtx  = hCtx;
    AppName  = "Sample";
```
DbName   = "Basic";
FileName = "Test";

sts = EssCalcFile (hCtx, hSrcCtx, AppName,
                  _DbName, FileName, ESS_TRUE);
if (!sts) {
    sts = EssGetProcessState (hCtx, &pState);
    while (!sts && (pState.State !=
                   ESS_STATE_DONE))
        sts = EssGetProcessState (hCtx, &pState);
    return(sts);
}

See Also
- EssBeginCalc
- EssCalc
- EssDefaultCalc
- EssSetDefaultCalcFile

**EssCancelAsyncProc**

Cancels an asynchronous data load or dimension build process.

**Syntax**

ESS_FUNC_M EssCancelAsyncProc (hCtx, ErrorFileName, ErFileOverWrite);

**Parameter** | **Data Type** | **Description**
---|---|---
hCtx | ESS_HCTX_T | API context handle.
ErrorFileName | ESS_STR_T | An error file name.
ErFileOverWrite | ESS_BOOL_T | If TRUE, overwrite the error file.

**Notes**

Call this function after initiating an asynchronous process using EssAsyncImport or EssAsyncBuildDim.

**Return Value**

If successful, the network connection is closed and the error log is returned. Otherwise, returns an error code.

**Example**

See the example for EssAsyncBuildDim.

**See Also**
- EssAsyncBuildDim
- EssAsyncImport
EssCancelProcess

Cancels an asynchronous process that has not yet completed

Syntax

```c
ESS_FUNC_M EssCancelProcess (hCtx);
```

Parameter | Data Type     | Description
----------|---------------|-------------
hCtx      | ESS_HCTX_T    | API context handle.

Notes

- If you use this function to cancel a process, the database may be left in an inconsistent state, with only some of the data recalculated.
- Calling this function except after initiating a successful asynchronous database operation (e.g. a calculation) will generate an error.

Return Value

None.

Access

This function requires no special privilege.

Example

```c
ESS_VOID_T
ESS_CancelProcess(ESS_HCTX_T hCtx)
{
    ESS_FUNC_M st = ESS_STS_NOERR;
    ESS_STR_T Script;
    ESS_PROCSTATE_T pState;
    ESS_USHORT_T Count;

    Script = "CALC ALL;";
    st = EssBeginCalc (hCtx, ESS_TRUE);
    if (!st)
        st = EssSendString (hCtx, Script);
    if (!st)
        st = EssEndCalc (hCtx);
    /*************************************
    Check process state and cancel it
    if it takes too long
    *************************************/
    ```
if (!sts)
{
    sts = EssGetProcessState (hCtx, &pState);
    while(!sts && (pState.State != ESS_STATE_DONE))
    {
        Count = Count + 1;
        if (Count == 1000)
            sts = EssCancelProcess(hCtx);

        sts = EssGetProcessState (hCtx, &pState);
    }
}

See Also
- EssBeginCalc
- EssCalc
- EssGetProcessState
- EssImport

EssCheckAttributes

Returns attribute information for each specified member.

Syntax

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx;</td>
<td>ESS_HCTX_T</td>
<td>Context handle</td>
</tr>
<tr>
<td>Count;</td>
<td>ESS_USHORT_T</td>
<td>Number of given dimensions and members</td>
</tr>
<tr>
<td>pMemberNameArray;</td>
<td>ESS_PMBRNAME_T</td>
<td>An array of names of given dimensions and members</td>
</tr>
<tr>
<td>ppAttributeTypeArray;</td>
<td>ESS_PPUSHORT_T</td>
<td>One of the following constant identifiers (see Table 6, “C API Attributes Terminology,” on page 102) for the attribute type array:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESS_ATTRIBUTE_DIMENSION</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESS_ATTRIBUTE_MEMBER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESS_STANDARD_DIMENSION</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESS_STANDARD_MEMBER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESS_BASE_DIMENSION</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESS_BASE_MEMBER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESS_ATTRIBUTED_MEMBER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESS_INVALID_MEMBER</td>
</tr>
</tbody>
</table>

Access

This function requires no special privileges.
void ESS_CheckAttributes()
{
    ESS_STS_T sts=-1,sts1=-1;
    int counter,i,j;
    ESS_PMBRNAME_T pMbrNames=ESS_NULL;
    ESS_PUSHORT_T pMbrAttrTypes=ESS_NULL;
    ESS_CHAR_T buf[80]=""

    /* counter = 4; */
    printf("Please enter the number of member names that follow: ");
    gets(buf);
    counter=atoi(buf);
    if (counter)
    {
        sts1 = EssAlloc(hInst, (counter * sizeof(ESS_MBRNAME_T)),
                (ESS_PPVOID_T)&pMbrNames);
        if (!sts1)
        {
            memset(pMbrNames, 0, (counter * sizeof(ESS_MBRNAME_T)));
            for (i = 0; i < counter; i++)
            {
                printf("Enter member name: ");
                gets(buf);
                strcpy(pMbrNames[i],buf);
            }
        }
        sts = EssCheckAttributes(hCtx,counter,pMbrNames,&pMbrAttrTypes);
        if (sts)
            fprintf(stderr, "sts = %ld \n",sts);
        else if (pMbrAttrTypes)
        {
            for (j = 0; j < counter; j++)
            {
                switch(pMbrAttrTypes[j])
                {
                    case ESS_STANDARD_MEMBER:
                        strcpy(buf,"ESS_STANDARD_MEMBER");
                        break;
                    case ESS_STANDARD_DIMENSION:
                        strcpy(buf,"ESS_STANDARD_DIMENSION");
                        break;
                    case ESS_BASE_MEMBER:
                        strcpy(buf,"ESS_BASE_MEMBER");
                        break;
                    case ESS_BASE_DIMENSION:
                        strcpy(buf,"ESS_BASE_DIMENSION");
                        break;
                    case ESS_ATTRIBUTE_MEMBER:
                        strcpy(buf,"ESS_ATTRIBUTE_MEMBER");
                        break;
                }
            }
        }
    }
}
case ESS_ATTRIBUTE_DIMENSION:
    strcpy(buf,"ESS_ATTRIBUTE_DIMENSION");
    break;

case ESS_ATTRIBUTED_MEMBER:
    strcpy(buf,"ESS_ATTRIBUTED_MEMBER");
    break;

default:
    strcpy(buf,"Unknown attribute type");
    break;  
}
printf("%s is of type %s\n",pMbrNames[j],buf);
}
printf("\n");
}

See Also

- EssFreeStructure
- EssGetAssociatedAttributesInfo
- EssGetAttributeInfo
- EssGetAttributeSpecifications
- EssOtlAssociateAttributeDimension
- EssOtlAssociateAttributeMember
- EssOtlDisassociateAttributeDimension
- EssOtlDisassociateAttributeMember
- EssOtlFindAttributeMembers
- EssOtlFreeStructure
- EssOtlGetAssociatedAttributes
- EssOtlGetAttributeInfo
- EssOtlGetAttributeSpecifications
- EssOtlQueryAttributes
- EssOtlSetAttributeSpecifications

EssCheckMemberName

Checks if a string is a valid member name within the active database outline.

Syntax

ESS_FUNC_M EssCheckMemberName (hCtx, MbrName, pValid);

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>MbrName</td>
<td>ESS_STR_T</td>
<td>Member name to be verified.</td>
</tr>
<tr>
<td>pValid</td>
<td>ESS_PBOOL_T</td>
<td>Address of variable to receive valid member flag.</td>
</tr>
</tbody>
</table>

See Also

- EssFreeStructure
- EssGetAssociatedAttributesInfo
- EssGetAttributeInfo
- EssGetAttributeSpecifications
- EssOtlAssociateAttributeDimension
- EssOtlAssociateAttributeMember
- EssOtlDisassociateAttributeDimension
- EssOtlDisassociateAttributeMember
- EssOtlFindAttributeMembers
- EssOtlFreeStructure
- EssOtlGetAssociatedAttributes
- EssOtlGetAttributeInfo
- EssOtlGetAttributeSpecifications
- EssOtlQueryAttributes
- EssOtlSetAttributeSpecifications
Notes
This function checks whether the relational span Boolean is set and can determine if the specified
member name is valid in the relational store.

Return Value
If successful, this function returns a flag, `pValid`, indicating if the name string `MbrName` is a
valid member name in the active database outline.

Access
This function requires the caller to have at least read access (ESS_PRIV_READ) to the database,
and to have selected it as their active database using `EssSetActive()`.

Example

```c
ESS_FUNC_M
ESS_CheckMemberName(ESS_HCTX_T hCtx)
{
    ESS_FUNC_M     sts;
    ESS_STR_T      MbrName;
    ESS_BOOL_T     pValid;

    MbrName = "Profit";
    sts = EssCheckMemberName(hCtx, MbrName, &pValid);

    if(pValid)
        printf("\"%s\" is a valid member name\n", MbrName);

    return (sts);
}
```

See Also
- `EssGetMemberInfo`
- `EssQueryDatabaseMembers`
- `EssSetActive`

EssClearActive
Clears the user's current active application and database.

Syntax

```c
ESS_FUNC_M EssClearActive (hCtx);
```

Parameter | Data Type | Description
---|---|---
hCtx | ESS_HCTX_T | API context handle.

Return Value
None.
**Access**

This function requires no special privileges.

**Example**

```c
ESS_FUNC_M
ESS_UnloadDb (ESS_HCTX_T hCtx)
{
    ESS_FUNC_M        sts = ESS_STS_NOERR;
    ESS_STR_T       AppName;
    ESS_STR_T        DbName;
   AppName = "Sample";
    DbName  = "Basic";
    /*
    * IF the current active is the same as the
    * unload db, ClearActive first
    */
    sts = EssClearActive(hCtx);
    /*
    * ELSE
    *
    */
    sts = EssUnloadDatabase(hCtx, AppName,
                      DbName);
    return (sts);
}
```

**See Also**

- EssGetActive
- EssSetActive

### EssClearAliases

Permanently removes all alias tables for the active database.

**Syntax**

```c
ESS_FUNC_M EssClearAliases (hCtx);
```

**Parameter Data Type Description**

| hCtx     | ESS_HCTX_T | API context handle. |

**Notes**

- This function can not remove the active alias table or the default alias table.
- Use EssSetAlias() to set an active alias to "default" prior to using this API function.
- Make sure that no one else is using the same database as the one you try to clear alias tables from by calling EssListConnections().

**Return Value**

None.
Access

This function requires the caller to have at least read access (ESS_PRIV_READ) to the database, and to have selected it as their active database using EssSetActive().

Example

```c
ESS_FUNC_M
ESS_ClearAliases (ESS_HCTX_T hCtx)
{
    ESS_FUNC_M sts = ESS_STS_NOERR;
    sts = EssClearAliases(hCtx);
    if(!sts)
        printf("All alias tables are removed.\r\n");
    return (sts);
}
```

See Also

- EssListAliases
- EssRemoveAlias
- EssSetActive

EssClearDatabase

Clears all loaded data in the active database.

Syntax

```c
ESS_FUNC_M EssClearDatabase (hCtx);
```

Parameter | Data Type          | Description
----------|-------------------|-------------
           | ESS_HCTX_T        | API context handle.

Notes

- Data deleted using this function cannot be restored. Use it with care!

Return Value

None.

Access

This function requires the caller to have Write privilege (ESS_PRIV_WRITE) for the database, and to have selected it as their active database using EssSetActive().

Example

```c
ESS_FUNC_M
ESS_ClearDb (ESS_HCTX_T hCtx)
{
    ESS_FUNC_M sts = ESS_STS_NOERR;
    sts = EssClearDatabase(hCtx);
    return (sts);
}
```
EssCloseAsyncProc

Closes the connection for a finished or canceled asynchronous dimension build or data load, and returns the current state of the process.

Syntax

```c
ESS_FUNC_M EssCloseAsyncProc (hCtx, ProcState);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>ProcState</td>
<td>ESS_PBLDDL_STATE_T</td>
<td>Address of pointer to receive allocated process state structure.</td>
</tr>
</tbody>
</table>

Notes

Call this function after initiating an asynchronous process using EssAsyncImport or EssAsyncBuildDim.

Return Value

Returns zero if successful. Otherwise, returns an error code.

Example

See the example for EssAsyncBuildDim.

See Also

- EssAsyncBuildDim
- EssAsyncImport
- EssGetAsyncProcLog
- EssGetAsyncProcState
- EssCancelAsyncProc

EssClrSpanRelationalSource

Clears the Boolean bSpanRelPart field informing Essbase that pertinent data exists in an attached relational store. Some other API functions, such as EssQueryDatabaseMembers, read bSpanRelPart and access the relational store if bSpanRelPart is set.

Syntax

```c
ESS_FUNC_M EssClrSpanRelationalSource (hCtx);
```
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
</tbody>
</table>

**Notes**

Several API functions have been enhanced to retrieve information from relational stores.

- **EssQueryDatabaseMembers** - returns member names from the relational store.
- **EssGetMemberInfo** - returns information on members in the relational store.
- **EssCheckMemberName** - checks in the relational store for valid member names.
- **EssGetMemberCalc** - recognizes a relational member passed as input and returns a null string for all relational members.

**Return Value**

None.

**Access**

This function requires the caller to have read privilege (ESS_PRIV_READ) to one or more members in the active database.

**Example**

```c
ESS_FUNC_M
ESS_Report (ESS_HCTX_T   hCtx,
            ESS_HINST_T  hInst
)
{
    ESS_FUNC_M      sts     = ESS_STS_NOERR;
    ESS_STR_T     rString = NULL;
    sts = EssBeginReport (hCtx, ESS_TRUE, ESS_FALSE);
    if (!sts)
        sts = EssSendString (hCtx, "<Desc Year !");
    if (!sts)
        sts = EssClrSpanRelationalSource (hCtx);
    /**************
    * Get report *
    ***************/

    if (!sts)
        sts = EssGetString (hCtx, &rString);
    while (!sts && (rString != NULL))
    {
        printf ("%s", rString);
        EssFree (hInst, rString);
        sts = EssGetString (hCtx, &rString);
    }
    printf ("\r\n");
    return(sts);
}
```
EssCommitDatabase

No longer in use because commits are handled automatically by the Essbase Server. This function now returns the error message ESS_STS_OBSOLETE. See the Oracle Essbase Database Administrator’s Guide for details about committing data.

EssCompactOutline

Compacts an outline file that requires compacting at the server side.

Syntax

ESS_FUNC_M EssCompactOutline (hCtx);

Parameter | Data Type | Description
-----------|-----------|-------------
hCtx       | ESS_HCTX_T | API context acquired during login

Notes

- The function requires that the user is set active.

Return Value

Returns 0 if successful. After verifying that no users are performing an action, an outline-only restructure is performed.

Example

```c
#include <windows.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>

#pragma pack(push, api, 1)
#include <essapi.h>
#include <essotl.h>
#pragma pack(pop, api)

/* default names */
ESS_SVRNAME_T srvrName = "localhost";
ESS_USERNAME_T userName = "essexer";
ESS_PASSWORD_T pswd = "password";
ESS_APPNAME_T app = "ASOSamp";
ESS_DBNAME_T db = "Sample";

int main(int argc, char *argv[]) {
    
    ESS_STS_T sts = ESS_STS_NOERR;
    ESS_HINST_T hInst = NULL;
```
ESS_HOUTLINE_T hOutlineQuery = NULL, hOutline = NULL;
ESS_HCTX_T hCtx = NULL;
ESS_USHORT_T Items;
ESS_PAPPDB_T pAppsDbs = NULL;
ESS_ACCESS_T Access;

ESS_INIT_T InitStruct =        /* Define init */
    /* structure */
{
    ESS_API_VERSION,       /* Version of API */
    (ESS_PVOID_T)0,    /* user-defined message context */
    0, /* max handles */
    0L, /* max buffer size */
    NULL, //(ESS_STR_T)"C:\Hyperion\products\Essbase\EssbaseServer", /* local path */
    /* The following parameters use defaults */
    NULL, /* message db path */
    NULL, /* allocation function pointer */
    NULL, /* reallocation function pointer */
    NULL, /* free function pointer */
    NULL, //(ESS_PFUNC_T)MessageFunc, /* error handling function pointer */
    NULL, /* path name of user-defined */
    /* Application help file */
    0L /* Reserved for internal use. */
    /* Set to NULL */
#endif
    , ESS_API_UTF8
#endif

/* initialize the Essbase API */
if ((sts = EssInit(&InitStruct, &hInst)) != ESS_STS_NOERR)
{
    printf("EssInit failure: %ld\n", sts);
    exit ((int) sts);
}

/* login to Essbase */
if ((sts = EssLogin (hInst, srvrName, userName, pswd, &Items, &pAppsDbs, &hCtx)) != ESS_STS_NOERR)
{
    printf("EssLogin failure: %ld\n", sts);
    exit ((int) sts);
}
if(pAppsDbs)
    EssFree(hInst, pAppsDbs);

/* Select the application */
if ((sts = EssSetActive(hCtx, app, db, &Access)) != ESS_STS_NOERR)
{
    printf("EssSetActive failure: %ld\n", sts);
    exit ((int) sts);
}

/* compact the outline and restructure */
if ((sts = EssCompactOutline(hCtx)) != ESS_STS_NOERR)
{
    printf("EssCompactOutline failure: %ld\n", sts);
    exit ((int) sts);
}

/* done, logout and terminate the api */
if ((sts = EssLogout (hCtx)) != ESS_STS_NOERR)
{
    printf("EssLogout failure: %ld\n", sts);
    exit ((int) sts);
}

if ((sts = EssTerm(hInst)) != ESS_STS_NOERR)
{
    /* error terminating API */
    exit((int) sts);
}

return(0);


EssConvertApplicationToUnicode

Create a Unicode mode application. When defined to be in Unicode mode, Essbase Server allows the migration of non-Unicode mode applications to Unicode mode.

Syntax

ESS_FUNC_M EssConvertApplicationToUnicode(hCtx, AppName);

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>AppName</td>
<td>ESS_STR_T</td>
<td>The name of the application to be migrated. The named application must exist and not be in Unicode mode.</td>
</tr>
</tbody>
</table>

Return Value

Returns 0 if successful; otherwise, returns an error.
Access
The caller must have Application Create/Delete/Edit privilege (ESS_PRIV_APPCREATE).

See Also
- EssCreateApplicationEx

**EssCopyApplication**

Copies an existing application, either on the client or the server, to a new application, including all associated databases and objects.

Syntax

```c
ESS_FUNC_M EssCopyApplication (hCtx, hSrcCtx, SrcApp, DestApp);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>hSrcCtx</td>
<td>ESS_HCTX_T</td>
<td>Not used—should be same as hCtx.</td>
</tr>
<tr>
<td>SrcApp</td>
<td>ESS_STR_T</td>
<td>Name of existing application to copy.</td>
</tr>
<tr>
<td>DestApp</td>
<td>ESS_STR_T</td>
<td>Name of new application. See “Application Name Limits” on page 1727.</td>
</tr>
</tbody>
</table>

Notes
- Copying a client application copies the local application directory and contents.
- This function can only be used to copy a client application to a new application on the client, or a server application to a new application on the same server. Use EssCopyObject() to copy an application between different servers.
- The new application is not started. Call EssLoadApplication to start the newly copied application.

Return Value

None.

Access

For a server application, the caller must have Application Create/Delete/Edit privilege (ESS_PRIV_APPCREATE), and application designer privilege on the source application to be copied (ESS_PRIV_APPDESIGN).

Example

```c
ESS_FUNC_M
ESS_CopyApp(ESS_HCTX_T hCtx)
{
    ESS_FUNC_M sts = ESS_STS_NOERR;
    ESS_HCTX_T hSrcCtx;
    ESS_STR_T SrcApp;
    ESS_STR_T DestApp;
```
hSrcCtx = hCtx;
SrcApp = "Sample";
DestApp = "NewSamp";

sts = EssCopyApplication(hCtx, hSrcCtx, SrcApp, DestApp);
return (sts);
}

See Also

- EssCopyDatabase
- EssCopyObject
- EssLoadApplication

EssCopyDatabase

Copies an existing database, either on the client or the server, to a new database, including all associated databases and objects. If the database is copied on the server, the new database is started.

**Syntax**

```c
ESS_FUNC_M EssCopyDatabase (hCtx, hSrcCtx, SrcApp, DestApp, SrcDb, DestDb);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>hSrcCtx</td>
<td>ESS_HCTX_T</td>
<td>Not used - should be same as hCtx.</td>
</tr>
<tr>
<td>SrcApp</td>
<td>ESS_STR_T</td>
<td>Name of source application.</td>
</tr>
<tr>
<td>DestApp</td>
<td>ESS_STR_T</td>
<td>Name of destination application.</td>
</tr>
<tr>
<td>SrcDb</td>
<td>ESS_STR_T</td>
<td>Name of existing database to copy.</td>
</tr>
<tr>
<td>DestDb</td>
<td>ESS_STR_T</td>
<td>Name of new database. See “Database Name Limits” on page 1728.</td>
</tr>
</tbody>
</table>

**Notes**

- Copying a client database copies the local database directory and contents.
- This function can only be used to copy a client database to another database on the client, or a server database to another database on the same server. Use EssCopyObject() to copy a database between different servers.

**Return Value**

None.
Access

For a server database, the caller must have database Create/Delete/Edit privilege (ESS_PRIV_DBCREATE), and database designer privilege on the source database to be copied (ESS_PRIV_DBDESIGN).

Example

```c
ESS_FUNC_M
ESS_CopyDatabase(ESS_HCTX_T hCtx) {
    ESS_FUNC_M     sts = ESS_STS_NOERR;
    ESS_HCTX_T    hSrcCtx;
    ESS_STR_T     SrcApp;
    ESS_STR_T     DestApp;
    ESS_STR_T     SrcDb;
    ESS_STR_T     DestDb;

    hSrcCtx = hCtx;
    SrcApp  = "Sample";
    DestApp = "NewSamp";
    SrcDb   = "Basic";
    DestDb  = "NewBasic";

    sts = EssCopyDatabase(hCtx, hSrcCtx, SrcApp, DestApp, SrcDb, DestDb);

    return(sts);
}
```

See Also

- EssCopyApplication
- EssCopyObject

EssCopyFilter

Copies an existing filter.

Syntax

```c
ESS_FUNC_M EssCopyFilter (hCtx, hSrcCtx, SrcApp, DestApp, SrcDb, DestDb, SrcName, DestName);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>hSrcCtx</td>
<td>ESS_HCTX_T</td>
<td>Not used—should be same as hCtx.</td>
</tr>
<tr>
<td>SrcApp</td>
<td>ESS_STR_T</td>
<td>Source application name.</td>
</tr>
<tr>
<td>DestApp</td>
<td>ESS_STR_T</td>
<td>Destination application name.</td>
</tr>
<tr>
<td>SrcDb</td>
<td>ESS_STR_T</td>
<td>Source databasesname.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DestDb</td>
<td>ESS_STR_T</td>
<td>Destination database name.</td>
</tr>
<tr>
<td>SrcName</td>
<td>ESS_STR_T</td>
<td>Source name of existing filter to be copied.</td>
</tr>
<tr>
<td>DestName</td>
<td>ESS_STR_T</td>
<td>Destination name of copied filter. See “Filter Name Limits” on page 1728.</td>
</tr>
</tbody>
</table>

**Notes**

- The source filter must already exist.
- To prevent overwriting an existing filter by mistake, the caller should check whether the destination filter already exists.

**Return Value**

None.

**Access**

This function requires the caller to have database design privilege (ESS_PRIV_DBDESIGN) for the specified database.

**Example**

```c
ESS_FUNC_M
ESS_CopyFilter (ESS_HCTX_T  hCtx)
{
    ESS_FUNC_M        sts = ESS_STS_NOERR;
    ESS_HCTX_T        hSrcCtx;
    ESS_STR_T         SrcApp;
    ESS_STR_T         DestApp;
    ESS_STR_T         SrcDb;
    ESS_STR_T         DestDb;
    ESS_STR_T         SrcName;
    ESS_STR_T         DestName;

    hSrcCtx   = hCtx;
    SrcApp    = "Sample";
    SrcDb     = "Basic";
    SrcName   = "OldFilter";
    DestApp   = "Sample";
    DestDb    = "Basic";
    DestName  = "NewFilter";

    sts = EssCopyFilter(hCtx, hSrcCtx, SrcApp, DestApp, SrcDb, DestDb, SrcName, DestName);
    if(!sts)
        printf("The Filter is copied.\r\n");

    return (sts);
}
```

**See Also**

- `EssDeleteFilter`
- `EssListFilters`
EssCopyObject

Copies an object to the server or client object system.

**Syntax**

```c
ESS_FUNC_M EssCopyObject (hSrcCtx, hDestCtx, ObjType, SrcApp, DestApp, SrcDb, DestDb, SrcObj, DestObj);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hSrcCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle for source object. Can be local context handle returned by</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EssCreateLocalContext().</td>
</tr>
<tr>
<td>hDestCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle for destination object. Can be local context handle</td>
</tr>
<tr>
<td></td>
<td></td>
<td>returned by EssCreateLocalContext().</td>
</tr>
<tr>
<td>ObjType</td>
<td>ESS_OBJTYPE_T</td>
<td>Object type (must be single type). See Bitmask Data Types for possible</td>
</tr>
<tr>
<td></td>
<td></td>
<td>values.</td>
</tr>
<tr>
<td>SrcApp</td>
<td>ESS_STR_T</td>
<td>Source application name.</td>
</tr>
<tr>
<td>DestApp</td>
<td>ESS_STR_T</td>
<td>Destination application name.</td>
</tr>
<tr>
<td>SrcDb</td>
<td>ESS_STR_T</td>
<td>Source database name. If NULL, uses the source application subdirectory.</td>
</tr>
<tr>
<td>DestDb</td>
<td>ESS_STR_T</td>
<td>Destination database name. If NULL, uses the destination application</td>
</tr>
<tr>
<td></td>
<td></td>
<td>subdirectory.</td>
</tr>
<tr>
<td>SrcObj</td>
<td>ESS_STR_T</td>
<td>Name of source object to copy from.</td>
</tr>
<tr>
<td>DestObj</td>
<td>ESS_STR_T</td>
<td>Name of destination object to copy to. See &quot;Object Name Limits&quot; on page 1728</td>
</tr>
</tbody>
</table>

**Notes**

- Objects may be copied from client to server, server to client, within the same server. In all cases the destination object must either not already exist or it must be locked by the caller.
- Outline objects cannot be copied. Use the EssCopyDatabase() function to copy a database, including its associated outline.

**Return Value**

None.

**Access**

This function requires the caller to have the appropriate level of access to the specified source application and/or database containing the object (depending on the object type), and to have Application or Database Design privilege (ESS_PRIV_APPDESIGN or ESS_PRIV_DBDESIGN) for the specified destination application or database.

**Example**

```c
ESS_FUNC_M
ESS_CopyObject(ESS_HCTX_T hCtx)
```
hDestCtx  = hCtx;
SrcApp    = "Sample";
SrcDb     = "Basic";
SrcObj    = "Test";
DestApp   = "Sample";
DestDb    = "Basic";
DestObj   = "NewTest";
ObjType   = ESS_OBJTYPE_TEXT;

sts = EssCopyObject(hCtx, hDestCtx, ObjType, SrcApp, DestApp, SrcDb, DestDb, SrcObj, DestObj);

if(!sts)
    printf("The Object is copied.\n\n");

return (sts);
}

See Also

- EssCreateObject
- EssDeleteObject
- EssListObjects
- EssRenameObject

EssCreateApplication

Creates a new application, either on the client or the server. If the application is created on the server, it is also started.

Syntax

ESS_FUNC_M EssCreateApplication (hCtx, AppName);

Parameter  Data Type  Description

hCtx        ESS_HCTX_T  API context handle.
AppName     ESS_STR_T    Name of application to create. See “Application Name Limits” on page 1727.

Notes

- Creating a client application creates a directory to contain local application files.
- A newly created database or application is not automatically set to active. Call
  EssSetActive() after calling EssCreateDatabase() or EssCreateApplication() to keep
subsequent functions, such as EssRestructure(), from operating on the wrong database or application (the application or database that is already active).

- To create a Unicode mode application, use EssCreateApplicationEx.

**Return Value**

None.

**Access**

For a server application, the caller must have Application Create/Delete/Edit privilege (ESS_PRIV_APPCREATE).

**Example**

```c
ESS_FUNC_M
ESS_CreateApp (ESS_HCTX_T hCtx)
{
    ESS_FUNC_M        sts = ESS_STS_NOERR;
    ESS_STR_T        AppName;
    AppName = "Sample";
    sts = EssCreateApplication (hCtx, AppName);
    return(sts);
}
```

**See Also**

- EssCreateStorageTypedApplication
- EssCreateDatabase
- EssCreateObject
- EssCreateApplicationEx

**EssCreateApplicationEx**

Creates a new application, either on the client or the server. If the application is created on the server, it is also started. This function can create Unicode mode applications.

**Syntax**

```c
ESS_FUNC_M EssCreateApplicationEx(hCtx, AppName, usAppType);
```

**Parameter** | **Data Type** | **Description**
---|---|---
hCtx | ESS_HCTX_T | API context handle.
AppName | ESS_STR_T | Name of application to create. See “Application Name Limits” on page 1727.
usAppType | | The application type (Unicode or non-Unicode) of the new application. The valid values are:
  - ESS_APP_UNICODE - 0x0003—Create a Unicode application. The function fails if the server is not in Unicode mode.
  - ESS_APP_NONUNICODE - 0x0002—Create a non-Unicode application.
Creating a client application creates a directory to contain local application files.

A newly created database or application is not automatically set to active. Call EssSetActive() after calling EssCreateDatabaseEx() or EssCreateApplicationEx() to keep subsequent functions, such as EssRestructure(), from operating on the wrong database or application (the application or database that is already active).

Returns 0 if successful; otherwise, returns an error.

For a server application, the caller must have Application Create/Delete/Edit privilege (ESS_PRIV_APPCREATE).

EssCreateDatabase

Creates a new database within an application, either on the client or the server. If the database is created on the server, it is also started.

\[ \text{Syntax} \]

\[ \text{ESS\_FUNC\_M } \text{EssCreateDatabase} \left( hCtx, \ AppName, \ DbName, \ DbType \right); \]

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>AppName</td>
<td>ESS_STR_T</td>
<td>Name of application to contain database.</td>
</tr>
<tr>
<td>DbName</td>
<td>ESS_STR_T</td>
<td>Name of database to create. See “Database Name Limits” on page 1728.</td>
</tr>
<tr>
<td>DbType</td>
<td>ESS_USHORT_T</td>
<td>Type of database to create. Can be ESS_DBTYPE_NORMAL, or ESS_DBTYPE_CURRENCY</td>
</tr>
</tbody>
</table>

Creating a client database creates a directory to contain local database files.

A newly created database or application is not automatically set to active. Call EssSetActive() after calling EssCreateDatabase() or EssCreateApplication() to keep subsequent functions, such as EssRestructure(), from operating on the wrong database or application (the application or database that is already active).
Return Value

None.

Access

For a server database, the caller must have database Create/Delete/Edit privilege (ESS_PRIV_DBCREATE).

Example

```c
ESS_FUNC_M
ESS_CreateDb (ESS_HCTX_T hCtx)
{
    ESS_FUNC_M        sts = ESS_STS_NOERR;
    ESS_STR_T        AppName;
    ESS_STR_T        DbName;

   AppName = "Sample";
   DbName  = "Basic";

    sts = EssCreateDatabase(hCtx, AppName, DbName, ESS_DBTYPE_NORMAL);
    return (sts);
}
```

See Also

- EssCreateApplication
- EssCreateObject

EssCreateDatabaseEx

Creates a new database within an application, either on the client or the server. If the database is created on the server, it is also started. This function can be used to create a database that supports duplicate member names.

Syntax

```c
ESS_FUNC_M EssCreateDatabaseEx (hCtx, AppName, DbName, DbType, bNonUniqueName);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>Essbase API context handle.</td>
</tr>
<tr>
<td>AppName</td>
<td>ESS_STR_T</td>
<td>Name of application to contain database.</td>
</tr>
<tr>
<td>DbName</td>
<td>ESS_STR_T</td>
<td>Name of database to create. See “Database Name Limits” on page 1728.</td>
</tr>
<tr>
<td>DbType</td>
<td>ESS_USHORT_T</td>
<td>Type of database to create. Can be ESS_DBTYPE_NORMAL, or ESS_DBTYPE_CURRENCY</td>
</tr>
<tr>
<td>bNonUniqueName</td>
<td>ESS_BOOL_T</td>
<td>When set to TRUE, EssCreateDatabaseEx creates a database that has a duplicate-member-name support-enabled outline. If set to FALSE, the functionality is the same as EssCreateDatabase().</td>
</tr>
</tbody>
</table>
Notes

- Creating a client database creates a directory to contain local database files.
- A newly created database or application is not automatically set to active. Call EssSetActive() after calling EssCreateDatabase(), EssCreateDatabaseEx() or EssCreateApplication() to keep subsequent functions, such as EssRestructure(), from operating on the wrong database or application (the application or database that is already active).

Return Value

Returns 0 if successful; otherwise, returns an error.

Access

For a server database, the caller must have database Create/Delete/Edit privilege (ESS_PRIV_DBCREATE).

Example

```c
ESS_FUNC_M ESS_CreateDb()
{
    ESS_STS_T sts = ESS_STS_NOERR;
    ESS_STR_T AppName;
    ESS_STR_T DbName;

    AppName = "Sample";
    DbName = "Basic";
    sts = EssCreateDatabaseEx(hCtx, AppName, DbName, ESS_DBTYPE_NORMAL, TRUE);
    return (sts);
}
```

See Also

- EssCreateDatabase
- EssCreateApplication
- EssCreateObject

**EssCreateDrillThruURL**

Creates a drill-through URL, with the given link and name, within the active database outline.

See “Drill-through URL Limits” on page 1729.

**Syntax**

```c
ESS_FUNC_M EssCreateDrillThruURL (hCtx, pUrl);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle</td>
</tr>
<tr>
<td>pUrl</td>
<td>ESS_PDURLINFO_T</td>
<td>URL definition</td>
</tr>
</tbody>
</table>
Return Value

- If successful, creates a drill-through URL in the active database outline.
- If unsuccessful, returns an error code.

Access

- Caller must have database Design privilege (ESS_PRIV_DBDESIGN) for the specified database.
- Caller must have selected the specified database as the active database using EssSetActive().

Example

/* Sample Code for EssCreateDrillThruURL */

ESS_STS_T sts = ESS_STS_NOERR;
ESS_DURLINFO_T url;
ESS_USHORT_T usCountOfURLs, i;
ESS_PDURLINFO_T listOfURLs;
ESS_STR_T urlName = "";
ESS_PDURLINFO_T urlInfo;
ESS_STR_T fileName = "";
ESS_CHAR_T xmlString[XML_CHAR_MAX];

/* Valid case */
memset(&url, '\0', sizeof(ESS_DURLINFO_T));
fileName = "F:\testarea\mainapi\sample1.xml";
GetFileContent(fileName, xmlString);

printf("Valid case:
");  
url.bIsLevel0 = ESS_TRUE;
url.cpURLName = "Drill Through to EPMI";
url.cpURLXml = xmlString;
url.iURLXmlSize = (ESS_SHORT_T) strlen(xmlString)+1;
url.iCountOfDrillRegions = 2;
sts = EssAlloc (hInst, sizeof(ESS_STR_T) * url.iCountOfDrillRegions, &url.cppDrillRegions);
url.cppDrillRegions[0] = @idesc("Qtr1")
url.cppDrillRegions[1] = @idesc("Qtr2")
sts = EssCreateDrillThruURL(hCtx, &url);
printf("EssCreateDrillThruURL sts: %ld\n",sts);

EssCreateExtGroup

Creates a group in the external user directory.

Syntax

ESS_FUNC_M EssCreateExtGroup (hCtx, GroupName);

Parameter Data Type Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle (input).</td>
</tr>
<tr>
<td>Parameter</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>GroupName</td>
<td>ESS_STR_T</td>
<td>Name of group to create (input). See “Group Name Limits” on page 1728.</td>
</tr>
</tbody>
</table>

**Notes**

The specified group must exist in Shared Services.

**Return Value**

None.

**Access**

This function requires the caller to have Create/Delete User privilege (ESS_PRIV_USERCREATE) for the logged in server.

**Example**

```c
ESS_FUNC_M ESS_CreateGroup (ESS_HCTX_T hCtx)
{
    ESS_FUNC_M sts = ESS_STS_NOERR;
    ESS_STR_T GroupName;
    ESS_STR_T Password;
    ESS_PROTOCOL_T Protocol;
    ESS_CONN_PARAM ConnParam;

    GroupName = "Group 1";
    Password = "Password";
    Protocol = CSS;
    ConnParam = NULL;

    sts = EssCreateExtGroup (hCtx, GroupName);

    return (sts);
}
```

**See Also**

- EssDeleteUser
- EssListUsers
- EssRenameUser
- EssSetPassword
- EssSetUser
- EssGetUserEx
- “ESS_USERINFOEX_T” on page 195

## EssCreateExtUser

Creates a new externally authenticated user.

**Syntax**

```c
ESS_FUNC_M EssCreateExtUser (hCtx, UserName, Password, SecurityProvider,
ProviderConnectionParameters);
```
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>UserName</td>
<td>ESS_STR_T</td>
<td>Name of user to create. See “User Name Limits” on page 1728.</td>
</tr>
<tr>
<td>Password</td>
<td>ESS_STR_T</td>
<td>Security password for new user. See “Password Limits” on page 1728.</td>
</tr>
<tr>
<td>Note:</td>
<td></td>
<td>The Password parameter has been made redundant by changes for Shared Services. You can use an empty string for this parameter.</td>
</tr>
<tr>
<td>SecurityProvider</td>
<td>ESS_PROTOCOL_T</td>
<td>The name of the external authentication mechanism.</td>
</tr>
<tr>
<td>ProviderConnectionParameters</td>
<td>ESS_CONNPARAM_T</td>
<td>Parameters used by the external authentication mechanism, if any.</td>
</tr>
</tbody>
</table>

**Notes**

- The specified user must not already exist.
- The user’s access level and other parameters may be set with the EssSetUser() function.
- Your program should ensure that the password has been entered correctly (e.g. by requiring the user to type it twice) before calling this function. Once entered, it is not possible to retrieve a password. However, a password can be changed using the EssSetPassword() function.

**Return Value**

None.

**Access**

This function requires the caller to have Create/Delete User privilege (ESS_PRIV_USERCREATE) for the logged in server.

**Example**

```c
ESS_FUNC_M ESS_CreateUser (ESS_HCTX_T   hCtx)
{
    ESS_FUNC_M     sts = ESS_STS_NOERR;
    ESS_STR_T      UserName;
    ESS_STR_T      Password;
    ESS_PROTOCOL_T Protocol;
    ESS_CONN_PARAM ConnParam;

    UserName = "Jim Smith";
    Password = "Password";
    Protocol = CSS;
    ConnParam = NULL;

    sts = EssCreateExtUser (hCtx, UserName, Password, Protocol, ConnParam);

    return (sts);
}
```
See Also

- EssDeleteUser
- EssListUsers
- EssRenameUser
- EssSetPassword
- EssSetUser
- EssGetUserEx
- “ESS_USERINFOEX_T” on page 195

## EssCreateFilter

Creates a new filter and starts setting its contents.

### Syntax

```
ESS_FUNC_M EssCreateFilter (hCtx, AppName, DbName, FilterName, Active, Access);
```

### Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle</td>
</tr>
<tr>
<td>AppName</td>
<td>ESS_STR_T</td>
<td>Application name</td>
</tr>
<tr>
<td>DbName</td>
<td>ESS_STR_T</td>
<td>Database name</td>
</tr>
<tr>
<td>FilterName</td>
<td>ESS_STR_T</td>
<td>Filter name. See “Filter Name Limits” on page 1728.</td>
</tr>
<tr>
<td>Active</td>
<td>ESS_BOOL_T</td>
<td>Filter active flag. If TRUE, the filter is set active, otherwise it is set inactive.</td>
</tr>
<tr>
<td>Access</td>
<td>ESS_ACCESS_T</td>
<td>The default filter access level</td>
</tr>
</tbody>
</table>

### Notes

- If the filter does not already exist, it will be created by this call.
- If the filter already exists, an error message is returned.
- This call must be followed by successive calls to EssSetFilterRow() to set all the rows for the filter.

### Return Value

None.

### Access

This function requires the caller to have database designer permission (ESS_PRIV_DBDESIGN) for the specified database.

### Example

```c
ESS_FUNC_M
ESS_CreateFilter (ESS_HCTX_T hCtx)
{
    ESS_FUNC_M  sts = ESS_STS_NOERR;
    ESS_STR_T  AppName;
```
ESS_STR_T  DbName;
ESS_STR_T  FilterName;
ESS_BOOL_T  Active;
ESS_ACCESS_T  Access, AccessAry[3];
ESS_STR_T  RowString[3];
ESS_USHORT_T  ind;

AppName   = "Sample";
DbName     = "Basic";
FilterName = "NewFilter";
Active     = ESS_TRUE;

/***** Create Filter *****/
sts = EssCreateFilter(hCtx, AppName, DbName,
                       FilterName, Active, Access);
if(!sts)
{
    RowString[0] = "@DESCENDANTS(Scenario)";
    RowString[1] = "@DESCENDANTS(Product)";
    RowString[2] = "Qtr1, @DESCENDANTS("Colas")";

    AccessAry[0] = ESS_ACCESS_READ;
    AccessAry[1] = ESS_ACCESS_NONE;

    /***** Set Filter Rows *****/
    for(ind = 0; ind < 3; ind++)
    {
        sts = EssSetFilterRow(hCtx, RowString[ind],
                               AccessAry[ind]);
        if(sts)
            printf(\“Cannot set Filter row %s\r\n\”,
                   RowString[ind]);
    }
    sts = EssSetFilterRow(hCtx, "", ESS_ACCESS_NONE);
}
return (sts);

See Also
- EssGetFilter
- EssListFilters
- EssSetFilterRow
- EssSetFilter

**EssCreateGroup**

Creates a new group.

**Syntax**

```c
ESS_FUNC_M EssCreateGroup (hCtx, GroupName);
```
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>GroupName</td>
<td>ESS_STR_T</td>
<td>Name of group to create. See “Group Name Limits” on page 1728.</td>
</tr>
</tbody>
</table>

**Notes**

- The specified group must not already exist.
- The group’s access level may be set with the `EssSetGroup()` function.

**Return Value**

None.

**Access**

This function requires the caller to have Create/Delete User privilege (ESS_PRIV_USERCREATE) for the logged in server.

**Example**

```c
ESS_FUNC_M
EssCreateGroup (ESS_HCTX_T  hCtx)
{
    ESS_FUNC_M     sts = ESS_STS_NOERR;
    ESS_STR_T     GroupName;
    GroupName = "PowerUsers";

    sts = EssCreateGroup (hCtx, GroupName);

    return (sts);
}
```

**See Also**

- `EssCreateExtGroup`
- `EssDeleteGroup`
- `EssListGroups`
- `EssRenameGroup`
- `EssSetGroup`

---

**EssCreateLocalContext**

Creates a local API context for use in local API operations.

**Syntax**

```c
ESS_FUNC_M  EssCreateLocalContext (hInstance, UserName, Password, phLocalCtx);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>ESS_HINST_T</td>
<td>API instance handle.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>UserName;</td>
<td>ESS_STR_T</td>
<td>Currently not used - should be NULL.</td>
</tr>
<tr>
<td>Password;</td>
<td>ESS_STR_T</td>
<td>Currently not used - should be NULL.</td>
</tr>
<tr>
<td>phLocalCtx;</td>
<td>ESS_PHCTX_T</td>
<td>Address of variable to receive Essbase local context handle.</td>
</tr>
</tbody>
</table>

**Notes**

- This function must be called if access to local API operations (for example local file/object functions) is desired. It should be called after calling `EssInit()`.
- It is only necessary to call the function once per client application - the context handle can be used for all local API operations.
- You should call `EssDeleteLocalContext()` when the application has finished accessing local objects.

**Return Value**

If successful, a valid local context handle is returned in `phLocalCtx`.

**Access**

This function requires no special privileges.

**Example**

See the example of `EssGetLocalPath`.

**See Also**

- `EssDeleteLocalContext`
- `EssInit`

---

**EssCreateLocationAlias**

Creates a new location alias; that is, it maps an alias name string to an ordered set of the following five strings: host name, application name, database name, user login name, and user password.

**Syntax**

```c
ESS_FUNC_M EssCreateLocationAlias (hCtx, pAlias, pHost, pApp, pDb, pName, pPassword);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx;</td>
<td>ESS_HCTX_T</td>
<td>API context handle</td>
</tr>
<tr>
<td>pAlias;</td>
<td>ESS_STR_T</td>
<td>Location alias</td>
</tr>
<tr>
<td>pHost;</td>
<td>ESS_STR_T</td>
<td>Target host</td>
</tr>
<tr>
<td>pApp;</td>
<td>ESS_STR_T</td>
<td>Target application</td>
</tr>
<tr>
<td>pDb;</td>
<td>ESS_STR_T</td>
<td>Target database</td>
</tr>
</tbody>
</table>
### Parameter | Data Type | Description
--- | --- | ---
pName; | ESS_STR_T | User login name
pPassword; | ESS_STR_T | User password

**Return Value**

Returns an error if a location alias with the name `pAlias` already exists.

**See Also**

- `EssDeleteLocationAlias`
- `EssGetLocationAliasList`

---

**EssCreateObject**

Creates a new object on the server or client object system.

**Syntax**

```c
ESS_FUNC_M EssCreateObject (hCtx, ObjType, AppName, DbName, ObjName);
```

**Parameter | Data Type | Description
--- | --- | ---
hCtx | ESS_HCTX_T | API context handle. Can be local context handle returned by `EssCreateLocalContext()`.
ObjType | ESS_OBJTYPE_T | Object type (must be single type). Refer to “Bitmask Data Types (C)” on page 96 for a list of possible values.
AppName | ESS_STR_T | Application name.
DbName | ESS_STR_T | Database name. If NULL, uses the Application subdirectory.
ObjName | ESS_STR_T | Name of object to create. See “Object Name Limits” on page 1728.

**Notes**

- To create an object, it must not already exist.
- A newly created object on the server contains no data and merely acts as a place holder to prevent another user from creating the object. If you wish to update the created object, you should lock it using `EssLockObject()`, then save it using `EssPutObject()`.

**Return Value**

None.

**Access**

This function requires the caller to have Application or Database Design privilege (ESS_PRIV_APPDESIGN or ESS_PRIV_DBDESIGN) for the specified application or database to contain the object.
Example

```c
ESS_FUNC_M
ESS_CreateObject (ESS_HCTX_T hCtx)
{
    ESS_FUNC_M        sts = ESS_STS_NOERR;
    ESS_STR_T       AppName;
    ESS_STR_T       DbName;
    ESS_STR_T       ObjName;
    ESS_OBJTYPE_T   ObjType;

   AppName = "Sample";
   DbName  = "Basic";
   ObjName = "Test";
   ObjType = ESS_OBJTYPE_OUTLINE;

    sts = EssCreateObject(hCtx, ObjType, AppName,
                          DbName, ObjName);

    if(!sts)
        printf("The Object is created.\r\n");
    return (sts);
}
```

See Also

- EssCopyObject
- EssDeleteObject
- EssListObjects
- EssLockObject
- EssPutObject
- EssRenameObject

**EssCreateStorageTypedApplication**

Creates a new application with the option of data storage mode: block (multidimensional) or aggregate.

**Syntax**

```c
ESS_FUNC_M EssCreateStorageTypedApplication (hCtx, AppName, StorageType);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>AppName</td>
<td>ESS_STR_T</td>
<td>Name of application to create. See &quot;Application Name Limits&quot; on page 1727.</td>
</tr>
<tr>
<td>StorageType</td>
<td>ESS_DATA_STORAGE_T</td>
<td>The data storage type of the new application.</td>
</tr>
</tbody>
</table>

The valid values for StorageType are:

- ESS_DEFAULT_DATA_STORAGE
- ESS_MULTIDIM_DATA_STORAGE — Block storage (multidimensional), the default storage type
- ESS_ASO_DATA_STORAGE — Aggregate storage
Notes

- The new application is created in non-Unicode mode.
- Creating a client application creates a directory to contain local application files.
- A newly created database or application is not automatically set to active. Call EssSetActive() after calling EssCreateDatabase(), EssCreateApplication(), EssCreateApplicationEx(), EssCreateStorageTypedApplication(), or EssCreateStorageTypedApplicationEx() to keep subsequent functions, such as EssRestructure(), from operating on the wrong database or application (the application or database that is already active).

Return Value

Returns 0 if successful; otherwise, returns an error.

Access

For a server application, the caller must have Application Create/Delete/Edit privilege (ESS_PRIV_APPCREATE).

Example

```c
ESS_FUNC_M
ESS_CreateASOApp (ESS_HCTX_T hCtx)
{
    ESS_FUNC_M sts = ESS_STS_NOERR;
    ESS_STR_T AppName;
    AppName = "Sample";
    sts = EssCreateStorageTypedApplication (hCtx, AppName, ESS_ASO_DATA_STORAGE);
    return(sts);
}
```

See Also

- EssCreateStorageTypedApplicationEx
- EssCreateApplication
- EssCreateDatabase
- EssCreateObject

**EssCreateStorageTypedApplicationEx**

Creates a new application with options for data storage mode (block or aggregate) and application mode (Unicode or non-Unicode).

Syntax

```c
ESS_FUNC_M EssCreateStorageTypedApplicationEx (hCtx, AppName, storageType, usAppType);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>AppName</td>
<td>ESS_STR_T</td>
<td>Name of application to create. See “Application Name Limits” on page 1727.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>--------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>StorageType</td>
<td>ESS_DATA_STORAGE_T</td>
<td>The data storage type of the new application.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The valid values for StorageType are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESS_DEFAULT_DATA_STORAGE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESS_MULTIDIM_DATA_STORAGE—Block storage (multidimensional), the default storage type</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESS_ASO_DATA_STORAGE—Aggregate storage</td>
</tr>
<tr>
<td>usAppType</td>
<td>ESS_USHORT_T</td>
<td>The application type (Unicode or non-Unicode) of the new application.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The valid values for usAppType are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESS_APP_UNICODE - 0x0003—Unicode application. The function fails if the server is not in Unicode mode.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESS_APP_NONUNICODE - 0x0002—Non-Unicode application.</td>
</tr>
</tbody>
</table>

**Notes**

- Creating a client application creates a directory to contain local application files.
- A newly created database or application is not automatically set to active. Call EssSetActive() after calling EssCreateDatabase(), EssCreateApplication(), EssCreateApplicationEx(), EssCreateStorageTypedApplication(), or EssCreateStorageTypedApplicationEx() to keep subsequent functions, such as EssRestructure(), from operating on the wrong database or application (the application or database that is already active).

**Return Value**

Returns 0 if successful; otherwise, returns an error.

**Access**

For a server application, the caller must have Application Create/Delete/Edit privilege (ESS_PRIV_APPCREATE).

**Example**

```c
ESS_FUNC_M
ESS_CreateASOApp (ESS_HCTX_T  hCtx)
{
    ESS_FUNC_M          sts = ESS_STS_NOERR;
    ESS_STR_T           AppName;
    AppName = "Sample";
    sts = EssCreateStorageTypedApplicationEx (hCtx, AppName, ESS_ASO_DATA_STORAGE, ESS_APP_UNICODE);
    return(sts);
}
```

**See Also**

- EssCreateStorageTypedApplication
- EssCreateApplication
- EssCreateDatabase
- EssCreateObject
EssCreateUser

Creates a new user.

Syntax

```c
ESS_FUNC_M EssCreateUser (hCtx, UserName, Password);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>UserName</td>
<td>ESS_STR_T</td>
<td>Name of user to create. See &quot;User Name Limits&quot; on page 1728.</td>
</tr>
<tr>
<td>Password</td>
<td>ESS_STR_T</td>
<td>Security password for new user. See &quot;Password Limits&quot; on page 1728.</td>
</tr>
</tbody>
</table>

Notes

- The specified user must not already exist.
- The user’s access level and other parameters may be set with the EssSetUser() function.
- Your program should ensure that the password has been entered correctly (for example, by requiring the user to type it twice) before calling this function. Once entered, it is not possible to retrieve a password. However, a password can be changed using the EssSetPassword() function.

Return Value

None.

Access

This function requires the caller to have Create/Delete User privilege (ESS_PRIV_USERCREATE) for the logged in server.

Example

```c
ESS_FUNC_M
ESS_CreateUser (ESS_HCTX_T hCtx)
{
    ESS_FUNC_M   sts = ESS_STS_NOERR;
    ESS_STR_T    UserName;
    ESS_STR_T    Password;

    UserName = "Jim Smith";
    Password = "Password";

    sts = EssCreateUser (hCtx, UserName, Password);

    return (sts);
}
```

See Also

- EssDeleteUser
- EssListUsers
- EssRenameUser
• EssSetPassword
• EssSetUser

EssCreateVariable

Creates a new substitution variable or modifies an existing substitution variable if the variable name already exists with the identical server, application, and database values.

Syntax

```c
ESS_FUNC_M EssCreateVariable (hCtx, pVariable);
```

### Parameter Data Type Description

- **hCtx** ESS_HCTX_T API context handle.
- **pVariable** ESS_PVARIABLE_T Pointer to the structure containing the description of the substitution variable being created.

### Notes

- The scope of the variable can apply to the server, the application, or the database. The scope is controlled through the ESS_VARIABLE_T structure. When the server, application, and database are all named, the substitution variable applies only to the specified database. When only the server and application are named, the substitution variable applies to all databases in the specified application. When only the server is named, the substitution variable applies to all applications and databases on the specified server.
- When a variable exists and a new variable is created with the same name and scope, the new value replaces the old value with no error message from Essbase.
- On a given server, you can create multiple substitution variables with the same name but different scopes (application and database).

### Return Value

If successful, returns zero.

### Example

```c
/*
** ESS_CreateVariable() creates a substitution variable using
** the API EssCreateVariable, and sets its value.
*/

ESS_FUNC_M
ESS_CreateVariable (ESS_HCTX_T hCtx)
{
    ESS_FUNC_M      sts = ESS_STS_NOERR;
    ESS_VARIABLE_T Variable;
    printf("\n ******************************************");
    printf("\n **** An example of using EssCreateVariable"");
    printf("\n ******************************************");
    /* Create Variable 'QuarterName' at the level of the server/App/Db */
    strcpy(Variable.VarName,   "QuarterName");
```
strcpy(Variable.Server,  "Local");
strcpy(Variable.AppName,  "Sample");
strcpy(Variable.DbName,   "Basic");
strcpy(Variable.VarValue, "Qtr1");
sts = EssCreateVariable(hCtx, &Variable);
if (sts == ESS_STS_NOERR)
    printf("\n Variable 'QuarterName' is created at the Server/App/Db level with value 'Qtr1'\n");
/* Change Value of 'QuarterName' from Qtr1 to Qtr2 */
if (sts == ESS_STS_NOERR)
{
    strcpy(Variable.VarName,  "QuarterName");
    strcpy(Variable.Server,   "Local");
    strcpy(Variable.AppName,  "Sample");
    strcpy(Variable.DbName,   "Basic");
    strcpy(Variable.VarValue, "Qtr2");
    sts = EssCreateVariable(hCtx, &Variable);
    if (sts == ESS_STS_NOERR)
        printf("\n Variable 'QuarterName' at the Server/App/Db level is updated to value 'Qtr2'\n");
}
/* Create Variable 'MarketName' at the level of the Server/App */
if (sts == ESS_STS_NOERR)
{
    strcpy(Variable.VarName,  "MarketName");
    strcpy(Variable.Server,   "Local");
    strcpy(Variable.AppName,  "Sample");
    strcpy(Variable.DbName,   ":");
    strcpy(Variable.VarValue, "East");
    sts = EssCreateVariable(hCtx, &Variable);
    if (sts == ESS_STS_NOERR)
        printf("\n Variable 'MarketName' is created at the Server/App level\n");
}
/* Create Variable 'MarketName' at the level of the Server */
/* This shows that you can have the same variable name at different levels*/
if (sts == ESS_STS_NOERR)
{
    strcpy(Variable.VarName,  "MarketName");
    strcpy(Variable.Server,   "Local");
    strcpy(Variable.AppName,  ":");
    strcpy(Variable.DbName,   ":");
    strcpy(Variable.VarValue, "Market");
    sts = EssCreateVariable(hCtx, &Variable);
    if (sts == ESS_STS_NOERR)
        printf("\n Variable 'MarketName' is created at the Server level\n");
}
if (sts == ESS_STS_NOERR)
    printf("\n --> No Errors in EssCreateVariable\n\n\n");
else
    printf("\n --> Error in EssCreateVariable number: %d\n\n\n", sts);
return (sts);
} /* end ESS_CreateVariable */

Output
An example of using EssCreateVariable

Variable 'QuarterName' is created at the Server/App/Db level with value 'Qtr1'
Variable 'QuarterName' at the Server/App/Db level is updated to value 'Qtr2'
Variable 'MarketName' is created at the Server/App level
Variable 'MarketName' is created at the Server level
--> No Errors in EssCreateVariable

See Also
- “ESS_VARIABLE_T” on page 197
- EssDeleteVariable
- EssGetVariable
- EssListVariables

EssDefaultCalc

Executes the default calculation for the active database.

Syntax

ESS_FUNC_M EssDefaultCalc (hCtx);

Parameter | Data Type | Description
---|---|---
hCtx | ESS_HCTX_T | API context handle.

Notes
- If this function succeeds and the calculation is started, it will continue on the server as an asynchronous process after the return from this call. The caller must check at regular intervals to see if the process has completed by calling EssGetProcessState() until it returns ESS_STATE_DONE().
- To get and set the default calc script, use the functions EssGetDefaultCalc(), EssSetDefaultCalc() and EssSetDefaultCalcFile().

Return Value

None.

Access

This function requires the caller to have calc privilege (ESS_PRIV_CALC) to the active database.

Example

ESS_FUNC_M
ESS_CalcDefault (ESS_HCTX_T hCtx)
{
  ESS_FUNC_M stSts = ESS_STS_NOERR;
  ESS_PROCSTATE_T pState;

  stSts = EssDefaultCalc(hCtx);
  if (!stSts)
{  
    sts = EssGetProcessState (hCtx, &pState);
    while (!sts && (pState.State != 
        ESS_STATE_DONE))  
        sts = EssGetProcessState (hCtx,  
            &pState);
}

return (sts);
}

See Also

- EssBeginCalc
- EssCalc
- EssGetDefaultCalc
- EssSetDefaultCalc
- EssSetDefaultCalcFile

**EssDeleteAllSplFiles**

Deletes all trigger logfiles for a database.

**Syntax**

```c
ESS_FUNC_M EssDeleteAllSplFiles (hCtx,AppName,DbName);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>AppName</td>
<td>ESS_STR_T</td>
<td>Application name.</td>
</tr>
<tr>
<td>DbName</td>
<td>ESS_STR_T</td>
<td>Database name.</td>
</tr>
</tbody>
</table>

**Access**

This function requires the caller to have database Design privilege (ESS_PRIV_DBDESIGN) for the specified database.

**See Also**

- EssDisplayTriggers
- EssListSpoolFiles
- EssGetSpoolFile
- EssDeleteSpoolFile
- EssMdxTrig

**EssDeleteApplication**

Deletes an existing application, either on the client or the server. If the application is running on the server, then it is first stopped.
Syntax

ESS_FUNC_M EssDeleteApplication (hCtx, AppName);

Parameter | Data Type | Description
---|---|---
hCtx | ESS_HCTX_T | API context handle.
AppName | ESS_STR_T | Name of application to delete.

Notes
Deleting a client application removes the local application directory and contents. It also removes all objects stored with the application, including all databases.

Return Value
None.

Access
For a server application, the caller must have Application Create/Delete/Edit privilege (ESS_PRIV_APPCREATE).

Example

ESS_FUNC_M
ESS_DeleteApp (ESS_HCTX_T hCtx)
{
    ESS_FUNC_M       sts = ESS_STS_NOERR;
    ESS_STR_T        AppName;
    AppName =  "Sample";

    sts = EssDeleteApplication (hCtx, AppName);

    return(sts);
}

See Also

- EssDeleteDatabase
- EssDeleteObject

EssDeleteDatabase

Deletes an existing database from an application, either on the client or the server. If the database is running on the server, then it is first stopped.

Syntax

ESS_FUNC_M EssDeleteDatabase (hCtx, AppName, DbName);

Parameter | Data Type | Description
---|---|---
hCtx | ESS_HCTX_T | API context handle.
### Parameter | Data Type | Description
---|---|---
AppName | ESS_STR_T | Name of application containing database.
DbName | ESS_STR_T | Name of database to delete.

**Notes**
- Deleting a client database removes the local database directory and contents.
- Deleting a server database removes all objects associated with that database.

**Return Value**
None.

**Access**
For a server database, the caller must have database Create/Delete/Edit privilege (ESS_PRIV_DBCREATE).

**Example**
```c
ESS_FUNC_M
ESS_DeleteDb (ESS_HCTX_T   hCtx)
{
    ESS_FUNC_M     sts = ESS_STS_NOERR;
    ESS_STR_T     AppName;
    ESS_STR_T     DbName;
    AppName = "Sample";
    DbName  = "Basic";

    /* IF the current active is the same as the
    * unload db, ClearActive first */
    sts = EssClearActive(hCtx);

    /* ELSE */
    sts = EssDeleteDatabase(hCtx, AppName, DbName);
    return (sts);
}
```

**See Also**
- [EssDeleteApplication](#)
- [EssDeleteObject](#)

## EssDeleteDrillThruURL

Deletes a drill-through URL, with the given URL name, within the active database outline.

### Syntax
```c
ESS_FUNC_M EssDeleteDrillThruURL (hCtx, URLName);
```
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle</td>
</tr>
<tr>
<td>URLName</td>
<td>ESS_STR_T</td>
<td>Drill-through URL name</td>
</tr>
</tbody>
</table>

**Return Value**

- If successful, deletes the named drill-through URL in the active database outline.
- If unsuccessful, returns an error code.

**Access**

- Caller must have database Design privilege (ESS_PRIV_DBDESIGN) for the specified database.
- Caller must have selected the specified database as the active database using EssSetActive().

**Example**

```c
ESS_STS_T sts = ESS_STS_NOERR;

sts = EssDeleteDrillThruURL(hCtx, "Drill Through to EPMI");
printf("EssDeleteDrillThruURL sts: %ld\n", sts);
```

---

**EssDeleteFilter**

Deletes an existing filter.

**Syntax**

```c
ESS_FUNC_M EssDeleteFilter (hCtx, AppName, DbName, FilterName);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle</td>
</tr>
<tr>
<td>AppName</td>
<td>ESS_STR_T</td>
<td>Application name</td>
</tr>
<tr>
<td>DbName</td>
<td>ESS_STR_T</td>
<td>Database name</td>
</tr>
<tr>
<td>FilterName</td>
<td>ESS_STR_T</td>
<td>Filter name</td>
</tr>
</tbody>
</table>

**Return Value**

None.

**Access**

This function requires the caller to have database Design privilege (ESS_PRIV_DBDESIGN) for the specified database.

**Example**

```c
ESS_FUNC_M
EssDeleteFilter (ESS_HCTX_T  hCtx)
```
```c
ESS_FUNC_M    sts = ESS_STS_NOERR;
ESS_STR_T    AppName;
ESS_STR_T     DbName;
ESS_STR_T     FilterName;

AppName    = "Sample";
DbName     = "Basic";
FilterName = "Test";

sts = EssDeleteFilter(hCtx, AppName, DbName,
                        FilterName);
return (sts);
}

See Also
● EssCopyFilter
● EssListFilters
● EssRenameFilter
● EssSetFilter

EssDeleteFromGroup

Removes a user from the list of group members.

Syntax

ESS_FUNC_M EssDeleteFromGroup (hCtx, GroupName, UserName);

Parameter      Data Type     Description
hCtx            ESS_HCTX_T   API context handle.
GroupName       ESS_STR_T    Group name.
UserName        ESS_STR_T    Name of user to remove from group list.

Notes

As well as deleting the specified user from the list of members for the specified group, this
function also deletes the group from the user’s own list of associated groups.

Return Value

None.

Access

This function requires the caller to have Create/Delete User privilege
(ESS_PRIV_USERCREATE) for the logged in server.

Example

ESS_FUNC_M
ESS_RemoveUser (ESS_HCTX_T    hCtx)
{
ESS_FUNC_M     sts = ESS_STS_NOERR;
ESS_STR_T     GroupName;
ESS_STR_T     UserName;
GroupName = "PowerUsers";
UserName  = "Jim Smith";

    sts = EssDeleteFromGroup (hCtx, GroupName, UserName);

    return (sts);
}

See Also

●      EssDeleteFromGroupEx
●      EssAddToGroup
●      EssGetGroupList
●      EssListGroups
●      EssSetGroupList

EssDeleteFromGroupEx

Removes a user from a group. Similar to EssDeleteFromGroup, but can accept a user directory specification or unique identity attribute for GroupId or UserId.

Syntax

ESS_FUNC_M EssDeleteFromGroupEx (hCtx, GroupId, bIsGroupId, UserId, bUsingIdentity);

Parameter    Data Type    Description
hCtx          ESS_HCTX_T    API context handle (input).
GroupId       ESS_STR_T     Group name or identity (input). Can be specified as groupname@provider or as a unique identity attribute.
bIsGroupId    ESS_BOOL_T    Input. Indicates if GroupId is a name or an identity. If TRUE, GroupId is an identity.
UserId        ESS_STR_T     Name of user to remove from group (input). Can be specified as username@provider or as a unique identity attribute.
bUsingIdentity ESS_BOOL_T    Input. Indicates if UserID is a name or an identity. If TRUE, UserID is an identity.

Notes

As well as deleting the specified user from the list of members for the specified group, this function also deletes the group from the user’s own list of associated groups.

Return Value

None.

Access

This function requires the caller to have Create/Delete User privilege (ESS_PRIV_USERCREATE) for the logged in server.
**Example**

```c
void DisplayUserList(ESS_USHORT_T count, ESS_PSTR_T UserList) {
    ESS_USHORT_T i;

    for (i = 0; i < count; i++) {
        if (UserList[i])
            printf ("%s\n", UserList[i]);
    }
}

ESS_FUNC_M ESS_RemoveUser (ESS_HCTX_T hCtx) {
    ESS_STS_T sts  = ESS_STS_NOERR;
    ESS_STR_T groupId, userId;
    ESS_BOOL_T bGroupId, bUserId;
    ESS_BOOL_T bisIdentity;
    ESS_USHORT_T type;
    ESS_USHORT_T count;
    ESS_BOOL_T bUsingIdentity;
    ESS_PSTR_T pUserList;

    groupId = "IDRegularGroup@ldap";
    userId = "IDUser8@ldap";
    bGroupId = ESS_FALSE;
    bUserId = ESS_TRUE;
    sts = EssDeleteFromGroupEx (hCtx, groupId, bGroupId, userId, bUserId);
    printf("EssDeleteFromGroupEx sts: %ld\n", sts);
    if(!sts)
    {
        sts = EssGetGroupListEx(hCtx, groupId, bisIdentity, type, &count, &bUsingIdentity,
                        &pUserList);
        printf("EssGetGroupListEx sts: %ld\n", sts);
        if(!sts)
        {
            if(pUserList)
            {
                printf ("\n---User/Group list for %s:\n", groupId);
                DisplayUserList(count, pUserList);
            }
            else
                printf("\tUser list is empty\n");
        }
    }

    return (sts);
}
```

**See Also**

- `EssAddToGroupEx`
- `EssGetGroupListEx`
- `EssListGroupsInfoEx`
**EssDeleteGroup**

Deletes an existing group.

**Syntax**

```c
ESS_FUNC_M EssDeleteGroup (hCtx, GroupName);
```

**Parameter**  
**Data Type**  
**Description**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>GroupName</td>
<td>ESS_STR_T</td>
<td>Name of group to delete.</td>
</tr>
</tbody>
</table>

**Return Value**

None.

**Access**

This function requires the caller to have Create/Delete User privilege (ESS_PRIV_USERCREATE) for the logged in server.

**Example**

```c
ESS_FUNC_M
ESS_DeleteGroup (ESS_HCTX_T hCtx)
{
    ESS_FUNC_M     sts = ESS_STS_NOERR;
    ESS_STR_T     GroupName;
    GroupName = "PowerUsers";

    sts = EssDeleteGroup (hCtx, GroupName);

    return (sts);
}
```

**See Also**

- EssDeleteGroupEx
- EssCreateGroup
- EssListGroup
- EssRenameGroup

**EssDeleteGroupEx**

Deletes an existing group. Similar to EssDeleteGroup, but can accept a user directory specification or unique identity attribute for GroupID.

**Syntax**

```c
ESS_FUNC_M EssDeleteGroupEx (hCtx, GroupId, bisIdentity );
```

**Parameter**  
**Data Type**  
**Description**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle (input).</td>
</tr>
</tbody>
</table>
**Parameter** | **Data Type** | **Description**
---|---|---
GroupId | ESS_STR_T | Name of group to delete (input). Can be specified as `groupname@provider` or as a unique identity attribute.
bIsIdentity | ESS_BOOL_T | Input. Indicates if `GroupId` is a name or an identity. If TRUE, `GroupId` is an identity.

**Return Value**
None.

**Access**
This function requires the caller to have Create/Delete User privilege (ESS_PRIV_USERCREATE) for the logged in server.

**Example**

```c
ESS_FUNC_M ESS_DeleteGroupEx (ESS_HCTX_T hCtx)
{
    ESS_STS_T sts = ESS_STS_NOERR;
    ESS_STR_T groupId;
    ESS_BOOL_T bIsIdentity;

    groupId = "IDTempGroup1@ldap";
    bIsIdentity = ESS_FALSE;
    sts = EssDeleteGroupEx(hCtx, groupId, bIsIdentity);
    printf("EssDeleteGroupEx sts: %ld\n", sts);

    return (sts);
}
```

**See Also**
- EssDeleteUserEx

## EssDeleteLocalContext

Releases a local context previously created by `EssCreateLocalContext()`.

**Syntax**

```
ESS_FUNC_M EssDeleteLocalContext (hLocalCtx);
```

**Parameter** | **Data Type** | **Description**
---|---|---
hLocalCtx | ESS_HCTX_T | API local context handle.

**Notes**
This function should only be used for local contexts. For login contexts, use the `EssLogout()` function.
Return Value
None.

Access
This function requires no special privileges.

Example
See the example of EssGetLocalPath.

See Also
● EssCreateLocalContext
● EssLogout
● EssTerm

EssDeleteLocationAlias
Deletes an existing location alias.

Syntax
ESS_FUNC_M EssDeleteLocationAlias (hCtx, pAlias);

Parameter Data Type Description
hCtx; ESS_HCTX_T API context handle
pAlias; ESS_STR_T Location alias

Return Value
Returns an error if a location alias with the name pAlias is not found.

See Also
● EssCreateLocationAlias
● EssGetLocationAliasList

EssDeleteLogFile
Deletes an application log file or the Essbase Server log file (essbase.log) on the server.

Syntax
ESS_FUNC_M EssDeleteLogFile (hCtx, AppName);

Parameter Data Type Description
hCtx ESS_HCTX_T API context handle.
AppName ESS_STR_T Application name or NULL. If NULL, EssDeleteLogFile deletes the Essbase Server log file (essbase.log).
Notes

- Use `EssGetLogFile()` to view message logs.
- For the location of `essbase.log`, see the Oracle Essbase Database Administrator’s Guide.

Return Value

None.

Access

The caller must have Application Designer privilege (ESS_PRIV_APPDESIGN) for the specified application.

Example

```c
ESS_FUNC_M
EssDeleteLogFile (ESS_HCTX_T hCtx)
{
    ESS_FUNC_M sts = ESS_STS_NOERR;
    ESS_STR_T AppName;
    AppName = "Sample";

    sts = EssDeleteLogFile (hCtx, AppName);
    return(sts);
}
EssDeleteLogFile ("")  //Deletes Agent log file.
```

See Also

- `EssGetLogFile`
- `EssLogSize`
- `EssWriteToLogFile`

---

EssDeleteObject

Deletes an existing object.

Syntax

```c
ESS_FUNC_M EssDeleteObject (hCtx, ObjType, AppName, DbName, ObjName);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>ObjType</td>
<td>ESS_OBJTYPE_T</td>
<td>Object type (must be single type).</td>
</tr>
<tr>
<td>AppName</td>
<td>ESS_STR_T</td>
<td>Application name.</td>
</tr>
<tr>
<td>DbName</td>
<td>ESS_STR_T</td>
<td>Database name. If NULL, uses the application subdirectory.</td>
</tr>
<tr>
<td>ObjName</td>
<td>ESS_STR_T</td>
<td>Object name to delete.</td>
</tr>
</tbody>
</table>

Notes

- To delete an object, the object must not be locked.
Outline objects cannot be deleted. Use the **EssDeleteDatabase()** function to delete a database, including its associated outline.

**Return Value**

None.

**Access**

This function requires the caller to have Application or Database Design privilege (ESS_PRIV_APPDESIGN or ESS_PRIV_DBDESIGN) for the specified application or database containing the object.

**Example**

```c
ESS_FUNC_M
EssDeleteObject (ESS_HCTX_T hCtx)
{
    ESS_FUNC_M         sts = ESS_STS_NOERR;
    ESS_STR_T         AppName;
    ESS_STR_T         DbName;
    ESS_STR_T         ObjName;
    ESS_OBJTYPE_T     ObjType;
    AppName = "Sample";
    DbName  = "Basic";
    ObjName = "Test";
    ObjType = ESS_OBJTYPE_TEXT;
    sts = EssDeleteObject(hCtx, ObjType, AppName,
                           DbName, ObjName);
    return (sts);
}
```

**See Also**

- **EssCreateObject**
- **EssListObjects**

### EssDeleteSplFile

Deleting a specific trigger logfile for a database.

**Syntax**

```c
ESS_FUNC_M EssDeleteSplFile (hCtx,AppName,DbName,SplName);
```

**Parameter** | **Data Type** | **Description**
---|---|---

hCtx | ESS_HCTX_T | API context handle.
AppName | ESS_STR_T | Application name.
DbName | ESS_STR_T | Database name.
SplName | ESS_STR_T | The name of the spool file to delete.
Access

This function requires the caller to have database design privilege (ESS_PRIV_DBDESIGN) for
the specified database.

See Also

- EssDisplayTriggers
- EssListSpoolFiles
- EssGetSpoolFile
- EssDeleteAllSpoolFiles
- EssMdxTrig

**EssDeleteUser**

Deletes a user.

Syntax

```c
ESS_FUNC_M EssDeleteUser (hCtx, UserName);
```

Parameter | Data Type | Description
--- | --- | ---
 hCtx | ESS_HCTX_T | API context handle.
 UserName | ESS_STR_T | Name of user to delete.

Notes

The caller may not delete itself nor the last Administrator on the server.

Return Value

None.

Access

This function requires the caller to have Create/Delete User privilege
(ESS_PRIV_USERCREATE) for the logged in server.

Example

```c
ESS_FUNC_M
ESS_DeleteUser (ESS_HCTX_T hCtx)
{
    ESS_FUNC_M sts = ESS_STS_NOERR;
    ESS_STR_T UserName;
    UserName = "Jim Smith";

    sts = EssDeleteUser (hCtx, UserName);

    return (sts);
}
```

See Also

- EssDeleteUserEx
EssDeleteUserEx

Deletes a user. Similar to `EssDeleteUser`, but can accept a user directory specification or unique identity attribute.

**Syntax**

```c
ESS_FUNC_M EssDeleteUserEx (hCtx, UserId, bIsIdentity);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>Context handle (input).</td>
</tr>
<tr>
<td>UserId</td>
<td>ESS_STR_T</td>
<td>Name of user to delete (input). Can be specified as username@provider or as a unique identity attribute.</td>
</tr>
<tr>
<td>bIsIdentity</td>
<td>ESS_BOOL_T</td>
<td>Input. Indicates if UserId is a name or an identity. If TRUE, UserId is an identity.</td>
</tr>
</tbody>
</table>

**Notes**

The caller may not delete itself nor the last Administrator on the server.

**Return Value**

None.

**Access**

This function requires the caller to have Create/Delete User privilege (ESS_PRIV_USERCREATE) for the logged in server.

**Example**

```c
ESS_FUNC_M ESS_DeleteUserEx (ESS_HCTX_T hCtx)
{
    ESS_FUNC_M sts = ESS_STS_NOERR;
    ESS_STR_T userId;
    ESS_BOOL_T bIsIdentity;

    userId = "IDUser3@ldap";
    bIsIdentity = ESS_FALSE;
    sts = EssDeleteUserEx(hCtx, userId, bIsIdentity);
    printf("EssDeleteUserEx sts: %ld\n", sts);

    return (sts);
}
```

**See Also**

- `EssDeleteGroupEx`
EssDeleteVariable

Deletes a substitution variable.

**Syntax**

```c
ESS_FUNC_M EssDeleteVariable (hCtx, pVariable);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>Context handle to the API.</td>
</tr>
<tr>
<td>pVariable</td>
<td>&quot;ESS_VARIABLE_T&quot; on page 197</td>
<td>The pointer to the structure containing the description of the substitution variable being deleted.</td>
</tr>
</tbody>
</table>

**Return Value**

If successful, returns zero.

**Example**

```c
/*
 ** Ess_DeleteVariable() deletes a substitution variable using
 ** the API EssDeleteVariable.
 */

ESS_FUNC_M

ESS_DeleteVariable (ESS_HCTX_T hCtx)
{
    ESS_FUNC_M       sts = ESS_STS_NOERR;
    ESS_VARIABLE_T  Variable;
    ESS_PVARIABLE_T pVariables;
    ESS_ULONG_T     ulCount, i;
    printf("\n ******************************************");
    printf("\n **** An example of using EssDeleteVariable");
    printf("\n ******************************************");

    strcpy(Variable.VarName, "QuarterName");
    strcpy(Variable.Server,  "Local");
    strcpy(Variable.AppName, "Sample");
    strcpy(Variable.DbName,  "Basic");
    sts = EssDeleteVariable(hCtx, &Variable);
    if (sts == ESS_STS_NOERR)
        printf("\n Variable 'QuarterName' at the Server/App/Db level is deleted");
    if (sts == ESS_STS_NOERR)
    {
        strcpy(Variable.VarName, "MarketName");
        strcpy(Variable.Server,  "Local");
        strcpy(Variable.AppName, "Sample");
        strcpy(Variable.DbName,  "");
        sts = EssDeleteVariable(hCtx, &Variable);
        if (sts == ESS_STS_NOERR)
            printf("\n Variable 'MarketName' at the Server/App level is deleted");
    }
    if (sts == ESS_STS_NOERR)
    {
        strcpy(Variable.VarName, "MarketName");
        strcpy(Variable.Server,  "Local");
```
strcpy(Variable.AppName, "");
strcpy(Variable.DbName, "");
sts = EssDeleteVariable(hCtx, &Variable);
if (sts == ESS_STS_NOERR)
    printf("\n Variable 'MarketName' at the Server level is deleted\n");

/**********************************************************/
/* List the variables at the level of the Server/App/Db- */
/* We should not have any                                */
/**********************************************************/
if (sts == ESS_STS_NOERR)
{
    strcpy(Variable.Server, "local");
    strcpy(Variable.AppName, "Sample");
    strcpy(Variable.DbName, "Basic");
    sts = EssListVariables(hCtx, &Variable, &ulCount, &pVariables);
    if (sts == ESS_STS_NOERR)
    {
        printf("\n--- Number of Substitution Variables at the Server, App and Db 
level is: %ld\n", ulCount);
        for (i = 0; i < ulCount; i++)
        {
            printf("Variable name    : %s\n", pVariables[i].VarName);
            printf("Server name      : %s\n", pVariables[i].Server);
            printf("Application name : %s\n", pVariables[i].AppName);
            printf("Database name    : %s\n", pVariables[i].DbName);
            printf("Variable value   : %s\n\n", pVariables[i].VarValue);
        }
    }
}
/**********************************************************/
/* List variables at the server level                    */
/**********************************************************/
if (sts == ESS_STS_NOERR)
{
    strcpy(Variable.Server, "local");
    strcpy(Variable.AppName, "Sample");
    strcpy(Variable.DbName, "");
    sts = EssListVariables(hCtx, &Variable, &ulCount, &pVariables);
    if (sts == ESS_STS_NOERR)
    {
        printf("\n--- Number of Substitution Variables at the Server and App 
level is: %ld\n", ulCount);
        for (i = 0; i < ulCount; i++)
        {
            printf("Variable name    : %s\n", pVariables[i].VarName);
            printf("Server name      : %s\n", pVariables[i].Server);
            printf("Application name : %s\n", pVariables[i].AppName);
            printf("Database name    : %s\n", pVariables[i].DbName);
            printf("Variable value   : %s\n\n", pVariables[i].VarValue);
        }
    }
}
if (sts == ESS_STS_NOERR)
{
    strcpy(Variable.Server, "local");
    strcpy(Variable.AppName, "");
    strcpy(Variable.DbName, "");
    sts = EssListVariables(hCtx, &Variable, &ulCount, &pVariables);
    if (sts == ESS_STS_NOERR)
    {
        printf("--- Number of Substitution Variables at the Server level is:
            %ld\n", ulCount);
        for (i = 0; i < ulCount; i++)
        {
            printf("Variable name    : %s\n",   pVariables[i].VarName);
            printf("Server name      : %s\n",   pVariables[i].Server);
            printf("Application name : %s\n",   pVariables[i].AppName);
            printf("Database name    : %s\n",   pVariables[i].DbName);
            printf("Variable value   : %s\n\n", pVariables[i].VarValue);
        }
    }
    if (sts == ESS_STS_NOERR)
        printf(" --> No Errors in EssDeleteVariable


");
    else
        printf(" --> Error in EssDeleteVariable number: %d\n

", sts);
    return (sts);
} /* end ESS_DeleteVariable */

Output

*****************************************************************************
*** An example of using EssDeleteVariable
*****************************************************************************
Variable 'QuarterName' at the Server/App/Db level is deleted
Variable 'MarketName' at the Server/App level is deleted
Variable 'MarketName' at the Server level is deleted
--- Number of Substitution Variables at the Server, App and Db level is: 0
--- Number of Substitution Variables at the Server and App level is: 0
--- Number of Substitution Variables at the Server level is: 0
--> No Errors in EssDeleteVariable

See Also

- “ESS_VARIABLE_T” on page 197
- EssCreateVariable
- EssGetVariable
- EssListVariables

EssDisplayAlias

Dumps the contents of an alias table in the active database.

Syntax

ESS_FUNC_M EssDisplayAlias (hCtx, AliasName, pCount, ppAliases);
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>AliasName</td>
<td>ESS_STR_T</td>
<td>Name of alias table.</td>
</tr>
<tr>
<td>pCount</td>
<td>ESS_PUSHORT_T</td>
<td>Address of variable to receive count of aliases.</td>
</tr>
<tr>
<td>ppAliases</td>
<td>“ESS_MBRALT_T” on page 148</td>
<td>Address of pointer to receive member alias table.</td>
</tr>
</tbody>
</table>

**Notes**

- The memory allocated for `ppAliases` should be freed using `EssFree()`.  
- Windows only: The information returned by this command can exceed the ability of Windows to allocate memory. The Windows memory limit is 32 K.

**Return Value**

None.

**Access**

This function requires the caller to have at least read access (`ESS_PRIV_READ`) to the database, and to have selected it as their active database using `EssSetActive()`.

**Example**

```c
ESS_FUNC_M
ESS_DisplayAlias (ESS_HCTX_T hCtx)
{
    ESS_FUNC_M stst = ESS_STS_NOERR;
    ESS_USHORT_T Count;
    ESS_USHORT_T ind;
    ESS_PMBRALT_T Altlist;
    ESS_STR_T AltName;

    AltName = "TestAlias";

    stst = EssDisplayAlias (hCtx, AltName, &Count, &Altlist);
    if (Count)
    {
        printf ("\r\n-----Alias Contents-----\r\n\r\n");

        for (ind = 0; ind < Count; ind++)
        {
            printf ("%s==>%s\r\n",
                    Altlist [ind].MbrName, Altlist [ind].AltName);
        }
        printf ("\r\n");
    }

    return (stst);
}
```

**See Also**

- `EssListAliases`
**EssDisplayTriggers**

Lists all triggers for a database.

**Syntax**

```c
ESS_FUNC_M EssDisplayTriggers (hCtx, AppName, DbName, pszTrg, pCount, ppTriggerList);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>AppName</td>
<td>ESS_STR_T</td>
<td>Application name.</td>
</tr>
<tr>
<td>DbName</td>
<td>ESS_STR_T</td>
<td>database name</td>
</tr>
<tr>
<td>pszTrg</td>
<td>ESS_STR_T</td>
<td>The name of the specific trigger to return information for. If pszTrg is &quot;&quot; (empty string), then all triggers in the specified database will be returned.</td>
</tr>
<tr>
<td>pCount</td>
<td>ESS_PUSHORT_T</td>
<td>Address of the variable to receive the number of triggers for which information will be returned.</td>
</tr>
<tr>
<td>ppTriggerList</td>
<td>ESS_PPTRIGGERINFO_T</td>
<td>Address of pointer to receive an allocated array of trigger information structures. The trigger information structure includes each trigger name, the trigger definition, and a boolean field indicating whether the trigger is enabled.</td>
</tr>
</tbody>
</table>

**Notes**

The memory allocated for `ppTriggerList` should be freed using `EssFree()`.

**Return Value**

If successful, returns the count of trigger in the database in `pCount`, and an array of trigger names in `ppTriggerList`.

**Access**

This function requires the caller to have database Design privilege (ESS_PRIV_DBDESIGN) for the specified database.

**See Also**

- EssListSpoolFiles
- EssGetSpoolFile
- EssDeleteAllSplFiles
- EssDeleteSplFile
- EssMdxTrig

---

**EssDTAPIClose**

Ends the drill-through session.

**Syntax**

```c
ESS_FUNC_M EssDTAPIClose (pDTAPIInst);
```
### Parameter Data Type Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pDTAPIInst;</td>
<td>ESS_PDTAPIHINST_T</td>
<td>Initialized drill-through instance handle for the given range of data cells</td>
</tr>
</tbody>
</table>

#### Notes

- **EssDTAPIExit**() closes the drill-through session, and frees up memory.
- **EssDTAPIClose**() closes the drill-through session, but does not free up memory.

#### See Also

- Chapter 6, “Using the C Main API”
- “C Main API Structures” on page 114
- **EssDTAPIConnect**
- **EssDTAPIExecuteReport**
- **EssDTAPIExit**
- **EssDTAPIGetColumns**
- **EssDTAPIGetData**
- **EssDTAPIGetError**
- **EssDTAPIGetInfo**
- **EssDTAPIGetReports**
- **EssDTAPIInit**
- **EssDTAPISetConnection**
- **EssDTAPISetInfo**

### EssDTAPIConnect

Establishes a connection to Essbase Studio for the given drill-through instance handle.

#### Syntax

```c
ESS_FUNC_M EssDTAPIConnect (pDTAPIInst);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pDTAPIInst;</td>
<td>ESS_DTAPIHINST_T</td>
<td>Initialized drill-through instance handle for the given range of data cells</td>
</tr>
</tbody>
</table>

#### See Also

- Chapter 6, “Using the C Main API”
- “C Main API Structures” on page 114
- **EssDTAPIClose**
- **EssDTAPIExecuteReport**
- **EssDTAPIExit**
- **EssDTAPIGetColumns**
- **EssDTAPIGetData**
- **EssDTAPIGetError**
- **EssDTAPIGetInfo**
- **EssDTAPIGetReports**
- **EssDTAPIInit**
EssDTAPIExecuteReport

Executes the drill-through report identified by its index to an array of report definition structures.

Syntax

ESS_FUNC_M EssDTAPIExecuteReport (pDTAPIInst, index);

Parameter | Data Type   | Description
----------|-------------|-------------
pDTAPIInst; ESS_PDTAPIHINST_T | Initialized drill-through instance handle for the given range of data cells
index; ESS ULONG_T | Index of the report to be executed

See Also

- Chapter 6, “Using the C Main API”
- “C Main API Structures” on page 114
- EssDTAPIClose
- EssDTAPIConnect
- EssDTAPIExit
- EssDTAPIGetColumns
- EssDTAPIGetData
- EssDTAPIGetError
- EssDTAPIGetInfo
- EssDTAPIGetReports
- EssDTAPIInit
- EssDTAPISetConnection
- EssDTAPISetInfo

EssDTAPIExit

Ends the drill-through session, and frees up memory for the given drill-through instance handle.

Syntax

ESS_FUNC_M EssDTAPIExit (pDTAPIInst);

Parameter | Data Type   | Description
----------|-------------|-------------
pDTAPIInst; ESS_PDTAPIHINST_T | Initialized drill-through instance handle for the given range of data cells

Notes

- EssDTAPIExit() closes the drill-through session, and frees up memory.
- EssDTAPIClose() closes the drill-through session, but does not free up memory.
See Also

- Chapter 6, “Using the C Main API”
- “C Main API Structures” on page 114
- EssDTAPIClose
- EssDTAPIConnect
- EssDTAPIExecuteReport
- EssDTAPIGetColumns
- EssDTAPIGetData
- EssDTAPIGetError
- EssDTAPIGetInfo
- EssDTAPIGetReports
- EssDTAPIInit
- EssDTAPISetConnection
- EssDTAPISetInfo

**EssDTAPIGetColumns**

Retrieves an array of report header information structures for the given drill-through instance handle.

**Syntax**

```
ESS_FUNC_M EssDTAPIGetColumns (pDTAPIInst, ppCol, pulCount);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pDTAPIInst;</td>
<td>ESS_DTAPIHINST_T</td>
<td>Initialized drill-through instance handle for the given range of data cells</td>
</tr>
<tr>
<td>ppCol;</td>
<td>“ESS_DTAPIHEADER_T” on page 135</td>
<td>An array of report header structures for the given columns</td>
</tr>
<tr>
<td>pulCount;</td>
<td>ESS_PULONG_T</td>
<td>Number of data blocks in the ppCol report header information array</td>
</tr>
</tbody>
</table>

See Also

- Chapter 6, “Using the C Main API”
- “C Main API Structures” on page 114
- EssDTAPIClose
- EssDTAPIConnect
- EssDTAPIExecuteReport
- EssDTAPIExit
- EssDTAPIGetData
- EssDTAPIGetError
- EssDTAPIGetInfo
- EssDTAPIGetReports
- EssDTAPIInit
- EssDTAPISetConnection
- EssDTAPISetInfo
# EssDTAPIGetData

Retrieves an array of report data structures for the given drill-through instance handle.

**Syntax**

```c
ESS_FUNC_M EssDTAPIGetData (pDTAPIInst, ppData, pulRowCount, pulColCount);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pDTAPIInst;</td>
<td>ESS_DTAPIHINST_T</td>
<td>Initialized drill-through instance handle for the given range of data cells</td>
</tr>
<tr>
<td>ppData;</td>
<td>“ESS_DTAPIDATA_T” on page 134</td>
<td>An array of report data structures for the given data cells</td>
</tr>
<tr>
<td>pulRowCount;</td>
<td>ESS_PULONG_T</td>
<td>Number of rows for the data blocks in the <code>ppData</code> report data array</td>
</tr>
<tr>
<td>pulColCount;</td>
<td>ESS_PULONG_T</td>
<td>Number of columns for the data blocks in the <code>ppData</code> report data array</td>
</tr>
</tbody>
</table>

**See Also**

- Chapter 6, “Using the C Main API”
- “C Main API Structures” on page 114
- EssDTAPIClose
- EssDTAPIConnect
- EssDTAPIExecuteReport
- EssDTAPIExit
- EssDTAPIGetDataColumns
- EssDTAPIGetError
- EssDTAPIGetInfo
- EssDTAPIGetReports
- EssDTAPIInit
- EssDTAPISetConnection
- EssDTAPISetInfo

# EssDTAPIGetError

Retrieves the error status, and message.

**Syntax**

```c
ESS_FUNC_M EssDTAPIGetError (pDTAPIInst, ppData, pMsg, ulCount);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pDTAPIInst;</td>
<td>ESS_DTAPIHINST_T</td>
<td>Initialized drill-through instance handle for the given range of data cells</td>
</tr>
<tr>
<td>ppData;</td>
<td>ESS_STS_T</td>
<td>Error status</td>
</tr>
<tr>
<td>pMsg;</td>
<td>ESS_PSTR_T</td>
<td>Error message</td>
</tr>
<tr>
<td>ulCount;</td>
<td>ESS_UULONG_T</td>
<td>Size of the error message buffer</td>
</tr>
</tbody>
</table>

314  C Main API Functions
### EssDTAPIGetInfo

Retrieves drill-through connection information for a given drill-through handle.

**Syntax**

```c
ESS_FUNC_M EssDTAPIGetInfo (pDTAPIInst, pDTInfo);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pDTAPIInst</td>
<td>ESS_PDTAPIHINST_T</td>
<td>Initialized drill-through instance handle for the given range of data cells</td>
</tr>
<tr>
<td>pDTInfo</td>
<td>“ESS_DTAPIINFO_T” on page 135</td>
<td>Pointer to a structure of connection information for a given range of data cells</td>
</tr>
</tbody>
</table>

**Notes**

- Allocate memory for ESS_DTAPIINFO_T before you call EssDTAPIGetInfo().
- `sPassword` is not returned in `pDTInfo`; that is, the `sPassword` field in ESS_DTAPIINFO_T is not returned.

**See Also**

- Chapter 6, “Using the C Main API”
- “C Main API Structures” on page 114
- EssDTAPIClose
- EssDTAPIConnect
- EssDTAPIExecuteReport
- EssDTAPIExit
- EssDTAPIGetColumns
- EssDTAPIGetData
- EssDTAPIGetInfo
- EssDTAPIGetReports
- EssDTAPIInit
- EssDTAPISetConnection
- EssDTAPISetInfo
EssDTAPIGetReports

Returns the list of reports for the given drill-through instance handle.

**Syntax**

```c
ESS_FUNC_M EssDTAPIGetReports (pDTAPIInst, ppReports, pulCount);
```

**Parameter**  | **Data Type**       | **Description**                                      
---------------|---------------------|-----------------------------------------------------
`pDTAPIInst`  | ESS_DTAPIHINST_T    | Initialized drill-through instance handle for the given range of data cells 
`ppReports`   | “ESS_DTAPIREPORT_T” on page 136 | Pointer to an array of report definition structures 
`pulCount`    | ESS_PULONG_T        | Number of data blocks in `ppReports` 

**See Also**

- Chapter 6, “Using the C Main API”
- “C Main API Structures” on page 114
- EssDTAPIClose
- EssDTAPIConnect
- EssDTAPIExecuteReport
- EssDTAPIExit
- EssDTAPIGetColumns
- EssDTAPIGetData
- EssDTAPIGetError
- EssDTAPIGetInfo
- EssDTAPIInit
- EssDTAPISetConnection
- EssDTAPISetInfo

EssDTAPIInit

Starts a drill-through session, and returns a drill-through instance handle.

**Syntax**

```c
ESS_FUNC_M EssDTAPIInit (pDTAPIInit, pDTAPIInst);
```

**Parameter**  | **Data Type** | **Description**                                      
---------------|---------------|-----------------------------------------------------
`pDTAPIInit`   | ESS_PDTAPIINIT_T | (Currently not used, and set to NULL.) 
`ppDTAPIInst`  | ESS_PDTAPIHINST_T | Pointer to a drill-through initialization structure 

316  C Main API Functions
EssDTAPISetConnection

Given a connection information string, and an Extended Member Comment string, initializes a drill-through handle, and starts the Drill-Through Wizard.

Syntax

ESS_FUNC_M EssDTAPISetConnection (pDTAPIInst, pEMC, ulCount, pDTInfo);

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pDTAPIInst;</td>
<td>ESS_PDTAPIHINST_T</td>
<td>Pointer to a drill-through initialization structure</td>
</tr>
<tr>
<td>pEMC;</td>
<td>ESS_PSTR_T</td>
<td>Extended Member Comment</td>
</tr>
<tr>
<td>ulCount;</td>
<td>ESS_ULONG_T</td>
<td>Number of Extended Member Comment blocks</td>
</tr>
<tr>
<td>pDTInfo;</td>
<td>ESS_PSTR_T</td>
<td>Connection information</td>
</tr>
</tbody>
</table>

Notes

Use EssGDTRequestDrillThrough() to initialize the drill-through instance handle, because:

- EssDTAPISetConnection() currently does not initialize pDTAPIInst.
- Due to security issues, pConnection (the connection information string) and pEMC (the extended member comment string) currently are not retrieved from the Essbase Server.

See Also

- Chapter 6, “Using the C Main API”
- “C Main API Structures” on page 114
- EssDTAPIClose
EssDTAPISetInfo

Sets drill-through connection information for a given drill-through handle.

**Syntax**

```c
ESS_FUNC_M EssDTAPISetInfo (pDTAPIInst, pDTInfo);
```

**Parameter** | **Data Type** | **Description**
---|---|---
pDTAPIInst; | ESS_PDTAPIHINST_T | Initialized drill-through instance handle for a given range of data cells
pDTInfo; | “ESS_DTAPIINFO_T” on page 135 | Pointer to a structure of connection information for a given range of data cells

**Notes**
The `uInputOption` field in ESS_DTAPIINFO_T is ignored.

**See Also**
- Chapter 6, “Using the C Main API”
- “C Main API Structures” on page 114
- EssDTAPIClose
- EssDTAPIConnect
- EssDTAPIExecuteReport
- EssDTAPIExit
- EssDTAPIGetColumns
- EssDTAPIGetData
- EssDTAPIGetError
- EssDTAPIGetInfo
- EssDTAPIGetReports
- EssDTAPIInit
- EssDTAPISetConnection

EssDTClose

Eds the drill-through session for the given drill-through instance handle.
Syntax

ESS_FUNC_M EssDTClose (pDTInst);

Parameter  Data Type  Description

pDTInst;  ESS_PDTHINST_T  Initialized drill-through instance handle for the given data cell range(s)

Notes

- EssDTClose() closes the drill-through session, but does not free up memory.
- EssDTExit() closes the drill-through session, and frees up memory.

See Also

- Chapter 6, “Using the C Main API”
- “C Main API Structures” on page 114
- EssDTClose
- EssDTGetData
- EssDTGetHeader
- EssDTGetHeaderInfo
- EssDTInit
- EssDTListReports
- EssDTOpen

EssDTExit

Eds the drill-through session and frees up memory for the given drill-through instance handle.

Syntax

ESS_FUNC_M EssDTExit (pDTInst);

Parameter  Data Type  Description

pDTInst;  ESS_PDTHINST_T  Initialized drill-through instance handle for the given data cell range(s)

Notes

- EssDTExit() closes the drill-through session, and frees up memory.
- EssDTClose() closes the drill-through session, but does not free up memory.

See Also

- Chapter 6, “Using the C Main API”
- “C Main API Structures” on page 114
- EssDTClose
- EssDTGetData
- EssDTGetHeader
- EssDTGetHeaderInfo
- EssDTInit
EssDTGetData

Retrieves an array of report data for the given drill-through instance handle.

**Syntax**

```c
ESS_FUNC_M EssDTGetData (pDTInst, pData, pulCount);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pDTInst;</td>
<td>ESS_PDTHINST_T</td>
<td>Initialized drill-through instance handle</td>
</tr>
<tr>
<td>pData;</td>
<td>“ESS_DTDATA_T” on page 136</td>
<td>Array of report data structures for given data cells</td>
</tr>
<tr>
<td>pulCount;</td>
<td>ESS_PULONG_T</td>
<td>Number of data blocks in the pData header information array</td>
</tr>
</tbody>
</table>

**Notes**

Call `EssDTGetData()` until `pulCount` is 0 (zero).

**See Also**

- Chapter 6, “Using the C Main API”
- “C Main API Structures” on page 114
- EssDTClose
- EssDTExit
- EssDTGetHeader
- EssDTGetHeaderInfo
- EssDTInit
- EssDTListReports
- EssDTOpen

EssDTGetHeader

Retrieves an array of report header structures for the given drill-through instance handle.

**Syntax**

```c
ESS_FUNC_M EssDTGetHeader (pDTInst, pBuffer, pulCount);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pDTInst;</td>
<td>ESS_PDTHINST_T</td>
<td>Initialized drill-through instance handle</td>
</tr>
<tr>
<td>pBuffer;</td>
<td>“ESS_DTBUFFER_T” on page 136</td>
<td>An array of report header structures for the given columns</td>
</tr>
<tr>
<td>pulCount;</td>
<td>ESS_PULONG_T</td>
<td>Number of data blocks in the pBuffer report header information array</td>
</tr>
</tbody>
</table>
See Also

- Chapter 6, “Using the C Main API”
- “C Main API Structures” on page 114
- EssDTClose
- EssDTExit
- EssDTGetData
- EssDTGetHeader
- EssDTInit
- EssDTListReports
- EssDTOpen

**EssDTGetHeader**

Retrieves report data header information for the given drill-through instance handle.

**Syntax**

```c
ESS_FUNC_M EssDTGetHeaderInfo (pDTInst, ppHeader, pulCount);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pDTInst</td>
<td>ESS_PDTINST_T</td>
<td>Initialized drill-through instance handle</td>
</tr>
<tr>
<td>ppHeader</td>
<td>“ESS_DTHEADER_T” on page 137</td>
<td>Array of header information structures for the given drill-through instance handle</td>
</tr>
<tr>
<td>pulCount</td>
<td>ESS_PULONG_T</td>
<td>Number of blocks in the <code>ppHeader</code> header information array</td>
</tr>
</tbody>
</table>

See Also

- Chapter 6, “Using the C Main API”
- “C Main API Structures” on page 114
- EssDTClose
- EssDTExit
- EssDTGetData
- EssDTGetHeader
- EssDTInit
- EssDTListReports
- EssDTOpen
- EssDTOpen

**EssDTInit**

Starts a drill-through session, and returns a drill-through instance handle.

**Syntax**

```c
ESS_FUNC_M EssDTInit (pInit, pDTInst);
```
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pDTInit</td>
<td>ESS_PDTINIT_T</td>
<td>(Currently not used, and set to NULL.)</td>
</tr>
<tr>
<td>ppDTHInst</td>
<td>ESS_PDTINST_T</td>
<td>Pointer to a drill-through initialization structure</td>
</tr>
</tbody>
</table>

**Notes**

- EssDTInit() initializes ppDTHInst.
- Currently, pDTInit (intended for input) is not used, and is set to NULL.

**See Also**

- Chapter 6, “Using the C Main API”
- “C Main API Structures” on page 114
- EssDTClose
- EssDTExit
- EssDTGetData
- EssDTGetHeader
- EssDTGetHeaderInfo
- EssDTListReports
- EssDTOpen

**EssDTListReports**

Returns a list of report names for the given drill-through instance handle.

**Syntax**

```c
ESS_FUNC_M EssDTListReports (pDTInst, pBuffer, pulCount);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pDTInst</td>
<td>ESS_PDTHINST_T</td>
<td>Initialized drill-through instance handle.</td>
</tr>
<tr>
<td>pBuffer</td>
<td>ESS_PSTR_T</td>
<td>Array of report names for the given drill-through instance handle.</td>
</tr>
<tr>
<td>pulCount</td>
<td>ESS_PULONG_T</td>
<td>Count of blocks in the pBuffer header information array.</td>
</tr>
</tbody>
</table>

**See Also**

- Chapter 6, “Using the C Main API”
- “C Main API Structures” on page 114
- EssDTClose
- EssDTExit
- EssDTGetData
- EssDTGetHeader
- EssDTInit
- EssDTOpen
**EssDTOpen**

Given a connection information string, and an Extended Member Comment string, initializes a drill-through handle, and starts the Drill-Through Wizard.

**Syntax**

```c
ESS_FUNC_M EssDTOpen (pDTInst, pEMC, ulCount, pConnection);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pDTInst;</td>
<td>ESS_PDTINST_T</td>
<td>Pointer to a drill-through initialization structure</td>
</tr>
<tr>
<td>pEMC;</td>
<td>ESS_PSTR_T</td>
<td>Extended Member Comment</td>
</tr>
<tr>
<td>ulCount;</td>
<td>ESS_ULONG_T</td>
<td>Number of Extended Member Comment blocks</td>
</tr>
<tr>
<td>pConnection;</td>
<td>ESS_PSTR_T</td>
<td>Connection information</td>
</tr>
</tbody>
</table>

**Notes**

- EssDTOpen initializes pDTInst.
- Given an outline, and a data cell selection, pConnection (the connection information string) and pEMC (the extended member comment string) are obtained from Essbase.

**See Also**

- Chapter 6, “Using the C Main API”
- “C Main API Structures” on page 114
- EssDTClose
- EssDTExit
- EssDTGetData
- EssDTGetHeader
- EssDTGetHeaderInfo
- EssDTInit
- EssDTListReports

**EssDumpPerfStats**

Dumps performance statistics tables to a character array.

**Syntax**

```c
ESS_FUNC_M EssDumpPerfStats (hCtx, pStatBuf, [thdSN])
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx;</td>
<td>ESS_HCTX_T</td>
<td>API context handle (input)</td>
</tr>
<tr>
<td>pStatBuf;</td>
<td>ESS_STR_T</td>
<td>Pointer to the address where performance statistics tables will be dumped (input)</td>
</tr>
<tr>
<td>thdSN;</td>
<td>ESS_INT_T</td>
<td>Optional. Thread serial number from which to dump statistics (input). Default is 0 (all threads are dumped).</td>
</tr>
</tbody>
</table>
Notes

Before you call `EssDumpPerfStats()`, call `EssGetStatBufSize()` to ascertain how much memory to allocate for the performance statistics tables at the address pointed to by `pStatBuf`.

Return Value

- If successful, `EssDumpPerfStats()`
  - Returns 0.
  - Dumps performance statistics tables to a character array that begins at the address pointed to by `pStatBuf`.
- The caller of `EssDumpPerfStats()` is responsible for allocating and freeing memory at the address pointed to by `pStatBuf`.
- For more information on performance statistics tables, see the Oracle Essbase Technical Reference.

Access

The caller of this function must have supervisor access.

Example

```c
/* This function gets the array of performance stats */

ESS_STS_T ESSGetPerfStats(ESS_HCTX_T *context)
{
    ESS_STS_T    sts;
    ESS_ULONG_T  bufsize;
    ESS_PUCHAR_T poutarray; /* Pointer to the stats staging area */

    /* Get the size of the output buffer */
    if(sts = EssGetStatBufSize(context, &bufsize))
        return(sts);

    if(bufsize)
    {
        /* Allocate a staging area */
        (ESS_PVOID_T)(poutarray) = malloc (bufsize);

        /* Fill the staging area */
        sts = EssDumpPerfStats(context, poutarray);
        if(sts)
            return(sts);

        /* Do something useful with the stats here */
        /* ....................................... */

        /* Free the staging area */
        sts = EssFree(context, poutarray);
        if(sts)
            return(sts);
    }
    else
    {
        printf("Performance Statistics not enabled, call ResetPerfStats()\n");
    }
```

EssEndCalc

Marks the end of a calc script being sent to the active database. This function must be called after sending the calc script (using EssSendString()).

Syntax

**ESS_FUNC_M EssEndCalc (hCtx);**

Parameter | Data Type | Description
--- | --- | ---
hCtx | ESS_HCTX_T | API context handle.

Notes

- This function must be preceded by a call to EssBeginCalc(), and at least one call to EssSendString().
- If the calls to EssBeginCalc(), EssSendString(), and EssEndCalc succeed, the caller must check at regular intervals to see if the process has completed by calling EssGetProcessState() until it returns ESS_STATE_DONE.

Return Value

None.

Access

This function requires the caller to have calc privilege (ESS_PRIV_CALC) to the active database.

Example

```c
ESS_FUNC_M
ESS_Calc (ESS_HCTX_T hCtx)
{
    ESS_FUNC_M sts = ESS_STS_NOERR;
    ESS_STR_T Script;
    ESS_PROCSTATE_T pState;
    Script = "CALC ALL;";
    sts = EssBeginCalc (hCtx, ESS_TRUE);
    if (!sts)
        sts = EssSendString (hCtx, Script);
    if (!sts)
        sts = EssEndCalc (hCtx);
    if (!sts)
    {
        sts = EssGetProcessState (hCtx, &pState);
    }
}
```c
while (!sts && (pState.State != ESS_STATE_DONE))
    sts = EssGetProcessState (hCtx, &pState);
return(sts);
}

See Also
- EssBeginCalc
- EssCalc
- EssSendString

EssEndDataload
Marks the end of an update specification being sent to the active database, and must be called after sending the update specification using EssSendString()

Syntax

ESS_STS_T EssEndDataload (hCtx, ppMbrError);

Parameter | Data Type | Description
---|---|---
hCtx; | ESS_HCTX_T | API context handle.
ppMbrError; | "ESS_MBRERR_T" on page 148 | Pointer to the linked list of errors contained in ESS_MBRERR_T. Possible errors (and error strings) are:
- ESS_MBRERR_UNKNOWN (Unknown member [membername] in dataload, [number] records returned.)
- ESS_MBRERR_DBACCESS (You have insufficient access privilege to perform a lock on this database.)
- ESS_MBRERR_BADDATA (Invalid member [membername] in data column.)
- ESS_MBRERR_DUPLICATE (Duplicate members from the same dimension on data record, [number] records completed.)
- AD_MSGDL_ERRORLOAD (Unable to do dataload at Item/Record [number].)

Notes
- EssEndDataload() must be preceded by a call to EssBeginDataload(), and at least one call to EssSendString().
- The memory allocated for ppMbrErr must be freed using EssFreeMbrErr().

Return Value
Returns zero, if successful. Otherwise, returns an error code, as follows:
- If abortOnError is TRUE:
  - The error code for the first error condition is returned.
  - The error list is NULL.
```
If `abortOnError` is FALSE:

- An error list is returned, if the server can process the data and can continue.
- Otherwise, in exceptional circumstances, the error code explaining why the server cannot continue is returned. For example:
  - `AD_MSGDL_COLS` (too many data values in a record)
  - `AD_MSGDL_MISDIM` (data value encountered before all dimensions selected)

**Access**

`EssEndDataload()` requires the caller to have write privilege (ESS_PRIV_WRITE) to the active database.

**Example**

```c
ESS_STS_T st = ESS_STS_NOERR;
ESS_BOOL_T Store;
ESS_BOOL_T Unlock;
ESS_STR_T Query1, Query2;
ESS_PMBRERR_T pMbrErr;

Store  = ESS_TRUE;
Unlock = ESS_FALSE;
Query1 = "Year Market Scenario Measures Product 12345";
Query2 = " Jan East Scenario Measures Coke 125";

/* Begin Update */
st = EssBeginDataload (hCtx, Store, Unlock, ESS_FALSE, ESS_NULL);

/* Send update specification */
if(!st)
    st = EssSendString(hCtx, Query1);
    st = EssSendString(hCtx, Query2);

/* End Update */
if(!st)
    st = EssEndDataload(hCtx, &pMbrErr);
```

**See Also**

- `EssBeginDataload`
- `EssSendString`
- `EssBeginUpdate`
- `EssEndUpdate`
- `EssUpdate`

**EssEndIncrementalBuildDim**

Finalizes the round of building dimensions: Performs outline verification, if there is no outline verification error, it writes and closes the outline and do restructure. If the outline has errors, it writes the outline to the outline file specified by “szTmpOtlFile” and close the outline. User can use outline editing tools, such as EAS outline editor to see what is wrong in the outline.
Syntax

```c
ESS_FUNC_M EssEndIncrementalBuildDim (hCtx, restructOption, szTmpOtlFile, ErrorName, bOverwrite)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>Essbase API context handle.</td>
</tr>
<tr>
<td>restructOption</td>
<td>ESS_SHORT_T</td>
<td>Restructure option. Valid value:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_DOR_ALLDATA Keep all data</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_DOR_NODATA Discard all data</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_DOR_LOWDATA Keep all level 0 data</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_DOR_INDATA Keep all input data</td>
</tr>
<tr>
<td>szTmpOtlFile</td>
<td>ESS_STR_T</td>
<td>The temp outline file name.</td>
</tr>
<tr>
<td>ErrorName</td>
<td>ESS_STR_T</td>
<td>Name of error output file on client.</td>
</tr>
<tr>
<td>bOverwrite</td>
<td>ESS_BOOL_T</td>
<td>Indicates overwrite or append error message to the error file.</td>
</tr>
</tbody>
</table>

Return Value

Returns zero if successful; error code if unsuccessful.

Example

```c
ESS_FUNC_M
ESS_IncBuildDim( ESS_HCTX_T hCtx) {
    ESS_STS_T     sts = 0;
    ESS_OBJDEF_T  RulesObj;
    ESS_OBJDEF_T  DataObj;
    ESS_STR_T     ErrorName;
    ESS_APPNAME_T appname;
    ESS_DBNAME_T  dbname;

    memset(&RulesObj,0,sizeof(ESS_OBJDEF_T));
    memset(&DataObj,0,sizeof(ESS_OBJDEF_T));
    strcpy(appname, "sample");
    strcpy(dbname,"basic");

    RulesObj.hCtx       = hCtx;
    RulesObj.FileName   = "genref";
    RulesObj.AppName    = appname;
    RulesObj.DbName     = dbname;
    RulesObj.ObjType    = ESS_OBJTYPE_RULES;

    DataObj.hCtx        = hCtx;
    DataObj.FileName    = "genref";
    DataObj.AppName     = appname;
```

328  C Main API Functions
DataObj.DbName = dbname;
DataObj.ObjType = ESS_OBJTYPE_TEXT;

ErrorName = "builddim.err";

sts = EssBeginIncrementalBuildDim(hCtx);

if (!sts)
    sts = EssIncrementalBuildDim(hCtx,&RulesObj,&DataObj,NULL,ErrorName,true,ESS_INCDIMBUILD_BUILD,NULL);
if (!sts)
    sts = EssIncrementalBuildDim(hCtx,&RulesObj,&DataObj,NULL,ErrorName,true,ESS_INCDIMBUILD_VERIFY,NULL);
if (!sts)
    sts = EssIncrementalBuildDim(hCtx,&RulesObj,&DataObj,NULL,ErrorName,true,ESS_INCDIMBUILD_SAVEOTL,"tmpotl");

sts = EssBeginStreamBuildDim(hCtx, &RulesObj,ESS_INCDIMBUILD_BUILD,"tmpotl");
if (!sts)
    sts = EssSendString(hCtx, "600 600-20 600-20-20\n");
if (!sts)
    sts = EssSendString(hCtx, "600 600-20 600-20-30\n");
if (!sts)
    sts = EssSendString(hCtx, "600 600-40 600-40-20\n");
sts = EssEndStreamBuildDim(hCtx,ErrorName,false);

sts = EssEndIncrementalBuildDim(hCtx,ESS_DOR_ALLDATA,"tmpotl",ErrorName,false);
return sts;

See Also

- EssIncrementalBuildDim
- EssBeginIncrementalBuildDim
- EssBeginStreamBuildDim
- EssEndIncrementalBuildDim
- EssEndStreamBuildDim

**EssEndReport**

Motion must be called after sending the report specification (using EssSendString()) and before reading any returned data (using EssGetString()).

**Syntax**

```
ESS_FUNC_M EssEndReport (hCtx);
```

**Parameter** | **Data Type** | **Description**
--- | --- | ---

hCtx | ESS_HCTX_T | API context handle.
Notes

- This function must be preceded by a call to EssBeginReport(), and at least one call to EssSendString().
- If the output flag is TRUE for the call to EssBeginReport() that begins the report sequence, the call to EssEndReport() must be followed by repeated calls to EssGetString() until a NULL string is returned.

Return Value

None.

Access

This function requires the caller to have read privilege (ESS_PRIV_READ) to one or more members in the active database.

Example

```c
ESS_FUNC_M
ESS_Report (ESS_HCTX_T hCtx,
            ESS_HINST_T hInst)
{
    ESS_FUNC_M sts = ESS_STS_NOERR;
    ESS_STR_T rString = NULL;
    sts = EssBeginReport (hCtx, ESS_TRUE, ESS_FALSE);
    if (!sts)
        sts = EssSendString (hCtx, "<Desc Year !");
    if (!sts)
        sts = EssEndReport (hCtx);
    /**************
    * Get report *
    *********************/
    if (!sts)
        sts = EssGetString (hCtx, &rString);
    while (!(sts) & (rString != NULL))
    {
        printf ("%s", rString);
        EssFree (hInst, rString);
        sts = EssGetString (hCtx, &rString);
    }
    printf ("\r\n");
    return(sts);
}
```

See Also

- EssBeginReport
- EssGetString
- EssSendString
EssEndStreamBuildDim

Ends the dimension build process.

This of function must be called after EssBeginStreamBuildDim(). After calling EssBeginStreamBuildDim(), call EssSendString() to send source records to the Essbase server.

**Syntax**

```
ESS_FUNC_M EssEndStreamBuildDim (hCtx, ErrorFileName, ErFileOverWrite)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>Essbase API context handle.</td>
</tr>
<tr>
<td>ErrorFileName</td>
<td>ESS_STR_T</td>
<td>Name of error output file on client.</td>
</tr>
<tr>
<td>ErFileOverWrite</td>
<td>ESS_BOOL_T</td>
<td>Indicates overwrite or append error message to the error file.</td>
</tr>
</tbody>
</table>

**Return Value**

Returns zero if successful; error code if unsuccessful.

**Example**

```
ESS_FUNC_M
ESS_InitBuildDim( ESS_HCTX_T hCtx)
{
    ESS_STS_T     sts = 0;
    ESS_OBJDEF_T  RulesObj;
    ESS_OBJDEF_T  DataObj;
    ESS_STR_T     ErrorName;
    ESS_APPNAME_T appname;
    ESS_DBNAME_T  dbname;
    memset(&RulesObj,0,sizeof(ESS_OBJDEF_T));
    memset(&DataObj,0,sizeof(ESS_OBJDEF_T));
    strcpy(appname, "sample");
    strcpy(dbname,"basic");

    RulesObj.hCtx     = hCtx;
    RulesObj.FileName = "genref";
    RulesObj.AppName  = appname;
    RulesObj.DbName   = dbname;
    RulesObj.ObjType  = ESS_OBJTYPE_RULES;

    DataObj.hCtx      = hCtx;
    DataObj.FileName  = "genref";
    DataObj.AppName   = appname;
    DataObj.DbName    = dbname;
    DataObj.ObjType   = ESS_OBJTYPE_TEXT;

    ErrorName         = "builddim.err";

    sts = EssBeginIncrementalBuildDim(hCtx);

    if (!sts)
        sts =
```
EssIncrementalBuildDim(hCtx,&RulesObj,&DataObj,NULL,ErrorName,true,ESS_INCDIMBUILD_BUILD ,NULL);
   if (!sts)
      sts = EssIncrementalBuildDim(hCtx,&RulesObj,&DataOb,NULL,ErrorName,true,ESS_INCDIMBUILD_VERIFY ,NULL);
   if (!sts)
      sts = EssIncrementalBuildDim(hCtx,&RulesObj,&DataOb,NULL,ErrorName,true,ESS_INCDIMBUILD_SAVEOT L,"tmpotl");

   sts = EssBeginStreamBuildDim(hCtx, &RulesObj,ESS_INCDIMBUILD_BUILD,"tmpotl");
   if (!sts)
      sts = EssSendString(hCtx, "600    600-20    600-20-20\n");
   if (!sts)
      sts = EssSendString(hCtx, "600    600-20    600-20-30\n");
   if (!sts)
      sts = EssSendString(hCtx, "600    600-40    600-40-20\n");
   sts = EssEndStreamBuildDim(hCtx,ErrorName,false);

   sts = EssEndIncrementalBuildDim(hCtx,ESS_DOR_ALLDATA,"tmpotl",ErrorName,false);
   return sts;

See Also

- EssIncrementalBuildDim
- EssBeginIncrementalBuildDim
- EssBeginStreamBuildDim
- EssEndIncrementalBuildDim
- EssSendString

EssEndUpdate

Marks the end of an update specification being sent to the active database. This function must
be called after sending the update specification (using EssSendString()).

Syntax

ESS_FUNC_M EssEndUpdate (hCtx);

Parameter   Data Type      Description
hCtx      ESS_HCTX_T      API context handle.

Notes

This function must be preceded by a call to EssBeginUpdate(), and at least one call to
EssSendString().

Return Value

None.
Access
This function requires the caller to have write privilege (ESS_PRIV_WRITE) to the active database.

Example
See the example of EssBeginUpdate.

See Also
- EssBeginUpdate
- EssSendString
- EssUpdate

EssExport
Exports a database to an ASCII file.

Syntax

```c
ESS_FUNC_M EssExport (hCtx, AppName, DbName, PathName, Level, Columns);
```

Parameter | Data Type   | Description
----------|-------------|-------------
hCtx       | ESS_HCTX_T  | API context handle.
AppName    | ESS_STR_T   | Name of application to export.
DbName     | ESS_STR_T   | Name of database to export.
PathName   | ESS_STR_T   | Full path name of server file to contain exported information.
Level      | ESS_SHORT_T | Controls level of data to export. Should be one of:
            |             | - ESS_DATA_ALL—Export all levels of data
            |             | - ESS_DATA_LEVEL0—Export all data only from level zero blocks
            |             | - ESS_DATA_INPUT—Export data only from input level blocks
Columns    | ESS_SHORT_T | Controls output of data blocks in column format (for creating rules files). Use non-zero for column format, and zero for no column format.

Notes
If the data for a thread exceeds 2 GB, Essbase may divide the export data into multiple files with numbers appended to the file names.

The naming convention for additional export files is as follows: _1, _2, etc. are appended to the additional file names. If the specified output file name contains a period, the numbers are appended before the period. Otherwise, they are appended at the end of the file name.

For example, if the given file name is `/home/exportfile.txt`, the next additional file is `/home/exportfile_1.txt`. If the file name is `/home/exportfile`, the next additional file is `/home/exportfile_1.`
Return Value

None.

Access

This function requires the caller to have at least read access (ESS_PRIV_READ) to the database, and to have selected it as their active database using EssSetActive().

Example

```c
ESS_FUNC_M
ESS_Export (ESS_HCTX_T   hCtx)
{
    ESS_FUNC_M       sts = ESS_STS_NOERR;
    ESS_SHORT_T     isLevel;
    ESS_STR_T       AppName;
    ESS_STR_T       DbName;
    ESS_STR_T       FileName;
    ESS_PROCSTATE_T pState;

    isLevel  = ESS_DATA_LEVEL0;
    AppName  = "Sample";
    DbName   = "Basic";

    sts = EssExport (hCtx, appName, dbName,
                     "D:\temp\asofile.txt", ESS_DATA_LEVEL0, ESS_FALSE);
    if (!sts)
    {
        sts = EssGetProcessState (hCtx, &pState);
        while (!sts && (pState.State !=
                      ESS_STATE_DONE))
            sts = EssGetProcessState (hCtx, &pState);
    }
    return (sts);
}
```

See Also

- EssImport

EssFixIBH

Repairs invalid header block corruption in the database. Currently, it removes all the invalid blocks from the database.

Syntax

```c
ESS_FUNC_M EssFixIBH (hCtx, action);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx;</td>
<td>ESS_HCTX_T</td>
<td>Context handle</td>
</tr>
<tr>
<td>action;</td>
<td>ESS_IBH_ACTION</td>
<td>An enumeration type. For this release the only valid value is REMOVE.</td>
</tr>
</tbody>
</table>
**See Also**
- EssLocateIBH
- EssGetIBH

**EssFree**

Frees a previously allocated block of memory, using the defined memory allocation scheme.

**Syntax**

```
ESS_FUNC_M EssFree (hInstance, pBlock);
```

**Parameter**  **Data Type**  **Description**

<table>
<thead>
<tr>
<th>hInstance</th>
<th>ESS_HINST_T</th>
<th>API instance handle.</th>
</tr>
</thead>
<tbody>
<tr>
<td>pBlock</td>
<td>ESS_PVOID_T</td>
<td>Pointer to allocated memory block.</td>
</tr>
</tbody>
</table>

**Notes**

- This function frees memory using the user-supplied memory management function passed to the EssInit() function. If no such function is supplied, the default memory freeing function (dependent on the platform) will be used.
- This function should be used to free any memory allocated using the EssAlloc() and EssRealloc() functions. It should also be used to free any allocated buffers returned from Essbase API functions.

**Return Value**

None.

**Access**

This function requires no special privileges.

**Example**

```
ESS_FUNC_M
ESS_GetAppActive (ESS_HCTX_T       hCtx,
                  ESS_HINST_T      hInst
                  )
{
    ESS_FUNC_M         sts = ESS_STS_NOERR;
    ESS_STR_T          pDbName;
    ESS_STR_T          pAppName;
    ESS_ACCESS_T       Access;
    if ((sts = EssAlloc (hInst, 80, (ESS_PVOID_T)&pAppName)) == 0)
    {
        if ((sts = EssAlloc (hInst, 80, (ESS_PVOID_T)&pDbName)) == 0)
        {
            if ((sts = EssGetActive (hCtx, &pAppName, &pDbName, &Access)) == 0)
            {
                if (pAppName)
                {
```
if (*pAppName)
    printf ("Current active application is [%s]\n", pAppName);
else
    printf ("No active Application is set\n");
    printf ("\r\n");
}

EssFree (hInst, pDbName);
}

EssFree (hInst, pAppName);
}
return (sts);
}

See Also
- EssAlloc
- EssInit
- EssOtlGetMemberCommentEx
- EssOtlSetMemberCommentEx
- EssRealloc

**EssFreeMbrErr**

Frees the memory allocated for a linked list of ESS_MBRERR_T structures.

**Syntax**

```c
ESS_FUNC_M EssFreeMbrErr (hCtx, ppMbrErr);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>ppMbrErr</td>
<td>ESS_PMBRERR_T</td>
<td>Pointer to linked list contained in ESS_MBRERR_T.</td>
</tr>
</tbody>
</table>

**Notes**

- This function can only be used to free the memory allocated for ESS_MBRERR_T as used by EssImport().
- See EssImport() for more information on ppMbrErr.

**Return Value**

None.

**Access**

This function requires no special privileges.

**Example**

See the example of EssImport.
EssFreeStructure

Frees memory dynamically allocated by EssGetAttributeInfo() and EssGetMemberInfo() for string type attribute information.

Syntax

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInst;</td>
<td>ESS_HINST_T</td>
<td>The instance handle of the process that called EssGetAttributeInfo() or EssGetMemberInfo() to allocate the structure.</td>
</tr>
<tr>
<td>structId;</td>
<td>ESS_ULONG_T</td>
<td>One of the following constant identifiers for the structure:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_DT_STRUCT_ATTRIBUTEINFO</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_DT_STRUCT_ATTRSPECS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_DT_STRUCT_MEMBERINFO</td>
</tr>
<tr>
<td>count;</td>
<td>ESS_ULONG_T</td>
<td>Number of structures</td>
</tr>
<tr>
<td>structPtr;</td>
<td>ESS_PVOID_T</td>
<td>Pointer to memory</td>
</tr>
</tbody>
</table>

Notes

- Always call this function to free structures allocated with either EssGetAttributeInfo() or EssGetMemberInfo() before you leave the local routine.

Access

This function requires no special privileges.

Example

```c
void  ESS_GetAttributeSpecifications()
{
    ESS_STS_T         sts = ESS_STS_NOERR;
    ESS_PATTRSPECS_T  pAttrSpecs;

    sts = EssGetAttributeSpecifications(hCtx, &pAttrSpecs);

    printf("\n --------Attribute Specifications--------\n\n");
    if (sts) return(sts);

    switch(pAttrSpecs->usGenNameBy)
    {
        case ESS_GENNAMEBY_PREFIX:
            printf("\n Prefix/Suffix : Prefix");
            break;
        case ESS_GENNAMEBY_SUFFIX:
            printf("\n Prefix/Suffix : Suffix");
            break;
    }
}
```
break;
default:
    printf("\n Prefix/Suffix : None");
break;
}
switch(pAttrSpecs->usUseNameOf)
{
    case ESS_USENAMEOF_PARENT:
        printf("\n Use Name of : Parent");
        break;
    case ESS_USENAMEOF_GRANDPARENTANDPARENT:
        printf("\n Use Name of : Grand Parent and Parent");
        break;
    case ESS_USENAMEOF_ALLANCESTORS:
        printf("\n Use Name of : All Ancestors");
        break;
    case ESS_USENAMEOF_DIMENSION:
        printf("\n Use Name of : Dimension");
        break;
    case ESS_USENAMEOF_NONE:
        printf("\n Use Name of : None");
        break;
    default:
        printf("\n Use Name of : Invalid setting");
        break;
}
switch(pAttrSpecs->cDelimiter)
{
    case ESS_DELIMITER_PIPE:
        printf("\n Delimiter : |");
        break;
    case ESS_DELIMITER_UNDERSCORE:
        printf("\n Delimiter : _");
        break;
    case ESS_DELIMITER_CARET:
        printf("\n Delimiter : ^");
        break;
    default:
        printf("\n Delimiter : Invalid setting");
        break;
}
switch(pAttrSpecs->usDateFormat)
{
    case ESS_DATEFORMAT_DDMMYYYY:
        printf("\n Date Format : DD-MM-YYYY");
        break;
    case ESS_DATEFORMAT_MMDDYYYY:
        printf("\n Date Format : MM-DD-YYYY");
        break;
    default:
        printf("\n Date Format : Invalid setting");
        break;
}
switch(pAttrSpecs->usBucketingType)
{
    case ESS_UPPERBOUNDINCLUSIVE:
printf("\n Bucketing Type  : Upper Bound inclusive");
break;
case ESS_UPPERBOUNDNONINCLUSIVE :
    printf("\n Bucketing Type  : Upper Bound non-inclusive");
break;
case ESS_LOWERBOUNDINCLUSIVE :
    printf("\n Bucketing Type  : Lower Bound inclusive");
break;
case ESS_LOWERBOUNDNONINCLUSIVE :
    printf("\n Bucketing Type  : Lower Bound non-inclusive");
break;
default:
    printf("\n Bucketing Type  : Invalid setting");
break;
}

printf("\n Default for TRUE       : %s", pAttrSpecs->pszDefaultTrueString);
printf("\n Default for FALSE      : %s", pAttrSpecs->pszDefaultFalseString);
printf("\n Default for Attr Calc  : %s", pAttrSpecs->pszDefaultAttrCalcDimName);
printf("\n Default for Sum        : %s", pAttrSpecs->pszDefaultSumMbrName);
printf("\n Default for Count      : %s", pAttrSpecs->pszDefaultCountMbrName);
printf("\n Default for Average    : %s", pAttrSpecs->pszDefaultAverageMbrName);
printf("\n Default for Min        : %s", pAttrSpecs->pszDefaultMinMbrName);
printf("\n Default for Max        : %s", pAttrSpecs->pszDefaultMaxMbrName);
printf("\n");

EssFreeStructure(hInst, ESS_DT_STRUCT_ATTRSPECS, 1,(ESS_PVOID_T)pAttrSpecs);

See Also

- EssCheckAttributes
- EssGetAssociatedAttributesInfo
- EssGetAttributeInfo
- EssGetAttributeSpecifications
- EssOtlAssociateAttributeDimension
- EssOtlAssociateAttributeMember
- EssOtlDisassociateAttributeDimension
- EssOtlDisassociateAttributeMember
- EssOtlFindAttributeMembers
- EssOtlFreeStructure
- EssOtlGetAssociatedAttributes
- EssOtlGetAttributeInfo
- EssOtlGetAttributeSpecifications
- EssOtlQueryAttributes
- EssOtlSetAttributeSpecifications

EssGetActive

Gets the names of the caller’s current active application and database.

Syntax

```plaintext
ESS_FUNC_M EssGetActive (hCtx, pAppName, pDbName, pAccess);
```
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>pAppName</td>
<td>ESS_PSTR_T</td>
<td>Address of pointer to receive allocated application name string.</td>
</tr>
<tr>
<td>pDbName</td>
<td>ESS_PSTR_T</td>
<td>Address of pointer to receive allocated database name string.</td>
</tr>
<tr>
<td>pAccess;</td>
<td>ESS_PACCESS_T</td>
<td>Address of variable to receive the user’s access level to the selected database. See “Bitmask Data Types (C)” on page 96 for a list of possible values for this field.</td>
</tr>
</tbody>
</table>

**Notes**

You should free the memory allocated for `pAppName` and `pDbName` using `EssFree()`.

**Return Value**

If successful, returns the user’s selected active application and database in `pAppName` and `pDbName`.

**Access**

This function requires no special privileges.

**Example**

```c
ESS_FUNC_M
ESS_GetAppActive (ESS_HCTX_T hCtx, ESS_HINST_T hInst)
{
    ESS_FUNC_M sts = ESS_STS_NOERR;
    ESS_STR_T pDbName;
    ESS_STR_T pAppName;
    ESS_ACCESS_T Access;
    if ((sts = EssAlloc (hInst, 80, (ESS_PPVOID_T)&pAppName)) == 0)
    {
        if ((sts = EssAlloc (hInst, 80, (ESS_PPVOID_T)&pDbName)) == 0)
        {
            if ((sts = EssGetActive (hCtx, &pAppName, &pDbName, &Access)) == 0)
            {
                if (pAppName)
                {
                    if (*pAppName)
                        printf ("Current active application is [%s]\n", pAppName);
                    else
                        printf ("No active Application is set\n");
                    printf ("\r\n");
                }
                EssFree (hInst, pDbName);
            }
            EssFree (hInst, pAppName);
        }
    }
}
```

340 C Main API Functions
EssGetAlias

Gets the active alias table name from the active database for a user.

Syntax
ESS_FUNC_M EssGetAlias (hCtx, pAliasName);

Parameter  Data Type  Description
hCtx       ESS_HCTX_T  API context handle.
pAliasName ESS_PSTR_T  Address of pointer to receive allocated name of active alias table.

Notes
The memory allocated for pAliasName should be freed using EssFree().

Return Value
If successful, returns the name of the active alias table in pAliasName.

Access
This function requires the caller to have at least read access (ESS_PRIV_READ) to the database, and to have selected it as their active database using EssSetActive().

Example
ESS_FUNC_M
ESS_GetAlias (ESS_HCTX_T hCtx, ESS_HINST_T hInst)
{
    ESS_FUNC_M     sts = ESS_STS_NOERR;
    ESS_STR_T      AliasName;

    sts = EssGetAlias(hCtx, &AliasName);

    if(!sts && AliasName)
    {
        printf("AliasName: %s\r\n",AliasName);
        EssFree(hInst,AliasName);
    }

    return (sts);
}

See Also
●   EssListAliases
EssGetAPIVersion

Returns the version of the Essbase API used to compile the current application.

Syntax

```c
ESS_FUNC_M EssGetAPIVersion (Version);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| Version   | ESS_PULONG_T | Version number of API. Hex value, in C notation, with the following format:
|           |           | 0x00000000 |
|           |           | ● First 4 numbers from right (low order word): release number between versions |
|           |           | ● Remaining numbers (high order word): version number |
|           |           | For example, 0x0004.0000 represents Release 4.0, and 0x0003.0002 represents Release 3.2. |

Notes

You can use this function to check the API version when your program requires a particular version.

Example

```c
ESS_VOID_T ESS_GetAPIVersion()
{
    ESS_FUNC_M    sts = ESS_STS_NOERR;
    ESS_ULONG_T  Version;
    
    sts = EssGetAPIVersion(&Version);
    
    if(!sts)
        printf("API Version %x\n",Version);
}
```

See Also

- EssGetObjectInfo

EssGetApplicationAccess

Gets a list of user application access structures, which contain information about user access to applications.

Syntax

```c
ESS_FUNC_M EssGetApplicationAccess (hCtx, UserName, AppName, pCount, ppUserApp);
```
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>UserName</td>
<td>ESS_STR_T</td>
<td>User name. If NULL, lists all users for the specified application.</td>
</tr>
<tr>
<td>AppName</td>
<td>ESS_STR_T</td>
<td>Application name. If NULL, lists all applications for the specified user.</td>
</tr>
<tr>
<td>pCount</td>
<td>ESS_PUSHORT_T</td>
<td>Address of variable to receive count of user application structures.</td>
</tr>
<tr>
<td>ppUserApp</td>
<td>“ESS_USERAPP_T, ESS_GROUPAPP_T” on page 189</td>
<td>Address of pointer to receive an allocated array of user application structures.</td>
</tr>
</tbody>
</table>

**Notes**

- If `UserName` is NULL, all users will be listed for the specified application. If `AppName` is NULL, all applications will be listed for the specified user. However, `UserName` and `AppName` cannot both be NULL.
- The `Access` field of the user application structure is used to represent the user’s granted access to the application, whereas the `MaxAccess` field represents the user’s highest access from all sources (e.g. via groups or default application access etc.).
- The memory allocated for `ppUserApp` should be freed using `EssFree()`.

**Return Value**

If successful, returns a count of users/applications in `pCount`, and a list of user application structures in `ppUserApp`.

**Access**

This function requires callers to have application designer privilege (ESS_PRIV_APPDESIGN) for the specified application, unless they are getting their own application access information.

**Example**

```c
ESS_FUNC_M
ESS_GetApplicationAccess (ESS_HCTX_T hCtx, ESS_HINST_T hInst) {
    ESS_FUNC_M       sts = ESS_STS_NOERR;
    ESS_STR_T       UserName;
    ESS_STR_T       AppName;
    ESS_USHORT_T    Count = 0;
    ESS_USHORT_T    ind;
    ESS_PUSERAPP_T  UserApp = NULL;

    UserName = "Admin";
    AppName  = "";

    sts = EssGetApplicationAccess(hCtx, UserName, AppName, &Count, &UserApp);
    if(!sts)
    {
        if(Count && UserApp)
        {
            printf ("\n-------Application Access List----\n\n");
        }
    }
}
```
for (ind = 0; ind < Count; ind++)
{
    printf("User->%s Application->%-10s
Access->%-4d MaxAccess->%-6d\n", 
    UserApp[ind].UserName, 
    UserApp[ind].AppName, 
    UserApp[ind].Access, 
    UserApp[ind].MaxAccess);
}
 mensenFree (hInst, UserApp);
}
else
    printf ("\rUser Application list is empty\n\n");
return (sts);
}
See Also
● EssGetApplicationAccessEx
● EssGetDatabaseAccess
● EssListUsers
● EssSetApplicationAccess
● EssSetUser

EssGetApplicationAccessEx

Gets a list of user or group application access structures, which contain information about user
or group access to applications. Similar to EssGetApplicationAccess, but can accept a user
directory specification or unique identity attribute for UserID.

Syntax

ESS_FUNC_M EssGetApplicationAccessEx (hCtx, UserId, bIsIdentity, type, AppName, pCount, ppUserApp);

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle (input).</td>
</tr>
<tr>
<td>UserId</td>
<td>ESS_STR_T</td>
<td>User or group name (input). Can be specified as name@provider or as a</td>
</tr>
<tr>
<td></td>
<td></td>
<td>unique identity attribute. If NULL, lists all users or groups for the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>specified application.</td>
</tr>
<tr>
<td>bIsIdentity</td>
<td>ESS_BOOL_T</td>
<td>Input. Indicates if UserID is a name or an identity. If TRUE, UserID is an</td>
</tr>
<tr>
<td></td>
<td></td>
<td>identity.</td>
</tr>
<tr>
<td>type</td>
<td>ESS_U Short_T</td>
<td>Type of entity (input). Indicates if UserID is a group or a user. Can be</td>
</tr>
<tr>
<td></td>
<td></td>
<td>one of the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESS_TYPE_USER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESS_TYPE_GROUP</td>
</tr>
<tr>
<td>AppName</td>
<td>ESS_STR_T</td>
<td>Application name (input). If NULL, lists all applications for the specified</td>
</tr>
<tr>
<td>pCount</td>
<td>ESS_PUSHORT_T</td>
<td>Address of variable to receive count of user application structures (output).</td>
</tr>
<tr>
<td>Parameter</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>----------------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ppUserApp</td>
<td>ESS_PPUSERAPPEX_T</td>
<td>Address of pointer to receive an allocated array of user application structures (output). The user application structure can include user directories and unique identity attributes.</td>
</tr>
</tbody>
</table>

**Notes**

- If *UserID* is NULL, all users will be listed for the specified application. If *AppName* is NULL, all applications will be listed for the specified user. However, *UserID* and *AppName* cannot both be NULL.
- The *Access* field of the user application structure is used to represent the user’s granted access to the application, whereas the *MaxAccess* field represents the user’s highest access from all sources (e.g., via groups or default application access etc.).
- The memory allocated for *ppUserApp* should be freed using **EssFree()**.

**Return Value**

If successful, returns a count of users/applications in `pCount`, and a list of user application structures in `ppUserApp`.

**Access**

This function requires callers to have application designer privilege (ESS_PRIV_APPDESIGN) for the specified application, unless they are getting their own application access information.

**Example**

```c
void DisplayUserAppInfo(ESS_PPUSERAPPEX_T userApp, ESS_USHORT_T count)
{
    ESS_STS_T sts = ESS_STS_NOERR;
    ESS_USHORT_T ind;

    printf ("\n------Application Access List------\n\n");
    for (ind = 0; ind < count; ind++)
    {
        printf("\tUser: %s\n", userApp[ind].UserName);
        printf("\tProvider Name: %s\n", userApp[ind].ProviderName);
        printf("\tConnection Param: %s\n", userApp[ind].connparam);
        printf("\tAppName: %s\n", userApp[ind].AppName);
        switch(userApp[ind].Access)
        {
            case ESS_PRIV_NONE:
                printf("\tAccess: %d - ESS_PRIV_NONE\n", userApp[ind].Access);
                break;
            case ESS_PRIV_READ:
                printf("\tAccess: %d - ESS_PRIV_READ\n", userApp[ind].Access);
                break;
            case ESS_PRIV_WRITE:
                printf("\tAccess: %d - ESS_PRIV_WRITE\n", userApp[ind].Access);
                break;
            case ESS_PRIV_CALC:
                printf("\tAccess: %d - ESS_PRIV_CALC\n", userApp[ind].Access);
        }
    }
}
```
break;
case ESS_PRIV_METAREAD:
    printf("\tAccess: %d - ESS_PRIV_METAREAD\n", userApp[ind].Access);
    break;
case ESS_PRIV_DBCREATE:
    printf("\tAccess: %d - ESS_PRIV_DBCREATE\n", userApp[ind].Access);
    break;
case ESS_PRIV_APPMANAGE:
    printf("\tAccess: %d - ESS_PRIV_APPMANAGE\n", userApp[ind].Access);
    break;
case ESS_PRIV_APPLOAD:
    printf("\tAccess: %d - ESS_PRIV_APPLOAD\n", userApp[ind].Access);
    break;
case ESS_PRIV_APPCREATE:
    printf("\tAccess: %d - ESS_PRIV_APPCREATE\n", userApp[ind].Access);
    break;
case ESS_PRIV_USERCREATE:
    printf("\tAccess: %d - ESS_PRIV_USERCREATE\n", userApp[ind].Access);
    break;
    case ESS_ACCESS_READ:
    printf("\tAccess: %d - ESS_ACCESS_READ\n", userApp[ind].Access);
    break;
case ESS_ACCESS_WRITE:
    printf("\tAccess: %d - ESS_ACCESS_WRITE\n", userApp[ind].Access);
    break;
case ESS_ACCESS_CALC:
    printf("\tAccess: %d - ESS_ACCESS_CALC\n", userApp[ind].Access);
    break;
case ESS_ACCESS_METAREAD:
    printf("\tAccess: %d - ESS_ACCESS_METAREAD\n", userApp[ind].Access);
    break;
case ESS_ACCESS_DBMANAGE:
    printf("\tAccess: %d - ESS_ACCESS_DBMANAGE\n", userApp[ind].Access);
    break;
case ESS_ACCESS_DBCREATE:
    printf("\tAccess: %d - ESS_ACCESS_DBCREATE\n", userApp[ind].Access);
    break;
case ESS_ACCESS_APPMANAGE:
    printf("\tAccess: %d - ESS_ACCESS_APPMANAGE\n", userApp[ind].Access);
    break;
case ESS_ACCESS_APPCREATE:
    printf("\tAccess: %d - ESS_ACCESS_APPCREATE\n", userApp[ind].Access);
    break;
case ESS_ACCESS_APPCREATE:
    printf("\tAccess: %d - ESS_ACCESS_APPCREATE\n", userApp[ind].Access);
    break;
case ESS_ACCESS_DBALL:
    printf("\tAccess: %d - ESS_ACCESS_DBALL\n", userApp[ind].Access);
    break;
case ESS_ACCESS_APPALL:
printf("\tAccess: %d - ESS_ACCESS_APPALL\n", userApp[ind].Access);
break;
case ESS_ACCESS_ADMIN:
    printf("\tAccess: %d - ESS_ACCESS_ADMIN\n", userApp[ind].Access);
    break;
default:
    printf("\tAccess: Unknown\n");
}

switch(userApp[ind].MaxAccess)
{
    case ESS_PRIV_NONE:
        printf("\tMax Access: %d - ESS_PRIV_NONE\n", userApp[ind].MaxAccess);
        break;
    case ESS_PRIV_READ:
        printf("\tMax Access: %d - ESS_PRIV_READ\n", userApp[ind].MaxAccess);
        break;
    case ESS_PRIV_WRITE:
        printf("\tMax Access: %d - ESS_PRIV_WRITE\n", userApp[ind].MaxAccess);
        break;
    case ESS_PRIV_CALC:
        printf("\tMax Access: %d - ESS_PRIV_CALC\n", userApp[ind].MaxAccess);
        break;
    case ESS_PRIV_METAREAD:
        printf("\tMax Access: %d - ESS_PRIV_METAREAD\n", userApp[ind].MaxAccess);
        break;
    case ESS_PRIV_DBLOAD:
        printf("\tMax Access: %d - ESS_PRIV_DBLOAD\n", userApp[ind].MaxAccess);
        break;
    case ESS_PRIV_DBMANAGE:
        printf("\tMax Access: %d - ESS_PRIV_DBMANAGE\n", userApp[ind].MaxAccess);
        break;
    case ESS_PRIV_DBCREATE:
        printf("\tMax Access: %d - ESS_PRIV_DBCREATE\n", userApp[ind].MaxAccess);
        break;
    case ESS_PRIV_APPLOAD:
        printf("\tMax Access: %d - ESS_PRIV_APPLOAD\n", userApp[ind].MaxAccess);
        break;
    case ESS_PRIV_APPMANAGE:
        printf("\tMax Access: %d - ESS_PRIV_APPMANAGE\n", userApp[ind].MaxAccess);
        break;
    case ESS_PRIV_APPCREATE:
        printf("\tMax Access: %d - ESS_PRIV_APPCREATE\n", userApp[ind].MaxAccess);
        break;
    case ESS_PRIV_USERCREATE:
        printf("\tMax Access: %d - ESS_PRIV_USERCREATE\n", userApp[ind].MaxAccess);
        break;
    case ESS_ACCESS_READ:
        printf("\tMax Access: %d - ESS_ACCESS_READ\n", userApp[ind].MaxAccess);
        break;
    case ESS_ACCESS_WRITE:
        printf("\tMax Access: %d - ESS_ACCESS_WRITE\n", userApp[ind].MaxAccess);
        break;
    case ESS_ACCESS_CALC:
        printf("\tMax Access: %d - ESS_ACCESS_CALC\n", userApp[ind].MaxAccess);
        break;
}
case ESS_ACCESS_METAREAD:
    printf("\tMax Access: %d - ESS_ACCESS_METAREAD\n", userApp[ind].MaxAccess);
    break;
    
case ESS_ACCESS_DBMANAGE:
    printf("\tMax Access: %d - ESS_ACCESS_DBMANAGE\n", userApp[ind].MaxAccess);
    break;
    
case ESS_ACCESS_DBCREATE:
    printf("\tMax Access: %d - ESS_ACCESS_DBCREATE\n", userApp[ind].MaxAccess);
    break;
    
case ESS_ACCESS_APPMANAGE:
    printf("\tMax Access: %d - ESS_ACCESS_APPMANAGE\n", userApp[ind].MaxAccess);
    break;
    
case ESS_ACCESS_APPCREATE:
    printf("\tMax Access: %d - ESS_ACCESS_APPCREATE\n", userApp[ind].MaxAccess);
    break;
    
case ESS_ACCESS_FILTER:
    printf("\tMax Access: %d - ESS_ACCESS_FILTER\n", userApp[ind].MaxAccess);
    break;
    
case ESS_ACCESS_DBALL:
    printf("\tMax Access: %d - ESS_ACCESS_DBALL\n", userApp[ind].MaxAccess);
    break;
    
case ESS_ACCESS_APPALL:
    printf("\tMax Access: %d - ESS_ACCESS_APPALL\n", userApp[ind].MaxAccess);
    break;
    
case ESS_ACCESS_ADMIN:
    printf("\tMax Access: %d - ESS_ACCESS_ADMIN\n", userApp[ind].MaxAccess);
    break;
    
default:
    printf("\tMax Access: Unknown\n");
    }
    
    printf("\n");
    }
    }

ESS_FUNC_M ESS_GetApplicationAccessEx (ESS_HCTX_T hCtx, ESS_HINST_T hInst)
{
    ESS_STS_T sts = ESS_STS_NOERR;
    ESS_STR_T userId;
    ESS_BOOL_T bIsIdentity;
    ESS_USHORT_T type;
    ESS_USHORT_T count = 0;
    ESS_USERAPPEX_T userApp[2];
    ESS_USERAPPEX_T pUserApp = ESS_NULL;

    count = 1;
    strcpy(userApp[0].UserName, "IDUser1");
    strcpy(userApp[0].ProviderName, ":");
    strcpy(userApp[0].connparam, ":");
    userApp[0].type = ESS_TYPE_USER;
    strcpy(userApp[0].AppName, AppName);
    userApp[0].Access = ESS_PRIV_APPMANAGE;
    userApp[0].MaxAccess = ESS_PRIV_APPMANAGE;
    sts = EssSetApplicationAccessEx(hCtx, count, &userApp);
    printf("EssSetApplicationAccessEx sts: %ld\n", sts);
userId = "IDUser1";
AppName = AppName;
type = ESS_TYPE_GROUP;
bIsIdentity = ESS_FALSE;
sts = EssGetApplicationAccessEx(hCtx, userId, bIsIdentity, type, AppName, &count, &pUserApp);
printf("EssGetApplicationAccessEx sts: %ld\n", sts);
if(!sts)
{
    if(count && pUserApp)
    {
        DisplayUserAppInfo(pUserApp, count);
        sts = EssFree (hInst, pUserApp);
    }
    else
        printf ("\rUser Application list is empty\n\n");
    return (sts);
}

See Also

- EssGetDatabaseAccessEx
- EssListUsersInfoEx
- EssSetApplicationAccessEx

**EssGetApplicationInfo**

Gets an application's information structure, which contains non user-configurable parameters for the application.

**Syntax**

ESS_FUNC_M EssGetApplicationInfo (hCtx, AppName, ppAppInfo);

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle (logged in)</td>
</tr>
<tr>
<td>AppName</td>
<td>ESS_STR_T</td>
<td>Application name</td>
</tr>
<tr>
<td>ppAppInfo</td>
<td>&quot;ESS_APPINFO_T&quot; on page 115</td>
<td>Address of pointer to receive allocated application info structure</td>
</tr>
</tbody>
</table>

**Notes**

- This function can only be called for applications on the server.
- The memory allocated for ppAppInfo should be freed using EssFree().

**Return Value**

If successful, this function returns a pointer to an allocated application info structure in ppAppInfo.
Access

This function requires the caller to have at least read access (ESS_PRIV_READ) to the specified application.

Example

```
ESS_FUNC_M
ESS_GetAppInfo (ESS_HCTX_T       hCtx,
                ESS_HINST_T      hInst

            )
{
    ESS_FUNC_M        sts = ESS_STS_NOERR;
    ESS_PAPPINFO_T   AppInfo;
    ESS_USHORT_T     ind;
    ESS_STR_T        AppName;
    AppName = "Sample";

    sts = EssGetApplicationInfo (hCtx, AppName, &AppInfo);
    if (!sts)
    {
        if (AppInfo)
        {
            printf ("-------Application Info-------\r\n\n");
            printf ("Name            : %s\r\n", AppInfo->Name);
            printf ("Server Name     : %s\r\n", AppInfo->Server);
            printf ("Status          : %d\r\n", AppInfo->Status);
            printf ("Users Connected : %d\r\n", AppInfo->nConnects);
            printf ("Number of DBs   : %d\r\n", AppInfo->nDbs);
            printf ("--List of Databases--\r\n\n");
            for (ind = 0; ind < AppInfo->nDbs; ind++)
                printf ("database(%d)    : %s\r\n", ind,
                        AppInfo->DbNames [ind]);

            EssFree (hInst, AppInfo);
        }
    }

    return (sts);
}
```

See Also

- EssGetApplicationInfoEx
- EssGetApplicationState
- EssGetDatabaseInfo

EssGetApplicationInfoEx

Retrieves information from one or more applications.

Syntax

```
ESS_FUNC_M EssGetApplicationInfoEx (hCtx, AppName, pusCount, ppAppInfoEx);
```
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle (logged in).</td>
</tr>
<tr>
<td>AppName</td>
<td>ESS_STR_T</td>
<td>Name of application for which to return information. If NULL, returns information for all applications.</td>
</tr>
<tr>
<td>pusCount</td>
<td>ESS_PUSHORT_T</td>
<td>Number of information structures returned.</td>
</tr>
<tr>
<td>ppAppInfoEx</td>
<td>&quot;ESS_APPINFOEX_T&quot; on page 116</td>
<td>Address of pointer to an array of allocated application info structures.</td>
</tr>
</tbody>
</table>

Notes
- This function can only be called for applications on the server.
- The memory allocated for `ppAppInfo` should be freed using `EssFree()`.

Return Value
If successful, this function returns an array of application information structures in `ppAppInfo`.

Access
This function requires the caller to have at least read access (ESS_PRIV_READ) to the specified application.

Example
```c
ESS_FUNC_M
ESS_GetApplicationInfoEx (ESS_HCTX_T hCtx, ESS_HINST_T hInst)
{
    ESS_FUNC_M           sts = ESS_STS_NOERR;
    ESS_USHORT_T         ind;
    ESS_STR_T            AppName;
    ESS_USHORT_T         Count;
    ESS_PAPPINFOEX_T     AppInfoEx = NULL;
   AppName = "";
   sts = EssGetApplicationInfoEx (hCtx, AppName,
                   &Count, &AppInfoEx);
   if(!sts)
    {
        if(AppInfoEx)
        {
            printf("-----Application Info Ex -----
            for (ind = 0; ind <Count; ind++)
            {
            printf("Name:%s\r\n",AppInfoEx[ind].Name);
            printf("Server Name:%s\r\n", AppInfoEx[ind].Server);
            printf("Status:%d\r\n",AppInfoEx[ind].Status);
            printf("Users Connected:%d\r\n", 
            AppInfoEx[ind].nConnects);
            printf("\r\n");
            EssFree(hInst, AppInfoEx);
        }
    }
}
```
See Also

- `EssGetApplicationInfo`
- `EssGetApplicationState`
- `EssGetDatabaseInfo`

**EssGetApplicationState**

Gets an application's state structure, which contains user-configurable parameters for the application.

**Syntax**

```c
ESS_FUNC_M EssGetApplicationState (hCtx,AppName, ppAppState);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>AppName</td>
<td>ESS_STR_T</td>
<td>Application name.</td>
</tr>
<tr>
<td>ppAppState</td>
<td>“ESS_APPSTATE_T” on page 117</td>
<td>Address of pointer to receive allocated application state structure.</td>
</tr>
</tbody>
</table>

**Notes**

- This function cannot be called for local applications; it can only be called for applications on the server.
- Memory allocated for `ppAppState` should be freed using `EssFree()`.

**Return Value**

If successful, this function returns a pointer to an allocated application state structure in `ppAppState`.

**Access**

This function requires the caller to have at least read access (ESS_PRIV_READ) to the specified application.

**Example**

```c
ESS_FUNC_M
ESS_GetAppState (ESS_HCTX_T hCtx,
                 ESS_HINST_T hInst
                 )
{
    ESS_FUNC_M            sts = ESS_STS_NOERR;
    ESS_PAPPSTATE_T     AppState;
    ESS_STR_T            AppName;
    AppName = "Sample";
    sts = EssGetApplicationState (hCtx, AppName,
```
&AppState);
if (!sts)
{
    if (AppState)
    {
        EssFree (hInst, AppState);
    }
}
return (sts);

See Also

- EssGetApplicationInfo
- EssGetDatabaseState
- EssSetApplicationState

## EssGetAssociatedAttributesInfo

Returns the attribute members associated with a given base member.

### Syntax

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx;</td>
<td>ESS_HCTX_T</td>
<td>Context handle</td>
</tr>
<tr>
<td>BaseMbrName;</td>
<td>ESS_STR_T</td>
<td>Base member name</td>
</tr>
<tr>
<td>AttrDimName;</td>
<td>ESS_STR_T</td>
<td>(Optional) attribute dimension name</td>
</tr>
<tr>
<td>pCount;</td>
<td>ESS_PULONG_T</td>
<td>Number of attribute members returned</td>
</tr>
<tr>
<td>ppAttrInfo;</td>
<td>“ESS_ATTRIBUTEINFO_T” on page 118</td>
<td>Attribute information</td>
</tr>
</tbody>
</table>

### Notes

- Call this function to retrieve more information for attribute members than you retrieve using EssQueryDatabaseMembers.
- Set AttrDimName to NULL to return all attribute members that are associated with the base member.
- Optionally, provide an attribute dimension name to retrieve information only about the member of that dimension which is associated with the base member.

### Access

This function requires no special privileges.

### Example

```c
//void ESS_GetAssociateAttributeInfo();
ESS_GetAssociatedAttributesInfo();
```
{  
    ESS_STS_T     sts;
    ESS_ULONG_T   pCount=0;
    ESS_PATTRIBUTEINFO_T pAttributeInfo;
    ESS_USHORT_T  index=0;
    ESS_CHAR_T    time_string[32];
    struct tm*    pTime;
    ESS_DATETIME_T   et;
    ESS_PATTRSPECS_T pAttrSpecs;
    ESS_USHORT_T    usDateFormat;
    ESS_MBRNAME_T   attributeName;
    ESS_MBRNAME_T   dimensionName;

    pAttributeInfo = NULL;
    strcpy(attributeName, "100-10");
    strcpy(dimensionName, "\0");

    sts = EssGetAssociatedAttributesInfo(hCtx, attributeName, dimensionName, &pCount,
              &pAttributeInfo);

    /* for handling time values */
    et = pAttributeInfo->Attribute.value.dtData;
    if (!sts)
    {  
        printf ("Associated Attr info for [%s]\n", attributeName);
        printf ("------------------------------------\n");
        for (index=0; index<pCount; index++)
        {
            printf ("MbrName      : %s\n", pAttributeInfo[index].MbrName);
            printf ("DimName      : %s\n", pAttributeInfo[index].DimName);

            switch(pAttributeInfo[index].Attribute.usDataType)
            {
                case ESS_ATTRMBRDT_BOOL:
                    printf ("Data Type    : Boolean \n");
                    if ( pAttributeInfo[index].Attribute.value.bData)
                        printf ("Data Value   : True \n");
                    else
                        printf ("Data Value   : False \n");
                    break;

                case ESS_ATTRMBRDT_DOUBLE:
                    printf ("Data Type    : Numeric(Double) \n");
                    printf ("Data Value   : %g\n",pAttributeInfo[index].Attribute.value.dblData);
                    break;

                case ESS_ATTRMBRDT_DATETIME:
                    printf ("Data Type    : Date \n");
                    sts = EssGetAttributeSpecifications(hCtx, &pAttrSpecs);
                    if (!sts)
                        usDateFormat = ESS_DATEFORMAT_MMDDYYYY;
                    else
                        usDateFormat = pAttrSpecs->usDateFormat;
                    pTime = gmtime((time_t*)&et);
                    switch(usDateFormat)
                    {  
                        
                    }  

                break;
            }
        }
    }  
}
{  
case ESS_DATEFORMAT_MMDDYYYY:  
    sprintf(time_string, "MM-DD-YYYY %02i-%02i-%04i",  
               pTime->tm_mon+1, pTime->tm_mday,pTime->tm_year+1900);  
    break;  
case ESS_DATEFORMAT_DDMMYYYY:  
    sprintf(time_string, "DD-MM-YYYY %02i-%02i-%04i",  
               pTime->tm_mday,pTime->tm_mon+1, pTime->tm_year+1900);  
    break;  
}  
printf ("Data Value : %s \n", time_string);  
break;  
case ESS_ATTRMBRDT_STRING:  
    printf ("Data Type : String \n");  
    printf ("Data Value : %s \n",  
               pAttributeInfo[index].Attribute.value.strData);  
    EssFree(hInst, pAttributeInfo[index].Attribute.value.strData);  
    break;  
}  
printf("\n");  
}  
if (pAttributeInfo)  
    EssFreeStructure(hInst, ESS_DT_STRUCT_ATTRIBUTEINFO, 1, pAttributeInfo);  
return (sts);  
}

See Also

- EssCheckAttributes
- EssFreeStructure
- EssGetAttributeInfo
- EssGetAttributeSpecifications
- EssOtlAssociateAttributeDimension
- EssOtlAssociateAttributeMember
- EssOtlDisassociateAttributeDimension
- EssOtlDisassociateAttributeMember
- EssOtlFindAttributeMembers
- EssOtlFreeStructure
- EssOtlGetAssociatedAttributes
- EssOtlGetAttributeInfo
- EssOtlGetAttributeSpecifications
- EssOtlQueryAttributes
- EssOtlSetAttributeSpecifications

**EssGetAsyncProcLog**

Gets the error log for an asynchronous data load or dimension build process.

Syntax

```c
ESS_FUNC_M EssGetAsyncProcLog (hCtx, ErrorFileName, ErFileOverWrite);
```
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>ErrorFileName</td>
<td>ESS_STR_T</td>
<td>An error file name.</td>
</tr>
<tr>
<td>ErFileOverWrite</td>
<td>ESS_BOOL_T</td>
<td>If TRUE, overwrite the error file.</td>
</tr>
</tbody>
</table>

**Notes**

Call this function after initiating an asynchronous process using EssAsyncImport or EssAsyncBuildDim.

**Return Value**

Returns zero if successful. Otherwise, returns an error code.

**Example**

See the example for **EssAsyncBuildDim**.

**See Also**

- **EssAsyncBuildDim**
- **EssAsyncImport**
- **EssAsyncImportASO**
- **EssGetAsyncProcState**
- **EssCancelAsyncProc**
- **EssCloseAsyncProc**

---

**EssGetAsyncProcState**

Queries the state of an asynchronous process an asynchronous data load or dimension build process.

**Syntax**

```c
ESS_FUNC_M EssGetAsyncProcState (hCtx, pBldDlState);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>pBldDlState</td>
<td>ESS_PBLDDL_STATE_T</td>
<td>Address of pointer to receive allocated process state structure.</td>
</tr>
</tbody>
</table>

**Notes**

Call this function after initiating an asynchronous process using EssAsyncImport or EssAsyncBuildDim.

**Return Value**

Returns zero if successful. Otherwise, returns an error code.
Example
See the example for EssAsyncBuildDim.

See Also
- EssAsyncBuildDim
- EssAsyncImport
- EssAsyncImportASO
- EssGetAsyncProcLog
- EssCancelAsyncProc
- EssCloseAsyncProc

**EssGetAttributeInfo**

Returns attribute information for a given attribute member or dimension.

**Syntax**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx;</td>
<td>ESS_HCTX_T</td>
<td>Context handle</td>
</tr>
<tr>
<td>szAttributeName;</td>
<td>ESS_STR_T</td>
<td>Name of the attribute member or dimension</td>
</tr>
<tr>
<td>pAttributeInfo;</td>
<td>“ESS_ATTRIBUTEINFO_T” on page 118</td>
<td>Attribute information</td>
</tr>
</tbody>
</table>

**Notes**

- After you call this function, call EssFreeStructure() to free memory dynamically allocated by EssGetAttributeInfo() for string type attribute information.

**Access**

This function requires no special privileges.

**Example**

```c
void ESS_GetAttributeInfo()
{
    ESS_STS_T             sts;
    ESS_PATTRIBUTEINFO_T  pAttributeInfo;
    ESS_CHAR_T            time_string[32];
    struct tm*            pTime;
    ESS_DATETIME_T        et;
    ESS_PATTRSPECS_T      pAttrSpecs;
    ESS_USHORT_T          usDateFormat;

    /* sts = EssGetAttributeInfo(hCtx, "ounces_12", &pAttributeInfo); */
    /* sts = EssGetAttributeInfo(hCtx, "ounces", &pAttributeInfo); */
    /* sts = EssGetAttributeInfo(hCtx, "caffeinated_true", &pAttributeInfo); */
    /* sts = EssGetAttributeInfo(hCtx, "caffeinated", &pAttributeInfo); */
    sts = EssGetAttributeInfo(hCtx, "intro date_10-01-1996", &pAttributeInfo);
```

C Main API Function Reference  357
if(sts)
    fprintf(stderr,"Error in EssGetAttributeInfo(): %ld", sts);

/* for handling time values */
et = pAttributeInfo->Attribute.value.dtData;
printf("Member name: %s\n", pAttributeInfo->MbrName);
printf("Dimension name: %s\n", pAttributeInfo->DimName);
/* printf("Attribute: %s\n", pAttributeInfo->Attribute); */
switch(pAttributeInfo->Attribute.usDataType)
{
    case ESS_ATTRMBRDT_BOOL:
        printf ("Data Type : Boolean \n");
        if ( pAttributeInfo->Attribute.value.bData)
            printf ("Data Value : True \n");
        else
            printf ("Data Value : False \n");
        break;

    case ESS_ATTRMBRDT_DOUBLE:
        printf ("Data Type : Numeric(Double) \n");
        printf ("Data Value : %g \n",pAttributeInfo->Attribute.value.dblData);
        break;

    case ESS_ATTRMBRDT_DATETIME:
        printf ("Data Type : Date \n");
        sts = EssGetAttributeSpecifications(hCtx, &pAttrSpecs);
        if (sts)
            usDateFormat = ESS_DATEFORMAT_MMDDYYYY;
        else
            usDateFormat = pAttrSpecs->usDateFormat;
        pTime =  gmtime((time_t*)&et);
        switch(usDateFormat)
        {
            case ESS_DATEFORMAT_MMDDYYYY:
                sprintf(time_string, "MM-DD-YYYY %02i-%02i-%04i",
                    pTime->tm_mon+1, pTime->tm_mday,pTime->tm_year+1900);
                break;
            case ESS_DATEFORMAT_DDMMYYYY :
                sprintf(time_string, "DD-MM-YYYY %02i-%02i-%04i",
                    pTime->tm_mday,pTime->tm_mon+1, pTime->tm_year+1900);
                break;
            }
        printf ("Data Value : %s \n", time_string);
        break;

    case ESS_ATTRMBRDT_STRING:
        printf ("Data Type : String \n");
        printf ("Data Value : %s \n", pAttributeInfo->Attribute.value.strData);
        EssFree(hInst, pAttributeInfo->Attribute.value.strData);
        break;

    }
    EssFreeStructure(hInst, ESS_DT_STRUCT_ATTRIBUTEINFO, 1, pAttributeInfo);
}
See Also

- EssCheckAttributes
- EssFreeStructure
- EssGetAssociatedAttributesInfo
- EssGetAttributeSpecifications
- EssOtlAssociateAttributeDimension
- EssOtlAssociateAttributeMember
- EssOtlDisassociateAttributeDimension
- EssOtlDisassociateAttributeMember
- EssOtlFindAttributeMembers
- EssOtlFreeStructure
- EssOtlGetAssociatedAttributes
- EssOtlGetAttributeInfo
- EssOtlGetAttributeSpecifications
- EssOtlQueryAttributes
- EssOtlSetAttributeSpecifications

EssGetAttributeSpecifications

Retrieves attribute specifications for the outline.

Syntax

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx;</td>
<td>ESS_HCTX_T</td>
<td>Context handle</td>
</tr>
<tr>
<td>pAttrSpecs;</td>
<td>“ESS_ATTRSPECS_T”</td>
<td>Attribute specifications</td>
</tr>
</tbody>
</table>

Notes

- Set attribute specifications for the outline using EssOtlSetAttributeSpecifications.
- Attribute specifications are used to do the following:
  - Generate a long name
  - Indicate the format of a datetime attribute
  - Indicate a numeric attribute's bucketing type
  - Provide the name of the attribute calculations dimension and the names for the values used with it

  See Table 6, “C API Attributes Terminology,” on page 102.

Access

This function requires no special privileges.
Example

```c
void ESS_GetAttributeSpecifications()
{
    ESS_STS_T     sts = ESS_STS_NOERR;
    ESS_PATTRSPECS_T pAttrSpecs;

    sts = EssGetAttributeSpecifications(hCtx, &pAttrSpecs);

    printf("\n ---------Attribute Specifications--------\n\n");
    if (sts) return(sts);

    switch(pAttrSpecs->usGenNameBy)
    {
        case ESS_GENNAMEBY_PREFIX:
            printf("\n Prefix/Suffix   : Prefix");
            break;
        case ESS_GENNAMEBY_SUFFIX:
            printf("\n Prefix/Suffix   : Suffix");
            break;
        default:
            printf("\n Prefix/Suffix   : None");
            break;
    }

    switch(pAttrSpecs->usUseNameOf)
    {
        case ESS_USENAMEOF_PARENT:
            printf("\n Use Name of     : Parent");
            break;
        case ESS_USENAMEOF_GRANDPARENTANDPARENT:
            printf("\n Use Name of     : Grand Parent and Parent");
            break;
        case ESS_USENAMEOF_ALLANCESTORS:
            printf("\n Use Name of     : All Ancestors");
            break;
        case ESS_USENAMEOF_DIMENSION:
            printf("\n Use Name of     : Dimension");
            break;
        case ESS_USENAMEOF_NONE:
            printf("\n Use Name of     : None");
            break;
        default:
            printf("\n Use Name of     : Invalid setting");
            break;
    }

    switch(pAttrSpecs->cDelimiter)
    {
        case ESS_DELIMITER_PIPE:
            printf("\n Delimiter       : '\''");
            break;
        case ESS_DELIMITER_UNDERSCORE:
            printf("\n Delimiter       : '_'");
            break;
        case ESS_DELIMITER_CARET:
            printf("\n Delimiter       : '^'");
            break;
        default:
            printf("\n Delimiter       : Invalid setting");
            break;
    }
}
```
switch(pAttrSpecs->usDateFormat)
{
    case ESS_DATEFORMAT_DDMMYYYY :
        printf("\n Date Format     : DD-MM-YYYY");
        break;
    case ESS_DATEFORMAT_MMDYYYY :
        printf("\n Date Format     : MM-DD-YYYY");
        break;
    default:
        printf("\n Date Format     : Invalid setting");
        break;
}

switch(pAttrSpecs->usBucketingType)
{
    case ESS_UPPERBOUNDINCLUSIVE :
        printf("\n Bucketing Type  : Upper Bound inclusive");
        break;
    case ESS_UPPERBOUNDNONINCLUSIVE :
        printf("\n Bucketing Type  : Upper Bound non-inclusive");
        break;
    case ESS_LOWERBOUNDINCLUSIVE :
        printf("\n Bucketing Type  : Lower Bound inclusive");
        break;
    case ESS_LOWERBOUNDNONINCLUSIVE :
        printf("\n Bucketing Type  : Lower Bound non-inclusive");
        break;
    default:
        printf("\n Bucketing Type   : Invalid setting");
        break;
}

printf("\n Default for TRUE       : %s",pAttrSpecs->pszDefaultTrueString);
printf("\n Default for FALSE      : %s",pAttrSpecs->pszDefaultFalseString);
printf("\n Default for Attr Calc  : %s",pAttrSpecs->pszDefaultAttrCalcDimName);
printf("\n Default for Sum        : %s",pAttrSpecs->pszDefaultSumMbrName);
printf("\n Default for Count      : %s",pAttrSpecs->pszDefaultCountMbrName);
printf("\n Default for Average    : %s",pAttrSpecs->pszDefaultAverageMbrName);
printf("\n Default for Min        : %s",pAttrSpecs->pszDefaultMinMbrName);
printf("\n Default for Max        : %s",pAttrSpecs->pszDefaultMaxMbrName);

EssFreeStructure(hInst, ESS_DT_STRUCT_ATTRSPECS, 1,(ESS_PVOID_T)pAttrSpecs);

See Also
- EssCheckAttributes
- EssFreeStructure
- EssGetAssociatedAttributesInfo
- EssGetAttributeInfo
- EssOtlAssociateAttributeDimension
- EssOtlAssociateAttributeMember
- EssOtlDisassociateAttributeDimension
EssGetCalcList

Gets the list of calc scripts objects which are accessible to a user.

Syntax

```c
ESS_FUNC_M EssGetCalcList (hCtx, UserName, AppName, DbName, pAllCalcs, pCount, ppCalcList);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle</td>
</tr>
<tr>
<td>UserName</td>
<td>ESS_STR_T</td>
<td>User name</td>
</tr>
<tr>
<td>AppName</td>
<td>ESS_STR_T</td>
<td>Application name</td>
</tr>
<tr>
<td>DbName</td>
<td>ESS_STR_T</td>
<td>Database name</td>
</tr>
<tr>
<td>pAllCalcs</td>
<td>ESS_PBOOL_T</td>
<td>Address of a variable to receive the allow all calcs flag. If TRUE, the user can access all calc scripts, otherwise, they can only access those specified in the CalcList argument.</td>
</tr>
<tr>
<td>pCount</td>
<td>ESS_PUSHORT_T</td>
<td>Address of variable to receive a count of the number of accessible calc script objects</td>
</tr>
<tr>
<td>ppCalcList</td>
<td>ESS_PPBJNAME_T</td>
<td>Address of a pointer to receive an allocated array of calc script object names</td>
</tr>
</tbody>
</table>

Notes

- In order to access any calc script objects, the specified user must have at least calculate access to the appropriate database.
- If the `pAllCalcs` flag is set to TRUE, `pCount` is zero, and `ppCalcList` is NULL.
- The memory allocated for `ppCalcList` should be freed using `EssFree()`.

Return Value

If successful, the user's allow all calcs setting is returned in `pAllCalcs`, a count of their accessible calc scripts objects is returned in `pCount`, and a list of calc script object names is returned in `ppCalcList`.

Access

This function requires the caller to have database Design privilege (ESS_PRIV_DBDESIGN) for the specified database, unless they are getting their own calc list.
Example

```c
ESS_FUNC_M
ESS_GetCalcList (ESS_HCTX_T hCtx, ESS_HINST_T hInst)
{
    ESS_FUNC_M       sts = ESS_STS_NOERR;
    ESS_STR_T       UserName;
    ESS_STR_T       AppName;
    ESS_STR_T       DbName;
    ESS_BOOL_T      AllCalcs;
    ESS_USHORT_T    Count, ind;
    ESS_POBJNAME_T  pCalcList = NULL;

    UserName = "Admin";
    AppName  = "Sample";
    DbName   = "Basic";
    sts = EssGetCalcList(hCtx, UserName, AppName,
                         DbName, &AllCalcs, &Count, &pCalcList);
    if(!sts && pCalcList)
    {
        printf("-------- Get Calc List -----------\r\n");
        for (ind = 0; ind < Count; ind++)
            printf(" %s\r\n",pCalcList[ind]);

        EssFree(hInst, pCalcList);
    }

    return (sts);
}
```

See Also

- EssListObjects
- EssListUsers
- EssSetCalcList

**EssGetCookie**

Gets the cookie associated with the current session, if a cookie was created at initialization. For more information, see the custom callback function, `CookieCreateFunc`, available with `ESS_INIT_T`.

**Syntax**

```c
ESS_FUNC_M EssGetCookie (ESS_HCTX_T, ESS_PPVOID_T);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>ppCookie</td>
<td>ESS_PPVOID_T</td>
<td>Address of a pointer to receive cookie.</td>
</tr>
</tbody>
</table>

**Return Value**

If successful, returns the cookie created at initialization.
Access
This function requires no special privileges.

Example
See the query cancellation example in the topic for ESS_INIT_T.

EssGetCellDrillThruReports

Gets the drill-through reports associated with a data cell as a list of URL XMLs, given the cell's member combination.

Syntax

```
ESS_FUNC_M EssGetCellDrillThruReports (hCtx, noMbrs, pMbrs, nURLXML, ppURLXMLLen, ppURLXML);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle</td>
</tr>
<tr>
<td>noMbrs</td>
<td>ESS_USHORT_T</td>
<td>Number of members in the member list pMbrs</td>
</tr>
<tr>
<td>pMbrs</td>
<td>ESS_PSTR_T</td>
<td>Pointer to the list of member names (or Aliases); the array size is assumed to be the dimension count</td>
</tr>
<tr>
<td>nURLXML</td>
<td>ESS_PUSHORT_T</td>
<td>Number of URL XMLs returned</td>
</tr>
<tr>
<td>ppURLXMLLen</td>
<td>ESS_PPUSHORT_T</td>
<td>Returns length of URL XML generated</td>
</tr>
<tr>
<td>ppURLXML</td>
<td>ESS_PPVOID_T</td>
<td>Returns pointers to the URL XML byte stream</td>
</tr>
</tbody>
</table>

Notes

The application database must be set to Active for this call. This function must be extended to support any additional information needed by the clients.

Return Value

- If successful, gets the list of URL XMLs.
- If unsuccessful, returns an error code.

Access

- Caller must have database Read privilege (ESS_PRIV_READ) for the specified database.
- Caller must have selected the specified database as the active database using EssSetActive().

Example

```
/* Sample Code for EssGetCellDrillThruReports */

ESS_STS_T sts = ESS_STS_NOERR;
ESS_SHORT_T numMbrs = 0;
ESS_STR_T *pMbrs = ESS_NULL;
ESS_USHORT_T numURLXML, i = 0;
```
ESS_USHORT_T *URLXMLLen = ESS_NULL;
ESS_PPVOID_T *URLXML = ESS_NULL;
ESS_CHAR_T pTmpXML[XML_CHAR_MAX];

/* Valid case */

numMbrs = 5;
sts = EssAlloc (hInst, sizeof(ESS_STR_T) * numMbrs , &pMbrs);
pMbrs[0] = "Jul";
pMbrs[1] = "100-10";
pMbrs[2] = "Actual";
sts = EssGetCellDrillThruReports(hCtx, numMbrs, pMbrs, &numURLXML, &URLXMLLen, &URLXML);
printf("EssGetCellDrillThruReports sts: %ld\n",sts);
if(!sts)
{
    printf("\nNumber of URL XML: %d", numURLXML);
    for (i = 0; i < numURLXML; i++)
    {
        memset(pTmpXML, 0, XML_CHAR_MAX);
        memcpy(pTmpXML, URLXML[i], URLXMLLen[i]);
        if ( URLXML[i] != ESS_NULL )
            printf("\tXML [\%d] : %s\n", i, pTmpXML );
        else
            printf("\tXML [\%d] : NULL STRING \n", i );
        if ( URLXML[i] != ESS_NULL )
            EssFree(hInst, URLXML[i]);
    }
    if ( URLXML != ESS_NULL )
        EssFree(hInst, URLXML);
    if ( URLXMLLen != ESS_NULL )
        EssFree(hInst, URLXMLLen);
}

EssGetCurrencyRateInfo

Gets a list of structures containing rate information for all members of the tagged currency
partition dimension in the active database outline.

Syntax

ESS_FUNC_M EssGetCurrencyRateInfo (hCtx, pCount, ppRateInfo);

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>pCount</td>
<td>ESS_PLONG_T</td>
<td>Address of variable to receive the count of rate info structures.</td>
</tr>
<tr>
<td>ppRateInfo</td>
<td>“ESS_RATEINFO_T” on page 179</td>
<td>Address of pointer to receive allocated array of currency rate info structures.</td>
</tr>
</tbody>
</table>
Notes

- The memory allocated for ppRateInfo should be freed using EssFree().
- This function can be called for regular databases with associated currency databases.

Return Value

If successful, this function returns a count of structures in pCount, and an allocated array of currency rate info structures in ppRateInfo.

Access

This function requires the caller to have at least read access (ESS_PRIV_READ) to the database, and to have selected it as their active database using EssSetActive().

Example

```c
ESS_FUNC_M
ESS_GetCrRate (ESS_HCTX_T hCtx,
    ESS_HINST_T hInst

    )
{
    ESS_FUNC_M        sts = ESS_STS_NOERR;
    ESS_LONG_T       count, i, j;
    ESS_PRATEINFO_T  pRateInfoList = NULL;
    ESS_CHAR_T       rateStr[(2 + ESS_MBRNAMELEN) * ESS_CRDB_MAXDIMNUM];
    sts = EssGetCurrencyRateInfo (hCtx, &count, &pRateInfoList);
    if (!sts)
    {
        if (count)
        {
            for (i = 0; i < count; i++)
            {
                rateStr[0] = '\0';
                for (j = 0; j < ESS_CRDB_MAXDIMNUM; j++)
                {
                    if (pRateInfoList[i].RateMbr[j][0])
                    {
                        if (rateStr[0])
                            strcat(rateStr, "->");
                        strcat(rateStr, pRateInfoList[i].RateMbr[j]);
                    }
                }
                if (!rateStr[0])
                    strcpy(rateStr, "(LOCAL)");
                if (i == 0)
                {
                    /* 1st is always DB rate */
                    printf("database [%s] : %s\n", pRateInfoList[i].MbrName, rateStr);
                }
                else
                {
                    printf("Partition [%s] : %s\n", pRateInfoList[i].MbrName, rateStr);
                }
            }
        }
    }
}
```
if (pRateInfoList)
    EssFree (hInst, pRateInfoList);
return (sts);
}

See Also
- EssListCurrencyDatabases
- EssSetActive

**EssGetDatabaseAccess**

Gets a list of user database access structures, which contain information about user access to databases.

**Syntax**

```c
ESS_FUNC_M EssGetDatabaseAccess (hCtx, UserName, AppName, DbName, pCount, ppUserDb);
```

**Parameter** | **Data Type** | **Description**
---|---|---
`hCtx` | ESS_HCTX_T | API context handle
`UserName` | ESS_STR_T | User name. If NULL, lists all users for the specified application and database.
`AppName` | ESS_STR_T | Application name. If NULL, lists all applications and databases for the specified user.
`DbName` | ESS_STR_T | Databasename. If NULL, lists all databases for the specified user or application.
`pCount` | ESS_PUSHORT_T | Address of variable to receive count of user database structures
`ppUserDb` | “ESS_USERDB_T, ESS_GROUPDB_T” on page 190 | Address of pointer to receive an allocated array of user database structures

**Notes**

- If any of `UserName`, `AppName`, or `DbName` are NULL, they will be treated as wild cards and all items of the appropriate type will be listed. If `AppName` is NULL, `DbName` is assumed to also be NULL. Any two of these arguments may be NULL, but not all three.
- The `Access` field of the user database structure is used to represent the user's granted access to the database, whereas the `MaxAccess` field represents the user's highest access from all sources (e.g. via groups or default database access etc.).
- The memory allocated for `ppUserDb` should be freed using `EssFree()`.
- Filter access privileges are equivalent to ESS_PRIV_DBLOAD privileges.

**Return Value**

If successful, returns a count of users/databases in `pCount`, and a list of user database structures in `ppUserDb`. 
Access

This function requires the caller to have database Design privilege (ESS_PRIV_DBDESIGN) for the specified database, unless they are getting their own database access information.

Example

ESS_FUNC_M
ESS_GetDatabaseAccess (ESS_HCTX_T hCtx, ESS_HINST_T hInst)
{
    ESS_FUNC_M stst = ESS_STS_NOERR;
    ESS_STR_T UserName;
    ESS_STR_T AppName;
    ESS_STR_T DbName;
    ESS_USHORT_T Count = 0;
    ESS_USHORT_T ind;
    ESS_PUSERDB_T UserDb = NULL;

    UserName = "Admin";
    AppName  = "Sample";
    DbName   = "";

    stst = EssGetDatabaseAccess(hCtx, UserName, AppName, DbName, &Count, &UserDb);
    if(!stst)
    {
        if(Count && UserDb)
        {
            printf ("-------Database Access List-------
               \r\n\r\n\r\n");
            for (ind = 0; ind < Count; ind++)
            {
                printf("User -> %s\r\n",UserDb[ind].UserName);
                printf("Application -> %s\r\n",  
                       UserDb[ind].AppName);
                printf("Database -> %s\r\n",UserDb[ind].DbName);
                printf("Access -> %d\r\n",UserDb[ind].Access);
                printf("MaxAccess -> %d\r\n",  
                       UserDb[ind].MaxAccess);
                printf("FilterName -> %s\r\n",  
                       UserDb[ind].FilterName);
                printf("=================================\n\r\n"));
            }
            EssFree (hInst, UserDb);
        }
        else
        printf ("Database list is empty\r\n\r\n");
    }
    return (stst);
}

See Also

- EssGetDatabaseAccessEx
- EssGetApplicationAccess
- EssGetUser
- EssListUsers
EssGetDatabaseAccessEx

Gets a list of user database access structures, which contain information about user access to databases. Similar to EssGetDatabaseAccess, but can accept a user directory specification or unique identity attribute for UserID.

Syntax

```c
ESS_FUNC_M EssGetDatabaseAccessEx (hCtx, UserId, bIsIdentity, type,AppName, DbName, pCount, ppUserDb);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle (input).</td>
</tr>
<tr>
<td>UserId</td>
<td>ESS_STR_T</td>
<td>User or group name (input). Can be specified as name@provider or as a unique identity attribute. If NULL, lists all users or groups for the specified database.</td>
</tr>
<tr>
<td>bIsIdentity</td>
<td>ESS_BOOL_T</td>
<td>Input. Indicates if UserID is a name or an identity. If TRUE, UserID is an identity.</td>
</tr>
<tr>
<td>type</td>
<td>ESS_USHORT_T</td>
<td>Type of entity (input). Indicates if UserID is a group or a user. Can be one of the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESS_TYPE_USER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESS_TYPE_GROUP</td>
</tr>
<tr>
<td>AppName</td>
<td>ESS_STR_T</td>
<td>Application name (input). If NULL, lists all applications and databases for the specified user.</td>
</tr>
<tr>
<td>DbName</td>
<td>ESS_STR_T</td>
<td>Database name (input). If NULL, lists all databases for the specified user or application.</td>
</tr>
<tr>
<td>pCount</td>
<td>ESS_PUSHORT_T</td>
<td>Address of variable to receive count of user database structures (output).</td>
</tr>
<tr>
<td>ppUserDb</td>
<td>ESS_PPUSERDBEX_T</td>
<td>Address of pointer to receive an allocated array of user database structures (output). The user database structure can include user directories and unique identity attributes.</td>
</tr>
</tbody>
</table>

Notes

- If any of UserID, AppName, or DbName are NULL, they will be treated as wild cards and all items of the appropriate type will be listed. If AppName is NULL, DbName is assumed to also be NULL. Any two of these arguments may be NULL, but not all three.
- The Access field of the user database structure is used to represent the user’s granted access to the database, whereas the MaxAccess field represents the user’s highest access from all sources (e.g. via groups or default database access etc.).
- The memory allocated for ppUserDb should be freed using EssFree().
- Filter access privileges are equivalent to ESS_PRIV_DBLOAD privileges.
Return Value

If successful, returns a count of users/databases in `pCount`, and a list of user database structures in `ppUserDb`.

Access

This function requires the caller to have database Design privilege (ESS_PRIV_DBDESIGN) for the specified database, unless they are getting their own database access information.

Example

```c
void DisplayUserDbInfo(ESS_PUSERDBEX_T userDb, ESS_USHORT_T count)
{
    ESS_STS_T sts = ESS_STS_NOERR;
    ESS_USHORT_T ind;

    printf ("\n-------Database Access List-------\n\n");
    for (ind = 0; ind < count; ind++)
    {
        printf("\tUser: %s\n", userDb[ind].UserName);
        printf("\tProvider Name: %s\n", userDb[ind].ProviderName);
        printf("\tConnection Param: %s\n", userDb[ind].connparam);
        printf("\tApp Name: %s\n", userDb[ind].AppName);
        printf("\tDb Name: %s\n", userDb[ind].DbName);
        switch(userDb[ind].Access)
        {
            case ESS_PRIV_NONE:
                printf("\tAccess: %d - ESS_PRIV_NONE\n", userDb[ind].Access);
                break;
            case ESS_PRIV_READ:
                printf("\tAccess: %d - ESS_PRIV_READ\n", userDb[ind].Access);
                break;
            case ESS_PRIV_WRITE:
                printf("\tAccess: %d - ESS_PRIV_WRITE\n", userDb[ind].Access);
                break;
            case ESS_PRIV_CALC:
                printf("\tAccess: %d - ESS_PRIV_CALC\n", userDb[ind].Access);
                break;
            case ESS_PRIV_METAREAD:
                printf("\tAccess: %d - ESS_PRIV_METAREAD\n", userDb[ind].Access);
                break;
            case ESS_PRIV_DBLOAD:
                printf("\tAccess: %d - ESS_PRIV_DBLOAD\n", userDb[ind].Access);
                break;
            case ESS_PRIV_DBMANAGE:
                printf("\tAccess: %d - ESS_PRIV_DBMANAGE\n", userDb[ind].Access);
                break;
            case ESS_PRIV_DBCREATE:
                printf("\tAccess: %d - ESS_PRIV_DBCREATE\n", userDb[ind].Access);
                break;
            case ESS_PRIV_APPLOAD:
                printf("\tAccess: %d - ESS_PRIV_APPLOAD\n", userDb[ind].Access);
                break;
            case ESS_PRIV_APPMANAGE:
                printf("\tAccess: %d - ESS_PRIV_APPMANAGE\n", userDb[ind].Access);
                break;
        }
    }
}
```
case ESS_PRIV_APPCREATE:
        printf("\tAccess: %d - ESS_PRIV_APPCREATE\n", userDb[ind].Access);
        break;
    case ESS_PRIV_USERCREATE:
        printf("\tAccess: %d - ESS_PRIV_USERCREATE\n", userDb[ind].Access);
        break;
    case ESS_ACCESS_READ:
        printf("\tAccess: %d - ESS_ACCESS_READ\n", userDb[ind].Access);
        break;
    case ESS_ACCESS_WRITE:
        printf("\tAccess: %d - ESS_ACCESS_WRITE\n", userDb[ind].Access);
        break;
    case ESS_ACCESS_CALC:
        printf("\tAccess: %d - ESS_ACCESS_CALC\n", userDb[ind].Access);
        break;
    case ESS_ACCESS_METAREAD:
        printf("\tAccess: %d - ESS_ACCESS_METAREAD\n", userDb[ind].Access);
        break;
    case ESS_ACCESS_DBMANAGE:
        printf("\tAccess: %d - ESS_ACCESS_DBMANAGE\n", userDb[ind].Access);
        break;
    case ESS_ACCESS_DBCREATE:
        printf("\tAccess: %d - ESS_ACCESS_DBCREATE\n", userDb[ind].Access);
        break;
    case ESS_ACCESS_APPMANAGE:
        printf("\tAccess: %d - ESS_ACCESS_APPMANAGE\n", userDb[ind].Access);
        break;
    case ESS_ACCESS_APPCREATE:
        printf("\tAccess: %d - ESS_ACCESS_APPCREATE\n", userDb[ind].Access);
        break;
    case ESS_ACCESS_FILTER:
        printf("\tAccess: %d - ESS_ACCESS_FILTER\n", userDb[ind].Access);
        break;
    case ESS_ACCESS_DBALL:
        printf("\tAccess: %d - ESS_ACCESS_DBALL\n", userDb[ind].Access);
        break;
    case ESS_ACCESS_APPALL:
        printf("\tAccess: %d - ESS_ACCESS_APPALL\n", userDb[ind].Access);
        break;
    case ESS_ACCESS_ADMIN:
        printf("\tAccess: %d - ESS_ACCESS_ADMIN\n", userDb[ind].Access);
        break;
    default:
        printf("\tAccess: Unknown\n");
}

switch(userDb[ind].MaxAccess)
{
    case ESS_PRIV_NONE:
        printf("\tMax Access: %d - ESS_PRIV_NONE\n", userDb[ind].MaxAccess);
        break;
    case ESS_PRIV_READ:
        printf("\tMax Access: %d - ESS_PRIV_READ\n", userDb[ind].MaxAccess);
        break;
    case ESS_PRIV_WRITE:
        printf("\tMax Access: %d - ESS_PRIV_WRITE\n", userDb[ind].MaxAccess);
        break;
printf("\tMax Access: %d - ESS_PRIV_WRITE\n", userDb[ind].MaxAccess); break;
case ESS_PRIV_CALC:
    printf("\tMax Access: %d - ESS_PRIV_CALC\n", userDb[ind].MaxAccess); break;
case ESS_PRIV_METAREAD:
    printf("\tMax Access: %d - ESS_PRIV_METAREAD\n", userDb[ind].MaxAccess); break;
case ESS_PRIV_DLOAD:
    printf("\tMax Access: %d - ESS_PRIV_DLOAD\n", userDb[ind].MaxAccess); break;
case ESS_PRIV_DBMANAGE:
    printf("\tMax Access: %d - ESS_PRIV_DBMANAGE\n", userDb[ind].MaxAccess); break;
case ESS_PRIV_DBCREATE:
    printf("\tMax Access: %d - ESS_PRIV_DBCREATE\n", userDb[ind].MaxAccess); break;
case ESS_PRIV_APPLOAD:
    printf("\tMax Access: %d - ESS_PRIV_APPLOAD\n", userDb[ind].MaxAccess); break;
case ESS_PRIV_APPMANAGE:
    printf("\tMax Access: %d - ESS_PRIV_APPMANAGE\n", userDb[ind].MaxAccess); break;
case ESS_PRIV_APPCREATE:
    printf("\tMax Access: %d - ESS_PRIV_APPCREATE\n", userDb[ind].MaxAccess); break;
case ESS_PRIV_USERCREATE:
    printf("\tMax Access: %d - ESS_PRIV_USERCREATE\n", userDb[ind].MaxAccess); break;
case ESS_ACCESS_READ:
    printf("\tMax Access: %d - ESS_ACCESS_READ\n", userDb[ind].MaxAccess); break;
case ESS_ACCESS_WRITE:
    printf("\tMax Access: %d - ESS_ACCESS_WRITE\n", userDb[ind].MaxAccess); break;
case ESS_ACCESS_CALC:
    printf("\tMax Access: %d - ESS_ACCESS_CALC\n", userDb[ind].MaxAccess); break;
case ESS_ACCESS_METAREAD:
    printf("\tMax Access: %d - ESS_ACCESS_METAREAD\n", userDb[ind].MaxAccess); break;
case ESS_ACCESS_DBMANAGE:
    printf("\tMax Access: %d - ESS_ACCESS_DBMANAGE\n", userDb[ind].MaxAccess); break;
case ESS_ACCESS_DBCREATE:
    printf("\tMax Access: %d - ESS_ACCESS_DBCREATE\n", userDb[ind].MaxAccess); break;
case ESS_ACCESS_APPMANAGE:
    printf("\tMax Access: %d - ESS_ACCESS_APPMANAGE\n", userDb[ind].MaxAccess); break;
case ESS_ACCESS_APPCREATE:
    printf("\tMax Access: %d - ESS_ACCESS_APPCREATE\n", userDb[ind].MaxAccess); break;
case ESS_ACCESS_FILTER:
    printf("\tMax Access: %d - ESS_ACCESS_FILTER\n", userDb[ind].MaxAccess); break;
case ESS_ACCESS_DBALL:
    printf("\tMax Access: %d - ESS_ACCESS_DBALL\n", userDb[ind].MaxAccess);
    break;

case ESS_ACCESS_APPALL:
    printf("\tMax Access: %d - ESS_ACCESS_APPALL\n", userDb[ind].MaxAccess);
    break;

case ESS_ACCESS_ADMIN:
    printf("\tMax Access: %d - ESS_ACCESS_ADMIN\n", userDb[ind].MaxAccess);
    break;

default:
    printf("\tMax Access: Unknown\n");

    printf("\tFilter Name: %s\n", userDb[ind].FilterName);
    printf("\n");

}

C Main API Function Reference 373

ESS_FUNC_M ESS_GetDatabaseAccessEx (ESS_HCTX_T hCtx, ESS_HINST_T hInst)
{
    ESS_STS_T sts = ESS_STS_NOERR;
    ESS_STR_T userId;
    ESS_BOOL_T bIsIdentity;
    ESS_USHORT_T type;
    ESS_USHORT_T count = 0;
    ESS_USERDBEX_T userDb[2];
    ESS_PUSERDBEX_T pUserDb = ESS_NULL;

count = 2;
    strcpy(userDb[0].UserName, "IDUser1");
    strcpy(userDb[0].ProviderName, "");
    strcpy(userDb[0].connparam, "");
    userDb[0].type = ESS_TYPE_USER;
    strcpy(userDb[0].AppName, AppName);
    strcpy(userDb[0].DbName, DbName);
    userDb[0].Access = ESS_PRIV_READ;
    userDb[0].MaxAccess = ESS_PRIV_READ;

    strcpy(userDb[1].UserName, "");
    strcpy(userDb[1].ProviderName, "");
    strcpy(userDb[1].connparam, "");
    userDb[1].type = ESS_TYPE_USER;
    strcpy(userDb[1].AppName, AppName);
    strcpy(userDb[1].DbName, DbName);
    userDb[1].Access = ESS_ACCESS_ADMIN;
    userDb[1].MaxAccess = ESS_ACCESS_ADMIN;

    sts = EssSetDatabaseAccessEx(hCtx, count, &userDb);
    printf(\"EssSetDatabaseAccessEx sts: %ld\n\n\", sts);

    userId = "IDUser1";
    AppName = AppName;
    DbName = DbName;
    type = ESS_TYPE_USER;
    bIsIdentity = ESS_TRUE;
sts = EssGetDatabaseAccessEx(hCtx, userId, bIsIdentity, type, AppName, DbName, &count, &pUserDb);
printf("EssGetDatabaseAccessEx sts: %ld\n", sts);
if(!sts)
{
    if(count && pUserDb)
    {
        DisplayUserDbInfo(pUserDb, count);
        sts = EssFree (hInst, pUserDb);
    }
    else
        printf ("\rUser Application list is empty\n\n");
}

return (sts);
}

See Also

- EssGetApplicationAccessEx
- EssListUsersInfoEx
- EssSetDatabaseAccessEx

## EssGetDatabaseInfo

Gets a database's information structure, which contains non user-configurable parameters for the database.

### Syntax

```
ESS_FUNC_M EssGetDatabaseInfo (hCtx, AppName, DbName, ppDbInfo);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>AppName</td>
<td>ESS_STR_T</td>
<td>Application name.</td>
</tr>
<tr>
<td>DbName</td>
<td>ESS_STR_T</td>
<td>Database name.</td>
</tr>
<tr>
<td>ppDbInfo</td>
<td>“ESS_DBINFO_T”</td>
<td>Address of pointer to receive allocated database info structure.</td>
</tr>
</tbody>
</table>

### Notes

- The memory allocated for the `ppDbInfo` structure should be freed using `EssFree()`.
- This function can only get the information structure for a server database.

### Return Value

If successful, this function returns a pointer to an allocated database info structure in `ppDbInfo`. 
Access

This function requires the caller to have at least read access (ESS_PRIV_READ) to the specified database.

Example

ESS_FUNC_M
ESS_GetDbInfo (ESS_HCTX_T hCtx,
    ESS_HINST_T hInst
)
{
    ESS_FUNC_M        sts = ESS_STS_NOERR;
    ESS_PDBINFO_T    DbInfo;
    ESS_STR_T        AppName;
    ESS_STR_T        DbName;
    AppName = "Sample";
    DbName = "Basic";

    sts = EssGetDatabaseInfo (hCtx, AppName,
    DbName, &DbInfo);
    if (!sts)
    {
        if (DbInfo)
        {
            EssFree (hInst, DbInfo);
        }
    }
    return(sts);
}

See Also

- EssGetApplicationInfo
- EssGetDatabaseInfoEx
- EssGetDatabaseState
- EssGetDatabaseStats

EssGetDatabaseInfoEx

Retrieves information for one or more databases, which contains non user-configurable parameters for the databases.

Syntax

ESS_FUNC_M EssGetDatabaseInfoEx (hCtx, AppName, DbName, pusCount; ppDbInfo);

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>AppName</td>
<td>ESS_STR_T</td>
<td>Name of application for which to return database information. If NULL,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>returns information for all applications and databases.</td>
</tr>
<tr>
<td>DbName</td>
<td>ESS_STR_T</td>
<td>Name of database for which to return database information. If NULL,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>returns information for all databases.</td>
</tr>
</tbody>
</table>
### Parameter Data Type Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pusCount</td>
<td>ESS_PUSHORT_T</td>
<td>Number of information structures to be returned</td>
</tr>
<tr>
<td>ppDbInfo</td>
<td>&quot;ESS_DBINFO_T&quot; on page 124</td>
<td>Pointer to array of information structures.</td>
</tr>
</tbody>
</table>

### Notes
- The memory allocated for the `ppDbInfo` structure should be freed using `EssFree()`.
- This function can only get the information structure for server databases.

### Return Value
If successful, this function returns an array of database information structures.

### Access
This function requires the caller to have at least read access (ESS_PRIV_READ) to the specified database.

### Example
```c
ESS_FUNC_M ESS_GetDatabaseInfoEx (ESS_HCTX_T hCtx, ESS_HINST_T hInst)
{
    ESS_FUNC_M   sts = ESS_STS_NOERR;
    ESS_STR_T    AppName;
    ESS_STR_T    DbName;
    ESS_PDBINFO_T DbInfo = NULL;
    ESS_USHORT_T Count;
    ESS_USHORT_T ind;

    AppName = "Sample";
    DbName  = "*";

    sts = EssGetDatabaseInfoEx(hCtx, AppName, DbName,
                               &Count, &DbInfo);

    if(!sts & DbInfo)
    {
        printf("\r\n------- Database Info Ex --------\r\n\r
        for(ind = 0; ind < Count; ind++)
        {
            printf("AppName: %s\r\n",DbInfo[ind].AppName);
            printf("DbName: %s\r\n",DbInfo[ind].Name);
            printf("DbType: %d\r\n",DbInfo[ind].DbType);
            printf("Status: %d\r\n",DbInfo[ind].Status);
            printf("nConnects: %d\r\n",DbInfo[ind].nConnects);
            printf("nLocks: %d\r\n",DbInfo[ind].nLocks);
        }
        EssFree(hInst, DbInfo);
    }
    return (sts);
}
```
See Also
- `EssGetApplicationInfo`
- `EssGetDatabaseInfo`
- `EssGetDatabaseState`
- `EssGetDatabaseStats`

**EssGetDatabaseNote**

Gets a database's note-of-the-day message.

**Syntax**

```c
ESS_FUNC_M EssGetDatabaseNote (hCtx,AppName,DbName,pDbNote);
```

**Parameter** | **Data Type** | **Description**
--- | --- | ---
`hCtx` | ESS_HCTX_T | API context handle.
`AppName` | ESS_STR_T | Application name.
`DbName` | ESS_STR_T | Database name.
`pDbNote` | ESS_PSTR_T | Address of pointer to receive allocated database note string.

**Notes**
- The note-of-the-day message may be used to display useful information about the database (whether data has been loaded, when it was last calculated, etc.) to users before they connect to the database.
- The database note string will always be less than 64 KB in length.
- The database's note is set by `EssSetDatabaseNote()`.
- The memory allocated for `pDbNote` should be freed using `EssFree()`.

**Return Value**

If successful, returns a pointer to an allocated database note string in `pDbNote`.

**Access**

This function requires the caller to have at least read access (ESS_PRIV_READ) to the specified database.

**See Also**
- `EssSetDatabaseNote`

**EssGetDatabaseState**

Gets a database's state structure, which contains user-configurable parameters for the database.
Syntax

ESS_FUNC_M EssGetDatabaseState (hCtx, AppName, DbName, ppDbState);

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle</td>
</tr>
<tr>
<td>AppName</td>
<td>ESS_STR_T</td>
<td>Application name</td>
</tr>
<tr>
<td>DbName</td>
<td>ESS_STR_T</td>
<td>Database name</td>
</tr>
<tr>
<td>ppDbState</td>
<td>“ESS_DBSTATE_T” on page 127</td>
<td>Address of pointer to receive allocated database state structure</td>
</tr>
</tbody>
</table>

Notes

- This function can get only a server database's state structure.
- The memory allocated for the `ppDbState` structure must be freed with `EssFree()`.

Return Value

If successful, this function returns a pointer to an allocated database state structure in `ppDbState`.

Access

To get a database's state structure, the connected user must have at least read access to the database.

Example

```c
ESS_FUNC_M ESS_GetCrType (ESS_HCTX_T hCtx, ESS_HINST_T hInst)
{
    ESS_FUNC_M        sts = ESS_STS_NOERR;
    ESS_PDBSTATE_T   pDbState;
    ESS_STR_T        AppName;
    ESS_STR_T        DbName;
    AppName = "Sample";
    DbName = "Basic";
    sts = EssGetDatabaseState (hCtx, AppName, DbName, &pDbState);
    if (!sts)
    {
        if (pDbState)
        {
            if (pDbState->CrDbName)
            {
                printf ("Currency Conversion Type Member:      %s\n", pDbState->CrTypeMember);
                if (pDbState->CrConvType == ESS_CRCTYPE_DIV)
                    printf ("Currency Conversion Type:             %s\n", "ESS_CRCTYPE_DIV");
                else if (pDbState->CrConvType == ESS_CRCTYPE_MULT)
                    printf ("Currency Conversion Type:             %s\n", "ESS_CRCTYPE_MULT");
```
} else
    printf ("No Currency database is set\r\n");
    EssFree (hInst, pDbState);
}
}
return (sts);

See Also

- EssGetApplicationState
- EssGetDatabaseInfo
- EssSetDatabaseState
- EssGetDatabaseStats

EssGetDatabaseStats

Gets a database's stats structure, which contains statistical information about the database.

Syntax

```
ESS_FUNC_M EssGetDatabaseStats (hCtx, AppName, DbName, ppDbStats);
```

Parameter | Data Type | Description
---|---|---
 hCtx | ESS_HCTX_T | API context handle
AppName | ESS_STR_T | Application name
DbName | ESS_STR_T | Database name
ppDbStats | "ESS_DBSTATS_T" on page 130 | Address of pointer to receive allocated database stats structure pointer

Notes

- This function can only be called for server databases.
- This function will load the database if it is not loaded.
- The memory allocated for ppDbStats should be freed using EssFree().

Return Value

If successful, this function returns a pointer to an allocated database stats structure in ppDbStats.

Access

This function requires the caller to have at least read access (ESS_PRIV_READ) to the database, and to have selected it as their active database using EssSetActive().

Example

```
ESS_FUNC_M
ESS_GetDbStats (ESS_HCTX_T  hCtx,
                ESS_HINST_T hInst
```


```c

ESS_FUNC_M        sts = ESS_STS_NOERR;
ESS_PDBSTATS_T    pDbStats;
ESS_STR_T        AppName;
ESS_STR_T         DbName;
AppName = "Sample";
DbName  = "Basic";

sts = EssGetDatabaseStats (hCtx, AppName,
      DbName, &pDbStats);
if (!sts)
{
    if (pDbStats)
    {
        EssFree (hInst, pDbStats);
    }
}
return(sts);
}

See Also

- EssGetDatabaseInfo
- EssGetDatabaseState

EssGetDefaultCalc

Gets the default calc script for the active database.

Syntax

```c

ESS_FUNC_M EssGetDefaultCalc (hCtx, pCalcScript);
```

Parameter | Data Type  | Description
----------|------------|-------------
hCtx      | ESS_HCTX_T | API context handle.
pCalcScript | ESS_PSTR_T  | Address of pointer to receive allocated calc script string.

Return Value

If successful, this function returns the default calc script for the database in `pCalcScript`.

- The returned calc script string will be less than 64 KB long.
- The memory allocated for `pCalcScript` should be freed using `EssFree()`.

Access

This function requires callers to have at least read access (ESS_PRIV_READ) to the database, and to have selected it as their active database using `EssSetActive()`.

```
Example

```c
ESS_FUNC_M
ESS_GetDefaultCalc (ESS_HCTX_T hCtx,
             ESS_HINST_T hInst
             )
{
    ESS_FUNC_M     sts = ESS_STS_NOERR;
    ESS_STR_T     cstr = NULL;
    sts = EssGetDefaultCalc(hCtx, &cstr);
    if (!sts)
    {
        if (cstr)
        {
            printf  (“Default Calc Script --\r\n%s\r\n”, cstr);
            EssFree (hInst, cstr);
        }
    }
    return (sts);
}
```

See Also

- EssDefaultCalc
- EssSetActive
- EssSetDefaultCalc
- EssSetDefaultCalcFile

### EssGetDimensionInfo

**Gets dimension information.**

**Syntax**

```c
ESS_FUNC_M EssGetDimensionInfo (hCtx, MbrName, pDims, ppDimInfo);
```

**Parameter** | **Data Type** | **Description**
--- | --- | ---
 hCtx | ESS_HCTX_T | API context handle
 MbrName | ESS_STR_T | Member name of dimension for which to return information. If NULL, returns information about every dimension. If member name is invalid, error results.
 pDims | ESS_PULONG_T | Pointer to the number of information structures returned
 ppDimInfo | “ESS_DIMENSIONINFO_T” on page 132 | Pointer to an array of information structures

**Notes**

- The constant values ESS_TTYPE_ATTRIBUTE and ESS_TTYPE_ATTRCALC for theDimTag field of the “ESS_DIMENSIONINFO_T” on page 132 structure indicate that the dimension is an attribute dimension.
The DimDataType field of the ESS_DIMENSIONINFO_T structure indicates the type of attribute dimension.

**Return Value**

If successful, returns an array of dimension information structures.

**Example**

```c
ESS_FUNC_M
ESS_GetDimensionInfo(ESS_HCTX_T hCtx, ESS_HINST_T hInst)
{
    ESS_FUNC_M sts = ESS_STS_NOERR;
    ESS_STR_T MbrName;
    ESS_ULONG_T nDims, ind;
    ESS_PDIMENSIONINFO_T DimInfo = NULL;

    MbrName = "Year";
    sts = EssGetDimensionInfo(hCtx, MbrName, &nDims,
        &DimInfo);

    if(!sts && DimInfo)
    {
        printf("-------- Dimension Information --------\n\n");
        for(ind = 0; ind < nDims; ind++)
        {
            printf("Dimension Name: %s\n", DimInfo[ind].DimName);
            printf("Dimension Number: %d\n", DimInfo[ind].DimNumber);

            switch (DimInfo[ind].DimType)
            {
                case ESS_DIMTYPE_DENSE:
                    printf("Dimension Type: %s\n","DENSE");
                    break;
                default:
                    printf("Dimension Type: %s\n","SPARSE");
                    break;
            }
            printf("\n");
        }
        EssFree(hInst, DimInfo);
    }
    return (sts);
}
```

**See Also**

- EssBuildDimension
- EssGetApplicationInfo
- EssGetApplicationInfoEx
- EssGetDatabaseInfo
- EssGetDatabaseInfoEx
**EssGetDrillThruURL**

Gets the drill-through URL within the active database outline.

**Syntax**

```c
ESS_FUNC_M EssGetDrillThruURL (hCtx, URLName, &pUrl);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle</td>
</tr>
<tr>
<td>URLName</td>
<td>ESS_STR_T</td>
<td>Drill-through URL name</td>
</tr>
<tr>
<td>pUrl</td>
<td>ESS_PDURLINFO_T</td>
<td>URL definition</td>
</tr>
</tbody>
</table>

**Return Value**

- If successful, gets the drill-through URL in the active database outline.
- If unsuccessful, returns an error code.

**Access**

- Caller must have database Read privilege (ESS_PRIV_READ) for the specified database.
- Caller must have selected the specified database as the active database using EssSetActive().

**Example**

```c
static void DisplayUrlDefn (ESS_PDURLINFO_T pUrls )
{
    ESS_UINT_T    i;

    printf("\tUrlname        : %s
", pUrls->cpURLName);
    if (pUrls->bIsLevel0)
        printf("\tUrl Is Level-0 slice : Yes\n");
    else
        printf("\tUrl Is Level-0 slice : No\n");

    printf("\tUrlXmlsize    : %i
", pUrls->iURLXmlSize);
    printf("\tUrlXml        : %s
", (ESS_STR_T) pUrls->cpURLXml);
    printf("\tNumber of drill region(s): %d
", pUrls->iCountOfDrillRegions);
    for ( i = 0; i < pUrls->iCountOfDrillRegions; i++ )
    {
        printf("\t	DrillRegion[%d]: %s
", i, pUrls->cppDrillRegions[i] );
    }
    printf("\n");
}
ESS_STS_T sts = ESS_STS_NOERR;
ESS_STR_T urlName = "";
ESS_USHORT_T usCountOfURLs, i;
ESS_PDURLINFO_T urlInfo;

/* Valid case*/

urlName = "Drill Through to EPMI";
sts = EssGetDrillThruURL(hCtx, urlName, &urlInfo);
```
printf("EssGetDrillThruURL sts: %ld\n",sts);
if(!sts)
    DisplayUrlDefn(urlInfo);

EssFreeStructure (hInst, ESS_DT_STRUCT_URLINFO, 1, (ESS_PVOID_T)urlInfo);

EssGetEssbaseSecurityMode
Displays the type of security in use: native or Shared Services mode.

**Syntax**

```c
ESS_FUNC_M EssGetEssbaseSecurityMode (hCtx, pMode);
```

**Parameter**  | **Data Type**  | **Description**  
---|---|---
 hCtx | ESS_HCTX_T | API context handle (input).
 pMode | ESS_PSECURITY_MODE_T | Address of variable to receive type of security in use.

**Return Value**

Returns 0 if successful; otherwise, returns an error.

**Example**

```c
/*
ESS_FUNC_M EssGetEssbaseSecurityMode (ESS_HCTX_T hCtx,
            ESS_PSECURITY_MODE_T mode);
*/
ESS_FUNC_M ESS_SS_GetEssbaseSecurityMode(ESS_HCTX_T  hCtx)
{
    ESS_STS_T           sts = ESS_STS_NOERR;
    ESS_SECURITY_MODE_T mode;

    sts = EssGetEssbaseSecurityMode(hCtx, &mode);

    if(sts)
    {
        printf("Failed to get Essbase Security mode.\n");
    }
    else
    {
        printf("Essbase Security Mode            : %d\n", mode);
    }
    return(sts);
}
```

See also an extended Appendix B

EssGetExtUser
Returns information about externally authenticated users.
Syntax

**EssGetExtUser** (hCtx, UserName, ppExtUserInfo);

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>UserName</td>
<td>ESS_STR_T</td>
<td>Name of user.</td>
</tr>
<tr>
<td>ppExtUserInfo</td>
<td>“ESS_USERINFOEX_T” on page 195</td>
<td>Address of pointer to receive allocated user info structure.</td>
</tr>
</tbody>
</table>

Notes

The memory allocated for *ppExtUserInfo* should be freed using EssFree().

Return Value

If successful, returns the user information structure in *ppExtUserInfo*.

Access

This function requires the caller to have Create/Delete User privilege (ESS_PRIV_USERCREATE) for the logged in server, unless they are getting their own user information.

See Also

- *EssListExtUsers*
- *EssSetExtUser*
- “ESS_USERINFOEX_T” on page 195

**EssGetFilter**

Starts getting the contents of a filter.

Syntax

**EssGetFilter** (hCtx, AppName, DbName, FilterName, pActive, pAccess);

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>AppName</td>
<td>ESS_STR_T</td>
<td>Application name.</td>
</tr>
<tr>
<td>DbName</td>
<td>ESS_STR_T</td>
<td>Database name.</td>
</tr>
<tr>
<td>FilterName</td>
<td>ESS_STR_T</td>
<td>Filter name.</td>
</tr>
<tr>
<td>pActive</td>
<td>ESS_PBOOL_T</td>
<td>Address of variable to receive filter active flag. If TRUE, the filter is currently in effect for the specified database.</td>
</tr>
<tr>
<td>pAccess</td>
<td>ESS_PACCESS_T</td>
<td>Address of variable to receive the default filter access level. For possible values, see “Bitmask Data Types (C)” on page 96.</td>
</tr>
</tbody>
</table>
Notes

This call must be followed by successive calls to EssGetFilterRow() to fetch the rows for the filter.

Return Value

If successful, returns the filter active flag in pActive, and the default filter access level in pAccess.

Access

This function requires the caller to have database designer privilege (ESS_PRIV_DBDESIGN) for the specified database.

Example

```c
ESS_FUNC_M
ESS_GetFilter (ESS_HCTX_T  hCtx, ESS_HINST_T hInst)
{
    ESS_FUNC_M       sts = ESS_STS_NOERR;
    ESS_STR_T      AppName;
    ESS_STR_T       DbName;
    ESS_STR_T       FilterName;
    ESS_BOOL_T      Active;
    ESS_ACCESS_T    Access;
    ESS_STR_T       RowString = NULL;
    ESS_STR_T       Acc_Str;

    AppName    = "Sample";
    DbName     = "Basic";
    FilterName = "Test";

    /*************************************************************************
    * Get Filter *
    *************************************************************************/
    sts = EssGetFilter(hCtx, AppName, DbName,
                       FilterName, &Active, &Access);

    /*************************************************************************/
    * Get Filter Rows *
    /*************************************************************************/
    if(!sts)
    {
        sts = EssGetFilterRow(hCtx, &RowString,
                               &Access);
        if(!sts && RowString)
        {
            printf("%s Filter Rows\r\n",FilterName);
            while(RowString)
            {
                switch (Access)
                {
                    case ESS_ACCESS_NONE:
                        Acc_Str = "NONE";
                        break;
                    case ESS_ACCESS_READ:
                        Acc_Str = "READ";
                        break;
                    default:
                        Acc_Str = "WRITE";
```
break;
}

printf("%s - %s\n",Acc_Str,RowString);
sts = EssGetFilterRow(hCtx, &RowString,
 &Access);
}
EssFree(hInst, RowString);
}
return (sts);

See Also

- EssGetFilterRow
- EssListFilters
- EssSetFilter

EssGetFilterList

Gets the list of users who are assigned a filter.

Syntax

ESS_FUNC_M EssGetFilterList (hCtx, AppName, DbName, FilterName, pCount, ppUserList);

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>AppName</td>
<td>ESS_STR_T</td>
<td>Application name.</td>
</tr>
<tr>
<td>DbName</td>
<td>ESS_STR_T</td>
<td>Database name.</td>
</tr>
<tr>
<td>FilterName</td>
<td>ESS_STR_T</td>
<td>Filter name.</td>
</tr>
<tr>
<td>pCount</td>
<td>ESS_PUSHORT_T</td>
<td>Address of variable to receive count of users assigned this filter.</td>
</tr>
<tr>
<td>ppUserList</td>
<td>ESS_PPUSERNAME_T</td>
<td>Address of pointer to receive allocated array of user names.</td>
</tr>
</tbody>
</table>

Notes

The memory allocated for ppUserList should be freed using EssFree().

Return Value

If successful, returns a count of the users assigned this filter in pCount, and an array of user names in ppUserList.

Access

This function requires the caller to have database designer privilege (ESS_PRIV_DBDESIGN) for the specified database.
Example

```c
ESS_STS_T ESS_GetFilterList (ESS_HCTX_T hCtx, ESS_HINST_T hInst)
{
    ESS_STS_T       sts = ESS_STS_NOERR;
    ESS_STR_T       AppName;
    ESS_STR_T       DbName;
    ESS_STR_T       FilterName;
    ESS_USHORT_T    Count = 0;
    ESS_USHORT_T    ind;
    ESS_PUSERNAME_T UserList = NULL;

    AppName    = "Sample";
    DbName     = "Basic";
    FilterName = "NewFilter";

    sts = EssGetFilterList(hCtx, AppName, DbName, 
                            FilterName, &Count, &UserList);
    if(!sts)
    {
        printf("--------%s User List---------\r\n\n", FilterName);
        if(Count && UserList)
            { 
            for (ind = 0; ind < Count; ind++)
                printf("%s\r\n",UserList[ind]);
                EssFree(hInst, UserList);
            }
        printf("\r\n");
        return (sts);
    }
}

See Also

- ESSGetFilter
- ESSListFilters
- ESSSetFilterList

EssGetFilterRow

Gets the next row of a filter.

Syntax

```c
ESS_FUNC_M EssGetFilterRow (hCtx, pRowString, pAccess);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>pRowString</td>
<td>ESS_PSTR_T</td>
<td>Address of pointer to receive the next row of the filter.</td>
</tr>
<tr>
<td>pAccess</td>
<td>ESS_PACCESS_T</td>
<td>Address of variable to receive the access level for the filter row. Possible values are listed in &quot;Bitmask Data Types (C)&quot; on page 96.</td>
</tr>
</tbody>
</table>
Notes

- This function should be called repeatedly after calling `EssGetFilter()`, until a NULL row string pointer is returned.
- The memory allocated for `pRowString` should be freed using `EssFree()`.

Return Value

If successful, returns the next filter row (if any) in `pRowString`, and the row access level in `pAccess`.

Access

This function requires the caller to have database designer privilege (ESS_PRIV_DBDESIGN) for the specified database.

Example

See the example of `EssGetFilter`.

See Also

- `EssGetFilter`
- `EssListFilters`

### EssGetGlobalState

Gets the server global state structure which contains parameters for system administration.

Syntax

```c
ESS_FUNC_M EssGetGlobalState (hCtx, ppGlobal);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>ppGlobal</td>
<td>“ESS_GLOBAL_T” on page 140</td>
<td>Address of pointer to receive allocated global state structure.</td>
</tr>
</tbody>
</table>

Notes

The memory allocated for `ppGlobal` should be freed using `EssFree()`.

Return Value

If successful, returns the current state of the server global state structure in `ppGlobal`.

Access

This function requires the caller to be a supervisor.

Example

```c
ESS_FUNC_M
EssGetGlobalState (ESS_HCTX_T hCtx, ESS_HINST_T hInst)
{
    ESS_FUNC_M     sts = ESS_STS_NOERR;
```
ESS_PGLOBAL_T pGlobal = NULL;

sts = EssGetGlobalState(hCtx, &pGlobal);

if(!sts & pGlobal)
{
    printf("-------- Global State --------\n\n\n");
    printf("Security->%d Logins->%d\r\n", 
pGlobal->Security, pGlobal->Logins);
    printf("Access->%d Validity->%d\r\n", 
pGlobal->Access, pGlobal->Validity);
    printf("Currency->%d PwMin->%d\r\n", 
pGlobal->Currency,pGlobal->PwMin);
    printf("InactivityTime->%ld InactivityCheck->%ld\r\n", pGlobal->InactivityTime, 
pGlobal->InactivityCheck);

    EssFree(hInst, pGlobal);
}
return (sts);

See Also

- EssSetGlobalState

**EssGetGroup**

Gets a group information structure, which contains security information for the group.

**Syntax**

```c
ESS_FUNC_M EssGetGroup (hCtx, GroupName, ppGroupInfo);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>GroupName</td>
<td>ESS_STR_T</td>
<td>Group name.</td>
</tr>
<tr>
<td>ppGroupInfo</td>
<td>“ESS_USERINFO_T, ESS_GROUPINFO_T”</td>
<td>Address of pointer to receive allocated group info structure (output).</td>
</tr>
</tbody>
</table>

**Notes**

The memory allocated for *ppGroupInfo* should be freed using EssFree().

**Return Value**

If successful, returns the group information structure in *ppGroupInfo*.

**Access**

This function requires the caller to have Create/Delete User privilege (ESS_PRIV_USERCREATE) for the logged in server.
See Also

- EssGetGroupInfoEx
- EssListGroups
- EssSetGroup

EssGetGroupInfoEx

Gets a group information structure, which contains security information for the group. Similar to EssGetGroup, but can accept a user directory specification or unique identity attribute for GroupID.

Syntax

```c
ESS_FUNC_M EssGetGroupInfoEx (hCtx, GroupId, bisIdentity, ppGroupInfo);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle (input).</td>
</tr>
<tr>
<td>GroupId</td>
<td>ESS_STR_T</td>
<td>Group name (input). Can be specified as <code>groupname@provider</code> or as a unique identity attribute.</td>
</tr>
<tr>
<td>bisIdentity</td>
<td>ESS_BOOL_T</td>
<td>Input. Indicates if <code>GroupId</code> is a name or an identity. If TRUE, <code>GroupId</code> is an identity.</td>
</tr>
<tr>
<td>ppGroupInfo</td>
<td>ESS_PGROUPINFOID_T</td>
<td>Address of pointer to receive allocated group info structure (output).       The group list structure can include user directories and unique identity attributes.</td>
</tr>
</tbody>
</table>

Notes

The memory allocated for `ppGroupInfo` should be freed using EssFree().

Return Value

If successful, returns the group information structure in `ppGroupInfo`.

Access

This function requires the caller to have Create/Delete User privilege (ESS_PRIV_USERCREATE) for the logged in server.

Example

```c
void DisplayGroupsInfoEx(ESS_GROUPINFOID_T groupInfo)
{
    ESS_STS_T sts = ESS_STS_NOERR;
    ESS_BOOL_T isDefined = ESS_TRUE;

    printf("\tUser Name: %s\n", groupInfo.Name);
    printf("\tProvider Name: %s\n", groupInfo.ProviderName);
    printf("\tIdentity: %s\n", groupInfo.connparam);
    printf("\tDescription: %s\n", groupInfo.Description);
    printf("\tEMail Identification: %s\n", groupInfo.EMailID);
```
if (groupInfo.LockedOut)
    printf("Locked out: Yes\n");
else
    printf("Locked out: No\n");

if (groupInfo.PwdChgNow)
    printf("Change the password now: Yes\n");
else
    printf("Change the password now: No\n");

printf("Password: %s\n", groupInfo.Password);
printf("Application: %s\n", groupInfo.AppName);
printf("Database: %s\n", groupInfo.DbName);

if (groupInfo.Login)
    printf("Logged in: Yes\n");
else
    printf("Logged in: No\n");

switch(groupInfo.Access)
{
    case ESS_ACCESS_ADMIN:
        printf("Access: %d - ESS_ACCESS_ADMIN\n", groupInfo.Access);
        break;
    case ESS_ACCESS_APPALL:
        printf("Access: %d - ESS_ACCESS_APPALL\n", groupInfo.Access);
        break;
    case ESS_ACCESS_DBALL:
        printf("Access: %d - ESS_ACCESS_DBALL\n", groupInfo.Access);
        break;
    case ESS_ACCESS_APPCREATE:
        printf("Access: %d - ESS_ACCESS_APPCREATE\n", groupInfo.Access);
        break;
    case ESS_ACCESS_APPMANAGE:
        printf("Access: %d - ESS_ACCESS_APPMANAGE\n", groupInfo.Access);
        break;
    case ESS_ACCESS_DBCREATE:
        printf("Access: %d - ESS_ACCESS_DBCREATE\n", groupInfo.Access);
        break;
    case ESS_ACCESS_DBMANAGE:
        printf("Access: %d - ESS_ACCESS_DBMANAGE\n", groupInfo.Access);
        break;
    case ESS_ACCESS_CALC:
        printf("Access: %d - ESS_ACCESS_CALC\n", groupInfo.Access);
        break;
    case ESS_ACCESS_WRITE:
        printf("Access: %d - ESS_ACCESS_WRITE\n", groupInfo.Access);
        break;
    case ESS_ACCESS_READ:
        printf("Access: %d - ESS_ACCESS_READ\n", groupInfo.Access);
        break;
    case ESS_PRIV_APPCREATE:
        printf("Access: %d - ESS_PRIV_APPCREATE\n", groupInfo.Access);
        break;
    case ESS_PRIV_USERCREATE:
        printf("Access: %d - ESS_PRIV_USERCREATE\n", groupInfo.Access);
        break;
}
break;
case ESS_PRIV_APPMANAGE:
    printf("\tAccess: %d - ESS_PRIV_APPMANAGE\n", groupInfo.Access);
    break;
case ESS_PRIV_APPLOAD:
    printf("\tAccess: %d - ESS_PRIV_APPLOAD\n", groupInfo.Access);
    break;
case ESS_PRIV_DBCREATE:
    printf("\tAccess: %d - ESS_PRIV_DBCREATE\n", groupInfo.Access);
    break;
case ESS_PRIV_DBMANAGE:
    printf("\tAccess: %d - ESS_PRIV_DBMANAGE\n", groupInfo.Access);
    break;
case ESS_PRIV_DBLOAD:
    printf("\tAccess: %d - ESS_PRIV_DBLOAD\n", groupInfo.Access);
    break;
case ESS_PRIV_CALC:
    printf("\tAccess: %d - ESS_ACCESS_DBALL\n", groupInfo.Access);
    break;
case ESS_PRIV_WRITE:
    printf("\tAccess: %d - ESS_PRIV_WRITE\n", groupInfo.Access);
    break;
case ESS_PRIV_READ:
    printf("\tAccess: %d - ESS_PRIV_READ\n", groupInfo.Access);
    break;
case ESS_PRIV_NONE:
    printf("\tAccess: %d - ESS_PRIV_NONE\n", groupInfo.Access);
    break;
default:
    printf("\tAccess: Unknown\n");
}

switch(groupInfo.MaxAccess)
{
    case ESS_ACCESS_ADMIN:
        printf("\tMax Access: %d - ESS_ACCESS_ADMIN\n", groupInfo.MaxAccess);
        break;
    case ESS_ACCESS_APPALL:
        printf("\tMax Access: %d - ESS_ACCESS_APPALL\n", groupInfo.MaxAccess);
        break;
    case ESS_ACCESS_DBALL:
        printf("\tMax Access: %d - ESS_ACCESS_DBALL\n", groupInfo.MaxAccess);
        break;
    case ESS_ACCESS_APPCREATE:
        printf("\tMax Access: %d - ESS_ACCESS_APPCREATE\n", groupInfo.MaxAccess);
        break;
    case ESS_ACCESS_APPMANAGE:
        printf("\tMax Access: %d - ESS_ACCESS_APPMANAGE\n", groupInfo.MaxAccess);
        break;
    case ESS_ACCESS_DBCREATE:
        printf("\tMax Access: %d - ESS_ACCESS_DBCREATE\n", groupInfo.MaxAccess);
        break;
    case ESS_ACCESS_DBMANAGE:
        printf("\tMax Access: %d - ESS_ACCESS_DBMANAGE\n", groupInfo.MaxAccess);
        break;
    case ESS_ACCESS_CALC:
        printf("\tMax Access: %d - ESS_ACCESS_CALC\n", groupInfo.MaxAccess);
break;
case ESS_ACCESS_WRITE:
    printf("\tMax Access: %d - ESS_ACCESS_WRITE\n", groupInfo.MaxAccess);
    break;
case ESS_ACCESS_READ:
    printf("\tMax Access: %d - ESS_ACCESS_READ\n", groupInfo.MaxAccess);
    break;
case ESS_PRIV_USERCREATE:
    printf("\tMax Access: %d - ESS_PRIV_USERCREATE\n", groupInfo.MaxAccess);
    break;
case ESS_PRIV_APPCREATE:
    printf("\tMax Access: %d - ESS_PRIV_APPCREATE\n", groupInfo.MaxAccess);
    break;
case ESS_PRIV_APPMANAGE:
    printf("\tMax Access: %d - ESS_PRIV_APPMANAGE\n", groupInfo.MaxAccess);
    break;
case ESS_PRIV_APPLOAD:
    printf("\tMax Access: %d - ESS_PRIV_APPLOAD\n", groupInfo.MaxAccess);
    break;
case ESS_PRIV_DBCREATE:
    printf("\tMax Access: %d - ESS_PRIV_DBCREATE\n", groupInfo.MaxAccess);
    break;
case ESS_PRIV_DBMANAGE:
    printf("\tMax Access: %d - ESS_PRIV_DBMANAGE\n", groupInfo.MaxAccess);
    break;
case ESS_PRIV_DBLOAD:
    printf("\tMax Access: %d - ESS_PRIV_DBLOAD\n", groupInfo.MaxAccess);
    break;
case ESS_PRIV_CALC:
    printf("\tMax Access: %d - ESS_ACCESS_DBALL\n", groupInfo.MaxAccess);
    break;
case ESS_PRIV_WRITE:
    printf("\tMax Access: %d - ESS_PRIV_WRITE\n", groupInfo.MaxAccess);
    break;
case ESS_PRIV_READ:
    printf("\tMax Access: %d - ESS_PRIV_READ\n", groupInfo.MaxAccess);
    break;
case ESS_PRIV_NONE:
    printf("\tMax Access: %d - ESS_PRIV_NONE\n", groupInfo.MaxAccess);
    break;
default:
    printf("\tMax Access: Unknown\n");
}

printf("\tPassword Expiration in Dates: %d\n", groupInfo.Expiration);
printf("\tFailed Login Attempts Since Then: %d\n", groupInfo.FailCount);
printf("\tLogin ID: %d\n", groupInfo.LoginId);
printf("\tProtocol: %s\n", groupInfo.protocol);
printf("\tConnection Parameter: %s\n", groupInfo.connparam);
printf( "\n");

ESS_FUNC_M ESS_GetGroupInfoEx (ESS_HCTX_T hCtx)
{
    ESS_STS_T sts = ESS_STS_NOERR;
ESS_STR_T groupId;
ESS_BOOL_T bisIdentity;
ESS_PGROUPINFOID_T groupInfo;

groupId = "IDAdminGroup@ldap";
bisIdentity = ESS_TRUE;
sts = EssGetGroupInfoEx(hCtx, groupId, bisIdentity, &groupInfo);
printf("EssGetGroupInfoEx sts: %ld\n", sts);
if(!sts && groupInfo)
{
    DisplayGroupsInfoEx(*groupInfo);
}

return (sts);

See Also
- EssListGroupsInfoEx

**EssGetGroupList**

Gets the list of users who are members of a group (or the list of groups to which a user belongs).

**Syntax**

ESS_FUNC_M EssGetGroupList (hCtx, GroupName, pCount, ppUserList);

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>GroupName</td>
<td>ESS_USERNAME_T</td>
<td>User name or group name.</td>
</tr>
<tr>
<td>pCount</td>
<td>ESS_PUSHORT_T</td>
<td>Address of variable to receive count of user names.</td>
</tr>
<tr>
<td>ppUserList</td>
<td>ESS_PPUSERNAME_T</td>
<td>Address of pointer to receive allocated array of user name strings.</td>
</tr>
</tbody>
</table>

**Notes**

- This function can also be used to get the list of groups to which a user belongs, by using a user name as the GroupName argument.
- The memory allocated for ppUserList should be freed using EssFree().

**Return Value**

If successful, returns a count of user names in pCount, and a array of user name strings in ppUserList.

**Access**

This function requires the caller to have Create/Delete User privilege (ESS_PRIV_USERCREATE) for the logged in server, unless they are a user getting their own list of groups.
Example

ESS_FUNC_M
ESS_ListGroupUsers (ESS_HCTX_T hCtx,
ESS_HINST_T hInst
)
{
    ESS_FUNC_M        sts = ESS_STS_NOERR;
    ESS_PUSERNAME_T  UserList = NULL;
    ESS_USHORT_T     ind;
    ESS_USHORT_T     Items;
    ESS_USERNAME_T   GroupName;
    strcpy (GroupName, "PowerUsers");
    sts = EssGetGroupList (hCtx, GroupName, &Items, &UserList);
    if (!sts)
    {
        if (Items && UserList)
        {
            printf ("-------%s User List-------\n", GroupName);
            for (ind = 0; ind < Items; ind++)
            {
                if (UserList [ind])
                    printf ("%s\n", UserList [ind]);
            }
            EssFree (hInst, UserList);
        }
        else
            printf ("Users list is empty\n");
    }
    return (sts);
}

See Also

- EssGetGroupListEx
- EssAddToGroup
- EssDeleteFromGroup
- EssListGroups
- EssSetGroupList

**EssGetGroupListEx**

Gets the list of users who are members of a group or the list of groups to which the user belongs. Similar to **EssGetGroupList**, but can accept a user directory specification or unique identity attribute for *GroupName*.

Syntax

**ESS_FUNC_M** EssGetGroupListEx (hCtx, GroupName, bIsIdentity, entityType, pCount, bOutputInIds, ppUserList);

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle (input).</td>
</tr>
<tr>
<td>Parameter</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>GroupName</td>
<td>ESS_STR_T</td>
<td>Group name or identity (input). Can be specified as <code>groupname@provider</code> or as a unique identity attribute.</td>
</tr>
<tr>
<td>bIsIdentity</td>
<td>ESS_BOOL_T</td>
<td>Input. Indicates if <code>GroupName</code> is a name or an identity. If <code>TRUE</code>, <code>GroupName</code> is an identity.</td>
</tr>
</tbody>
</table>
| entityType | ESS_USHORT_T | Type of entity (input). Can be one of the following:  
| | | - ESS_TYPE_USER – Returns the list of groups to which this user belongs  
| | | - ESS_TYPE_GROUP – Returns the list of users to which this group belongs |
| pCount | ESS_PUSHORT_T | Address of variable to receive count of user names (output). |
| bOutputInIds | ESS_BOOL_T | Input. Indicates whether the output must be in identities. If `TRUE`, `ppUserList` returns an array of identities. |
| ppUserList | ESS_PSTR_T | Address of pointer to receive allocated array of user name strings or identities (output). |

**Notes**

- This function can also be used to get the list of groups to which a user belongs, by using a user name as the `GroupName` argument.
- The memory allocated for `ppUserList` should be freed using `EssFree()`. 

**Return Value**

If successful, returns a count of user names in `pCount`, and an array of user name strings or identities in `ppUserList`. 

**Access**

This function requires the caller to have Create/Delete User privilege (ESS_PRIV_USERCREATE) for the logged in server, unless they are a user getting their own list of groups. 

**Example**

```c
void DisplayUserList(ESS_USHORT_T count, ESS_PSTR_T UserList)
{
    ESS_USHORT_T i;
    for (i = 0; i < count; i++)
    {
        if (UserList [i])
            printf ("%s\n", UserList[i]);
    }
}

ESS_FUNC_M ESS_ListGroupUsers (ESS_HCTX_T hCtx, ESS_HINST_T hInst)
{
    ESS_STS_T sts = ESS_STS_NOERR;
    ESS_STR_T groupId;
```
ESS_BOOL_T bisIdentity;
ESS_USHORT_T type;
ESS_USHORT_T count;
ESS_BOOL_T bUsingIdentity;
ESS_PSTR_T pUserList;

groupId = "IDAdminGroup";
bisIdentity = ESS_TRUE;
type = ESS_TYPE_GROUP;
sts = EssGetGroupListEx(hCtx, groupId, bisIdentity, type, &count, &bUsingIdentity, &pUserList);
printf("EssGetGroupListEx sts: %ld\n", sts);
if(!sts)
{
    if(pUserList)
    {
        printf ("\n---User/Group list for %s:\n", groupId);
        DisplayUserList(count, pUserList);
    }
    else
        printf ("\tUser list is empty\n");
}
return (sts);

See Also

- EssAddToGroupEx
- EssDeleteFromGroupEx
- EssListGroupsInfoEx

**EssGetIBH**

Creates a local log file with all index combinations for which blocks contain invalid block headers. The database administrator can use this information to reload the datapoints that were identified as corrupted.

**Syntax**

```c
ESS_FUNC_M EssGetIBH (hCtx, destFileName);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx;</td>
<td>ESS_HCTX_T</td>
<td>Context handle</td>
</tr>
<tr>
<td>destFileName;</td>
<td>ESS_STR_T</td>
<td>Name of the file in which the IBH information is to be stored at client side.</td>
</tr>
</tbody>
</table>

See Also

- EssLocateIBH
- EssFixIBH
**EssGetLocalPath**

Gets the full local file path for a specific object file on the client.

**Syntax**

```c
ESS_FUNC_M EssGetLocalPath (hCtx, ObjType, AppName, DbName, ObjName, Create, pPath);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle returned by EssCreateLocalContext()</td>
</tr>
<tr>
<td>ObjType</td>
<td>ESS_OBJTYPE_T</td>
<td>Object type (must be single type). See “Bitmask Data Types (C)” on page 96 for a list of object types.</td>
</tr>
<tr>
<td>AppName</td>
<td>ESS_STR_T</td>
<td>Application name or NULL (ESS_NULL). If NULL, command assumes a file name, and EssGetLocalPath() returns the ObjName in pPath as is.</td>
</tr>
<tr>
<td>DbName</td>
<td>ESS_STR_T</td>
<td>Database name. If NULL, uses the application subdirectory.</td>
</tr>
<tr>
<td>ObjName</td>
<td>ESS_STR_T</td>
<td>Object name or file name if AppName is NULL. ObjName is not parsed for correctness; no suffix is appended to the path.</td>
</tr>
<tr>
<td>Create</td>
<td>ESS_BOOL_T</td>
<td>Create directories flag. If TRUE, the appropriate application and database subdirectories will be created if necessary. If FALSE, and the directories do not exist, an error will be generated.</td>
</tr>
<tr>
<td>pPath</td>
<td>ESS_PSTR_T</td>
<td>Address of pointer to receive allocated local path name string</td>
</tr>
</tbody>
</table>

**Notes**

The memory allocated for pPath should be freed using EssFree().

**Return Value**

If successful, returns the full path name of the appropriate object file in pPath.

**Access**

This function requires no special privileges.

**Example**

```c
ESS_VOID_T ESS_GetLocalPath (ESS_HINST_T hInst)
{
    ESS_FUNC_M         sts = ESS_STS_NOERR;
    ESS_HCTX_T        hLocalCtx;
    ESS_STR_T         AppName;
    ESS_STR_T         DbName;
    ESS_STR_T         ObjName;
    ESS_OBJTYPE_T     ObjType;
    ESS_BOOL_T        Create;
    ESS_STR_T         Path;

    AppName = "Sample";
    DbName  = "Basic";
    ObjName = "Basic";
    ObjType = ESS_OBJTYPE_OUTLINE;
    ```
Create = ESS_TRUE;
sts = EssCreateLocalContext(hInst, NULL, NULL, &hLocalCtx);
if(!sts && hLocalCtx)
{
    sts = EssGetLocalPath(hLocalCtx, ObjType, 
                         AppName, DbName, ObjName, Create, &Path);
    if(!sts)
    { 
        if(*Path)
        {
            printf("Path: %s\r\n", Path); 
            EssFree(hInst,Path);
        }
    }
}
if(hLocalCtx) 
    sts = EssDeleteLocalContext(hLocalCtx);
}

See Also
- EssCreateLocalContext
- EssListObjects

EssGetLocationAliasList

Returns a list of all currently-defined location aliases, together with lists of the host names, application names, database names and user names to which the location aliases are mapped.

Syntax

\[
\text{ESS\_FUNC\_M} \quad \text{EssGetLocationAliasList} \quad (\text{hCtx, pusListCnt, ppAliasNames, ppHostNames, ppAppNames, ppDbNames, ppUserNames});
\]

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle</td>
</tr>
<tr>
<td>pusListCnt</td>
<td>ESS_USHORT_T</td>
<td>Number of location aliases returned</td>
</tr>
<tr>
<td>ppAliasNames</td>
<td>ESS_PSTR_T *</td>
<td>Location alias name buffer</td>
</tr>
<tr>
<td>ppHostNames</td>
<td>ESS_PSTR_T *</td>
<td>Host name buffer</td>
</tr>
<tr>
<td>ppAppNames</td>
<td>ESS_PSTR_T *</td>
<td>Application name buffer</td>
</tr>
<tr>
<td>ppDbNames</td>
<td>ESS_PSTR_T *</td>
<td>Database name buffer</td>
</tr>
<tr>
<td>ppUserNames</td>
<td>ESS_PSTR_T *</td>
<td>User login name buffer</td>
</tr>
</tbody>
</table>

Notes
- hCtx is the only input parameter.
- `pusListCnt`, `ppAliasNames`, `ppHostNames`, `ppAppName`, `ppDbNames` and `ppUserName` are output parameters; that is, values returned.

- After you call this function, you must call `EssFree()` to free the memory used by the returned lists.

**See Also**
- `EssCreateLocationAlias`
- `EssDeleteLocationAlias`

### EssGetLogFile

Copies all or part of an application log file (`appname.log`) or the Essbase Server log file (`essbase.log`) from the server to the client.

**Syntax**

```c
ESS_FUNC_M EssGetLogFile (hCtx, AppName, TimeStamp, LocalName);
```

**Parameter** | **Data Type** | **Description**
--- | --- | ---
`hCtx` | ESS_HCTX_T | API context handle.
`AppName` | ESS_STR_T | Application name or NULL. If NULL, `EssGetLogFile` accesses the Essbase Server log file (`essbase.log`).
`TimeStamp` | ESS_TIME_T | Time stamp, indicating date and time of earliest log file entry required. If `TimeStep` is set to 0 (zero), `EssGetLogFile` copies the entire log file.
`LocalName` | ESS_STR_T | Full path name of local destination file on client.

**Notes**

- `TimeStamp` represents the number of seconds elapsed since midnight (00:00:00) Greenwich Mean Time on January 1, 1970. This function copies to the client only log file entries occurring after the date & time specified by `TimeStamp`.

- For the locations of `essbase.log` and `appname.log`, see the *Oracle Essbase Database Administrator’s Guide*.

**Return Value**

If successful, the file is copied to the local file specified by `LocalName`.

**Access**

This function requires the caller to have application Design privilege (ESS_PRIV_APPDESIGN), or database Design privilege (ESS_PRIV_DBDESIGN) for the specified application or any of its databases.

**Example**

```c
ESS_FUNC_M
ESS_GetLogFile (ESS_HCTX_T hCtx) {
```
EssGetMemberCalc

Gets the calc equation for a specific member in the active database outline.

Syntax

```c
ESS_FUNC_M EssGetMemberCalc (hCtx, MbrName, pCalcStr, pLastCalcStr);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>MbrName</td>
<td>ESS_STR_T</td>
<td>Member name.</td>
</tr>
<tr>
<td>pCalcStr</td>
<td>ESS_PSTR_T</td>
<td>Address of pointer to receive allocated member calc string.</td>
</tr>
<tr>
<td>pLastCalcStr</td>
<td>ESS_PSTR_T</td>
<td>Address of pointer to receive allocated member last calc string.</td>
</tr>
</tbody>
</table>

Notes

- The last calc string is the formula used to calculate the member the last time the database was calculated. It might be left from `pCalStr` if a calc script was used to calculate the database.
- This function checks whether the relational span Boolean is set and can determine if members stored in attached relational data sets have calc strings, but returns a NULL string instead of the calc string.
- The memory allocated for `pCalcStr` and `pLastCalcStr` should be freed using `EssFree()`.

Return Value

If successful, this function returns the calc string and last calc string in `pCalcStr` and `pLastCalcStr`.
Access

This function requires the caller to have at least read access (ESS_PRIV_READ) to the database, and to have selected it as their active database using EssSetActive().

Example

```c
ESS_FUNC_M
ESS_GetMbrCalc (ESS_HCTX_T hCtx,
                  ESS_HINST_T hInst
               )
{
    ESS_FUNC_M     sts = ESS_STS_NOERR;
    ESS_STR_T     calcStr, lastCalcStr;

    calcStr = lastCalcStr = NULL;
    sts = EssGetMemberCalc(hCtx, "Year", &calcStr, &lastCalcStr);
    if (!sts)
    {
        if (calcStr)
        {
            printf ("Outline Defined Calc Equation -- [%s]\r\n", calcStr);
        }
        else
        {
            printf ("Outline Defined Calc Equation -- [Default Rollup]\r\n");
        }
        if (lastCalcStr)
        {
            printf ("Last Calculated Calc Equation -- [%s]\r\n", lastCalcStr);
        }
        else
        {
            if (calcStr)
            {
                printf ("Last Calculated Calc Equation -- [%s]\r\n", calcStr);
            }
            else
            {
                printf ("Last Calculated Calc Equation -- [Default Rollup]\r\n");
            }
        }
    }
    if (calcStr)
    EssFree (hInst, calcStr);
    if (lastCalcStr)
    EssFree (hInst, lastCalcStr);

    return (sts);
}
```

See Also

- EssGetMemberInfo
- EssSetActive
EssGetMemberInfo

Gets a structure containing information about a specific member in the active database outline.

Syntax

```c
ESS_FUNC_M EssGetMemberInfo (hCtx, MbrName, ppMbrInfo);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>MbrName</td>
<td>ESS_STR_T</td>
<td>Member name.</td>
</tr>
<tr>
<td>ppMbrInfo</td>
<td>“ESS_MEMBERINFO_T” on page 149</td>
<td>Address of pointer to receive allocated member information structure.</td>
</tr>
</tbody>
</table>

Notes

- Call `EssFree()` or `EssFreeStructure()` to free memory dynamically allocated for `ppMbrInfo`.

  For an attribute member of type ESS_ATTRMBRDT_STRING, you must call `EssFreeStructure()` to free memory dynamically allocated for `ppMbrInfo`. Specify `ESS_DT_STRUCT_MBRINFO` as the structure ID. `EssFree()` will not free memory dynamically allocated for an attribute string value.

- The ESS_MBRSTS_ATTRIBUTE constant for the Status field of the “ESS_MEMBERINFO_T” on page 149 structure indicates that the dimension or member is an attribute dimension or attribute member.

- Two fields of the ESS_MEMBERINFO_T structure are for attributes only:
  - fAttributed
  - Attribute

- This function checks whether the relational span Boolean is set (set by `EssSetSpanRelationalPartition`) and can return information on members in the relational store.

- Two fields of the “ESS_MEMBERINFO_T” on page 149 structure are used only for members in relational stores:
  - fHasRelDesc
  - fHasRelPartEnabled

- Two fields of the “ESS_MBRINFO_T” on page 699 structure are used only for members in relational stores:
  - fHasRelDesc
  - fHasRelPartEnabled

Return Value

If successful, this function returns an allocated member information structure, `ppMbrInfo`. If a member has no parent, this function returns an empty string in the ParentMbrName field of the ESS_MEMBERINFO_T structure.
**Access**

This function requires the caller to have at least read access (ESS_PRIV_READ) to the database, and to have selected it as their active database using EssSetActive().

**Example**

```c
ESS_FUNC_M
ESS_GetMbrInfo (ESS_HCTX_T  hCtx,
                ESS_HINST_T hInst
)
{
    ESS_FUNC_M         sts = ESS_STS_NOERR;
    ESS_MEMBERINFO_T  *pMbrInfo = NULL;
    sts = EssGetMemberInfo(hCtx, "Profit",
                           &pMbrInfo);
    if (!sts)
    {
        if (pMbrInfo)
        {
            EssFreeStructure(hCtx, structId, count, structPtr);
        }
    }
    return (sts);
}
```

**See Also**

- EssCheckMemberName
- EssFreeStructure
- EssGetMemberCalc
- EssQueryDatabaseMembers
- EssSetActive

---

**EssGetObject**

Copies an object from the server or client object system to a local file, and optionally locks it.

**Syntax**

```c
ESS_FUNC_M EssGetObject (hCtx, ObjType,AppName, DbName, ObjName, LocalName, Lock);
```

**Parameter** | **Data Type** | **Description** |
--- | --- | --- |
`hCtx` | ESS_HCTX_T | API context handle. Can be local context handle returned by EssCreateLocalContext(). |
`ObjType` | ESS_OBJTYPE_T | Object type (must be single type). Refer to “Bitmask Data Types (C)” on page 96 for a list of possible values. |
`AppName` | ESS_STR_T | Application name. |
`DbName` | ESS_STR_T | Database name. If NULL, uses the application subdirectory. |
`ObjName` | ESS_STR_T | Name of object to get. |
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LocalName</td>
<td>ESS_STR_T</td>
<td>Full path name of local destination file on client.</td>
</tr>
<tr>
<td>Lock</td>
<td>ESS_BOOL_T</td>
<td>Flag to control object locking. If TRUE, the server object is locked to prevent updates by other users.</td>
</tr>
</tbody>
</table>

**Notes**

To lock an object, it must already exist on the server and not be locked by another user. Locking is not supported on the client.

**Return Value**

If successful, the object is copied to the local file specified by `LocalName`.

**Access**

This function requires the caller to have the appropriate level of access to the specified application and/or database containing the object (depending on the object type). To lock the object (lock flag is TRUE), the caller must have application or Database Designer privilege (ESS_PRIV_APPDESIGN or ESS_PRIV_DBDESIGN) for the specified application or database containing the object.

**Example**

```c
ESS_FUNC_M
ESS_GetObject (ESS_HCTX_T hCtx)
{
    ESS_FUNC_M         sts = ESS_STS_NOERR;
    ESS_STR_T         AppName;
    ESS_STR_T         DbName;
    ESS_STR_T         ObjName;
    ESS_STR_T         LocalName;
    ESS_OBJTYPE_T     ObjType;
    ESS_BOOL_T        Lock;

    AppName = "Sample";
    DbName  = "Basic";
    ObjName = "Basic";
    ObjType = ESS_OBJTYPE_OUTLINE;
    LocalName = "C:\Hyperion\products\Essbase\EssbaseClient\client\Basic.otl";
    Lock    = ESS_TRUE;

    sts = EssGetObject (hCtx, ObjType, AppName, DbName, ObjName, LocalName, Lock);
    return (sts);
}
```

**See Also**

- `EssGetObjectInfo`
- `EssListObjects`
- `EssLockObject`
- `EssPutObject`
EssUnlockObject

**EssGetObjectInfo**

Gets information about a specific object on the server or locally on the client.

**Syntax**

```c
ESS_FUNC_M EssGetObjectInfo (hCtx, ObjType, AppName, DbName, ObjName, ppObject);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle. Can be local context handle returned by EssCreateLocalContext().</td>
</tr>
<tr>
<td>ObjType</td>
<td>ESS_OBJTYPE_T</td>
<td>Object type (must be single type). Refer to “Bitmask Data Types (C)” on page 96 for a list of possible values.</td>
</tr>
<tr>
<td>AppName</td>
<td>ESS_STR_T</td>
<td>Application name.</td>
</tr>
<tr>
<td>DbName</td>
<td>ESS_STR_T</td>
<td>Database name. If NULL, uses the application subdirectory.</td>
</tr>
<tr>
<td>ObjName</td>
<td>ESS_STR_T</td>
<td>Object name.</td>
</tr>
<tr>
<td>ppObject</td>
<td>ESS_PPOBJINFO_T</td>
<td>Address of pointer to receive allocated object info structure.</td>
</tr>
</tbody>
</table>

**Notes**

The memory allocated for `ppObject` should be freed using EssFree().

**Return Value**

If successful, returns an object structure containing information about the appropriate object in `ppObject`.

**Access**

This function requires the caller to have the appropriate level of access to the specified application and/or database containing the object (depending on the object type).

**See Also**

- EssGetObject
- EssListObjects

**EssGetProcessState**

Gets the current state of an asynchronous process, such as a calculate or a data import.

**Syntax**

```c
ESS_FUNC_M EssGetProcessState (hCtx, pProcState);
```
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>pProcState</td>
<td>“ESS_PROCSTATE_T” on page 178</td>
<td>Pointer to process state structure</td>
</tr>
</tbody>
</table>

**Notes**

- Your program should call this function at regular intervals (between 5 & 10 seconds) until it returns ESS_STATE_DONE in `pProcState`.
- Calling this function except after initiating a successful asynchronous database operation, for example, a calculation, generates an error.
- The memory allocated for `pProcState` should be freed using `EssFree()`.

**Return Value**

If this function is unable to get the process state, an error is returned. If the process terminates because of an error, then its error code is returned. Otherwise, this function returns ESS_STS_NOERR, and the current process state is given in the state structure `pProcState`. Values for `pProcState`:

- ESS_STATE_DONE—0 = Done
- ESS_STATE_INPROGRESS—1 = In progress
- ESS_STATE_FINALSTAGE—5 = In final stage; cannot be canceled

**Access**

This function requires no special privilege.

**Example**

```c
ESS_FUNC_M
ESS_RunCalc (ESS_HCTX_T hCtx)
{
    ESS_FUNC_M      sts = ESS_STS_NOERR;
    ESS_SHORT_T    isResponse;
    ESS_HCTX_T   hSrcCtx;
    ESS_BOOL_T   isObject = ESS_FALSE;
    ESS_STR_T    AppName;
    ESS_STR_T    DbName;
    ESS_STR_T    FileName;
    ESS_PROCSTATE_T pState;

    hSrcCtx  = hCtx;
    AppName  = "Sample";
    DbName   = "Basic";
    FileName = "Test";

    sts = EssCalcFile (hCtx, hSrcCtx, AppName,
                        DbName, FileName, ESS_TRUE);
    if (!sts)
    {
        sts = EssGetProcessState (hCtx, &pState);
        while(!sts && (pState.State !=
                      ESS_STATE_DONE))
```

408  C Main API Functions
sts = EssGetProcessState (hCtx, &pState);
}
return(sts);
}

See Also
- EssBeginCalc
- EssCalc
- EssCancelProcess
- EssImport

EssGetServerLocaleString

Gets the server locale description; for example, English_UnitedStates.US-ASCII@Default.

Syntax

```c
ESS_FUNC_M EssGetServerLocaleString (hCtx, localeString);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx;</td>
<td>ESS_HCTX_T</td>
<td>Context handle</td>
</tr>
<tr>
<td>localeString;</td>
<td>ESS_PSTR_T</td>
<td>Address of pointer to receive allocated string of server locale description.</td>
</tr>
</tbody>
</table>

Notes

The memory allocated for localeString should be freed using EssFree().

Return Value

If successful, returns the name of the server locale description in localeString.

Access

This function requires no special privileges.

Example

```c
ESS_FUNC_M
ESS_GetServerLocaleString (ESS_HCTX_T hCtx)
{
    ESS_FUNC_M     sts = ESS_STS_NOERR;
    ESS_STR_T localeStr= NULL;
    sts = EssGetServerLocaleString(hCtx, &localeStr);
    if (localeStr)
    {
        printf("server locale: %s\r\n",localeStr);
        EssFree(hInst,localeStr);
    }
    return sts;
}
```
EssGetServerMode

Returns a value indicating whether the Essbase Server is in Unicode mode or non-Unicode mode.

Syntax

```c
ESS_FUNC_M EssGetServerMode(hCtx, *bUnicode);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle (logged in)</td>
</tr>
<tr>
<td>bUnicode</td>
<td>ESS_BOOL_T</td>
<td>The returned value, <code>bUnicode</code>, where <code>bUnicode</code> can be:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESS_TRUE—Essbase Server is in Unicode mode. Essbase Server allows creation of Unicode mode applications or migration of non-Unicode mode applications to Unicode mode.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESS_FALSE—Essbase Server is in non-Unicode mode. Essbase Server does not allow creation of Unicode mode applications or migration of non-Unicode mode applications to Unicode mode.</td>
</tr>
</tbody>
</table>

Return Value

None.

Access

This function does not require the caller to have a special privilege.

See Also

• EssSetServerMode

EssGetSpoolFile

Returns a specific trigger log file for a database.

Syntax

```c
ESS_FUNC_M EssGetSpoolFile(hCtx, AppName, DbName, SplName, LocalName);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>AppName</td>
<td>ESS_STR_T</td>
<td>Application name.</td>
</tr>
<tr>
<td>DbName</td>
<td>ESS_STR_T</td>
<td>Database name.</td>
</tr>
<tr>
<td>SplName</td>
<td>ESS_STR_T</td>
<td>The name of a specific spool file to return.</td>
</tr>
<tr>
<td>LocalName</td>
<td>ESS_STR_T</td>
<td>The new name of the spool file on the server.</td>
</tr>
</tbody>
</table>

Return Value

If successful, returns a specific trigger spool file for a database.
Access

This function requires the caller to have database designer privilege (ESS_PRIV_DBDESIGN) for the specified database.

See Also

- EssDisplayTriggers
- EssListSpoolFiles
- EssDeleteAllSp1Files
- EssDeleteSplFile
- EssMdxTrig

EssGetSrvOutlineInfo

Gets outline information stored on the Essbase Server. There is no requirement to open the outline in query mode before using this function.

Syntax

```
ESS_FUNC_M EssGetSrvOutlineInfo (hCtx, AppName, DbName, pSvrOutlineInfo);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>AppName</td>
<td>ESS_STR_T</td>
<td>Application name.</td>
</tr>
<tr>
<td>DbName</td>
<td>ESS_STR_T</td>
<td>Database name.</td>
</tr>
<tr>
<td>pSvrOutlineInfo</td>
<td>“ESS_SVROTLINFO_T” on page 710</td>
<td>Pointer to structure containing outline information stored on the Essbase Server.</td>
</tr>
</tbody>
</table>

Return Value

Returns 0 if successful; otherwise, returns an error.

Example

```
ESS_FUNC_M ESS_GetSrvOutlineInfo()
{
    ESS_STS_T     sts = 0;
    ESS_INT_T     i;
    ESS_OBJDEF_T  Object;
    ESS_APPNAME_T szAppName;
    ESS_DBNAME_T  szDbName;
    ESS_OBJNAME_T szFileName;
    ESS_SVROTLINFO_T SvrOutlineInfo;

    memset(&Object, '\0', sizeof(Object));
    Object.hCtx =   hCtx;
    Object.ObjType =  ESS_OBJTYPE_OUTLINE;
    strcpy(szAppName, "Sample");
}```
strcpy(szDbName, "Basic");
strcpy(szFileName, "Basic");
Object.AppName = szAppName;
Object.DbName = szDbName;
Object.FileName = szFileName;

sts = EssGetSrvOutlineInfo (hCtx, szAppName, szDbName, &SvrOutlineInfo);

if (!sts)
{
    printf("\nCase sensitivity is set to: %d", (SvrOutlineInfo).fCaseSensitive);
    printf("\nOutline type is set to: %d", (SvrOutlineInfo).usOutlineType);
    printf("\nOutline allows duplicate names is set to: %d",
          (SvrOutlineInfo).fNonUniqueName);
    printf("\nNumber of alias tables is: %d", (SvrOutlineInfo).usNumAliasTables);
    printf("\nNames of the alias tables are: ");

    for (i = 0; i < (SvrOutlineInfo).usNumAliasTables; ++i)
        printf("\n   %s", (SvrOutlineInfo).pAliasTables[i]);
}

return sts;
}

The output of the above example is:

Case sensitivity is set to: 0
Outline type is set to: 0
Outline allows duplicate names is set to: 1
Number of alias tables is: 2
Names of the alias tables are:
   Default
   Long Names

**EssGetStatBufSize**

Returns a pointer to the size of the buffer needed for the performance statistics tables retrieved by **EssDumpPerfStats()**.

**Syntax**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx;</td>
<td>ESS_HCTX_T</td>
<td>API context handle</td>
</tr>
<tr>
<td>pBufSize;</td>
<td>ESS_PULONG_T</td>
<td>Pointer to the size of the buffer needed for the performance statistics tables</td>
</tr>
</tbody>
</table>

**Notes**

- Before you call **EssDumpPerfStats()**, call **EssGetStatBufSize()** to ascertain how much memory to allocate for the performance statistics tables at the address pointed to by `pStatBuf`. 
The buffer size pointed to by `pBufSize` is 0 if performance statistics have never been enabled; that is, if `persistence` in `EssResetPerfStats()` has never been set to 4.

**Return Value**

- If successful,
  - `EssGetStatBufSize()` returns 0.
  - `pBufSize` contains a pointer to the size of the buffer needed for the character array that will hold the performance statistics tables retrieved by `EssDumpPerfStats()`.
- For more information on performance statistics tables, see the ESSCMD commands `GETPERFSTATS` and `RESETPERFSTATS` in the *Oracle Essbase Technical Reference*.

**Access**

The caller of this function must have supervisor access.

**Example**

For a code example that calls `EssGetStatBufSize()`, see the example in `EssDumpPerfStats()`.

**See Also**

- `EssDumpPerfStats`
- `EssResetPerfStats`

### EssGetString

Gets a string of data from the active database. This function should be called after `EssReport()` or `EssEndReport()` if data is returned.

**Syntax**

```c
ESS_FUNC_M EssGetString (hCtx, pString);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>pString</td>
<td>ESS_PSTR_T</td>
<td>Address of pointer to receive allocated returned data string.</td>
</tr>
</tbody>
</table>

**Notes**

- Calling this function other than after successfully executing a report will generate an error.
- The returned string will be less than 64 KB long.
- Always include the carriage return/line feed with this command or errors will result.
- You must call `EssGetString()` until it returns a NULL string.
- The memory allocated for `pString` should be freed using `EssFree()`.
Return Value

An allocated pointer to the data string is returned in `pString`. This pointer will be NULL if there is no more data to be returned.

Access

This function requires no special privileges.

Example

```c
ESS_FUNC_M
ESS_Report (ESS_HCTX_T hCtx,
             ESS_HINST_T hInst
)
{
    ESS_FUNC_M     sts     = ESS_STS_NOERR;
    ESS_STR_T     rString = NULL;
    sts = EssBeginReport (hCtx, ESS_TRUE, ESS_FALSE);
    if (!sts)
        sts = EssSendString (hCtx, "<Desc Year !");
    if (!sts)
        sts = EssEndReport (hCtx);
    /**************
    * Get report *
    ***************/

    if (!sts)
        sts = EssGetString (hCtx, &rString);
    while (!sts && (rString != NULL))
    {
        printf ("%s", rString);
        EssFree (hInst, rString);
        sts = EssGetString (hCtx, &rString);
    }
    printf ("\r\n");
    return(sts);
}
```

See Also

- EssEndReport
- EssReport
- EssQueryDatabaseMembers

**EssGetUser**

Gets a user information structure, which contains security information for the user.

Syntax

```c
ESS_FUNC_M EssGetUser (hCtx, UserName, ppUserInfo);
```
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>UserName</td>
<td>ESS_STR_T</td>
<td>User name.</td>
</tr>
<tr>
<td>ppUserInfo</td>
<td>“ESS_USERINFO_T, ESS_GROUPINFO_T”</td>
<td>Address of pointer to receive allocated user info structure.</td>
</tr>
</tbody>
</table>

**Notes**

The memory allocated for `ppUserInfo` should be freed using `EssFree()`.

**Return Value**

If successful, returns the user information structure in `ppUserInfo`.

**Access**

This function requires the caller to have Create/Delete User privilege (ESS_PRIV_USERCREATE) for the logged in server, unless they are getting their own user information.

**Example**

```c
ESS_FUNC_M
ESS_GetUserInfo (ESS_HCTX_T  hCtx,
                 ESS_HINST_T hInst
               )
{
    ESS_FUNC_M      sts  = ESS_STS_NOERR;
    ESS_PUSERINFO_T User = NULL;

    sts = EssGetUser (hCtx, "Jim Smith", &User);
    if (!sts)
    {
        if (User)
        {
            EssFree (hInst, User);
        }
        return (sts);
    }
}
```

**See Also**

- `EssGetUserInfoEx`
- `EssGetApplicationAccess`
- `EssListUsers`
- `EssSetUser`

### EssGetUserInfoEx

Gets a user information structure, which contains security information for the user.
### Syntax

```c
ESS_FUNC_M EssGetUserEx (hCtx, UserName, ppUserInfo);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>UserName</td>
<td>ESS_STR_T</td>
<td>User name.</td>
</tr>
<tr>
<td>ppUserInfoEx</td>
<td>“ESS_USERINFOEX_T” on page 195</td>
<td>Address of pointer to receive info structure of externally authenticated user.</td>
</tr>
</tbody>
</table>

### Notes

The memory allocated for `ppUserInfo` should be freed using `EssFree()`.

### Return Value

If successful, returns the user information structure in `ppUserInfo`.

### Access

This function requires the caller to have Create/Delete User privilege (`ESS_PRIV_USERCREATE`) for the logged in server, unless they are getting their own user information.

### Example

```c
ESS_FUNC_M Ess_GetUserInfo (ESS_HCTX_T hCtx,  
                            ESS_HINST_T hInst  
)  
{  
    ESS_FUNC_M sts = ESS_STS_NOERR;  
    ESS_PUSERINFO_T User = NULL;  
    
    sts = EssGetUserEx (hCtx, "Jim Smith", &User);  
    if (!sts)  
    {  
        if (User)  
            EssFree (hInst, User);  
    
    return (sts);  

### See Also

- `EssGetApplicationAccess`
- `EssCreateExtUser`
- `EssListUsers`
- `EssSetUser`
- `EssSetUserEx`
- “ESS_USERINFOEX_T” on page 195
**EssGetUserInfoEx**

Gets a user information structure, which contains security information for the user. Similar to
**EssGetUser**, but can accept a user directory specification or unique identity attribute for
**UserID**.

**Syntax**

ESS_FUNC_M EssGetUserInfoEx (hCtx, UserId, bIsIdentity, ppUserInfo);

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle (input).</td>
</tr>
<tr>
<td>UserName</td>
<td>ESS_STR_T</td>
<td>User name (input). Can be specified as username@provider or as a unique identity attribute.</td>
</tr>
<tr>
<td>bIsIdentity</td>
<td>ESS_BOOL_T</td>
<td>Input. Indicates if UserID is a name or an identity. If TRUE, UserID is an identity.</td>
</tr>
<tr>
<td>ppUserInfo</td>
<td>ESS_PUSERINFOID_T</td>
<td>Address of pointer to receive allocated user info structure (output). The user list structure can include user directories and unique identity attributes.</td>
</tr>
</tbody>
</table>

**Notes**

The memory allocated for ppUserInfo should be freed using EssFree().

**Return Value**

If successful, returns the user information structure in ppUserInfo

**Access**

This function requires the caller to have Create/Delete User privilege (ESS_PRIV_USERCREATE) for the logged in server.

**Example**

```c
void DisplayUserInfoID(ESS_PUSERINFOID_T userInfo)
{
    ESS_STS_T sts = ESS_STS_NOERR;
    ESS_BOOL_T isDefined = ESS_TRUE;

    printf("\tUser Name: %s\n", userInfo->Name);
    printf("\tProvider Name: %s\n", userInfo->ProviderName);
    printf("\tConnparam: %s\n", userInfo->connparam);
    printf("\tDescription: %s\n", userInfo->Description);
    printf("\tEMail Identification: %s\n", userInfo->EMailID);

    if (userInfo->LockedOut)
        printf("\tLocked out: Yes\n");
    else
        printf("\tLocked out: No\n");

    if (userInfo->PwdChgNow)
        printf("\tChange the password now: Yes\n");
    else
        printf("\tChange the password now: No\n");
```
printf("%tConnected Application: %s\n", userInfo->AppName);
printf("%tConnected Database: %s\n", userInfo->DbName);

if (userInfo->Login)
    printf("%tLogged in: Yes\n");
else
    printf("%tLogged in: No\n");

switch(userInfo->Access)
{
    case ESS_ACCESS_ADMIN:
        printf("%tAccess: %d - ESS_ACCESS_ADMIN\n", userInfo->Access);
        break;
    case ESS_ACCESS_APPALL:
        printf("%tAccess: %d - ESS_ACCESS_APPALL\n", userInfo->Access);
        break;
    case ESS_ACCESS_DBALL:
        printf("%tAccess: %d - ESS_ACCESS_DBALL\n", userInfo->Access);
        break;
    case ESS_ACCESS_APPCREATE:
        printf("%tAccess: %d - ESS_ACCESS_APPCREATE\n", userInfo->Access);
        break;
    case ESS_ACCESS_APPMANAGE:
        printf("%tAccess: %d - ESS_ACCESS_APPMANAGE\n", userInfo->Access);
        break;
    case ESS_ACCESS_DBCREATE:
        printf("%tAccess: %d - ESS_ACCESS_DBCREATE\n", userInfo->Access);
        break;
    case ESS_ACCESS_DBMANAGE:
        printf("%tAccess: %d - ESS_ACCESS_DBMANAGE\n", userInfo->Access);
        break;
    case ESS_ACCESS_CALC:
        printf("%tAccess: %d - ESS_ACCESS_CALC\n", userInfo->Access);
        break;
    case ESS_ACCESS_WRITE:
        printf("%tAccess: %d - ESS_ACCESS_WRITE\n", userInfo->Access);
        break;
    case ESS_ACCESS_READ:
        printf("%tAccess: %d - ESS_ACCESS_READ\n", userInfo->Access);
        break;
    case ESS_PRIV_USERCREATE:
        printf("%tAccess: %d - ESS_PRIV_USERCREATE\n", userInfo->Access);
        break;
    case ESS_PRIV_APPCREATE:
        printf("%tAccess: %d - ESS_PRIV_APPCREATE\n", userInfo->Access);
        break;
    case ESS_PRIV_APPMANAGE:
        printf("%tAccess: %d - ESS_PRIV_APPMANAGE\n", userInfo->Access);
        break;
    case ESS_PRIV_APPLOAD:
        printf("%tAccess: %d - ESS_PRIV_APPLOAD\n", userInfo->Access);
        break;
    case ESS_PRIV_DBCREATE:
        printf("%tAccess: %d - ESS_PRIV_DBCREATE\n", userInfo->Access);
        break;
    case ESS_PRIV_DBMANAGE:
        printf("%tAccess: %d - ESS_PRIV_DBMANAGE\n", userInfo->Access);
        break;
}

C Main API Functions
printf("\tAccess: %d - ESS_PRIV_DBMANAGE\n", userInfo->Access);
break;
case ESS_PRIV_DLOAD:
    printf("\tAccess: %d - ESS_PRIV_DLOAD\n", userInfo->Access);
    break;
case ESS_PRIV_CALC:
    printf("\tAccess: %d - ESS_ACCESS_DBALL\n", userInfo->Access);
    break;
case ESS_PRIV_WRITE:
    printf("\tAccess: %d - ESS_PRIV_WRITE\n", userInfo->Access);
    break;
case ESS_PRIV_READ:
    printf("\tAccess: %d - ESS_PRIV_READ\n", userInfo->Access);
    break;
case ESS_PRIV_NONE:
    printf("\tAccess: %d - ESS_PRIV_NONE\n", userInfo->Access);
    break;
default:
    printf("\tAccess: Unknown\n");
}

switch(userInfo->MaxAccess)
{
    case ESS_ACCESS_ADMIN:
        printf("\tMax Access: %d - ESS_ACCESS_ADMIN\n", userInfo->MaxAccess);
        break;
case ESS_ACCESS_APPALL:
        printf("\tMax Access: %d - ESS_ACCESS_APPALL\n", userInfo->MaxAccess);
        break;
case ESS_ACCESS_DBALL:
        printf("\tMax Access: %d - ESS_ACCESS_DBALL\n", userInfo->MaxAccess);
        break;
case ESS_ACCESS_APPCREATE:
        printf("\tMax Access: %d - ESS_ACCESS_APPCREATE\n", userInfo->MaxAccess);
        break;
case ESS_ACCESS_APPMANAGE:
        printf("\tMax Access: %d - ESS_ACCESS_APPMANAGE\n", userInfo->MaxAccess);
        break;
case ESS_ACCESS_DBCREATE:
        printf("\tMax Access: %d - ESS_ACCESS_DBCREATE\n", userInfo->MaxAccess);
        break;
case ESS_ACCESS_DBMANAGE:
        printf("\tMax Access: %d - ESS_ACCESS_DBMANAGE\n", userInfo->MaxAccess);
        break;
case ESS_ACCESS_CALC:
        printf("\tMax Access: %d - ESS_ACCESS_CALC\n", userInfo->MaxAccess);
        break;
case ESS_ACCESS_WRITE:
        printf("\tMax Access: %d - ESS_ACCESS_WRITE\n", userInfo->MaxAccess);
        break;
case ESS_ACCESS_READ:
        printf("\tMax Access: %d - ESS_ACCESS_READ\n", userInfo->MaxAccess);
        break;
case ESS_PRIV_USERCREATE:
        printf("\tMax Access: %d - ESS_PRIV_USERCREATE\n", userInfo->MaxAccess);
        break;
case ESS_PRIV_APPCREATE:
printf("\tMax Access: %d - ESS_PRIV_APPCREATE\n", userInfo->MaxAccess);
break;
case ESS_PRIV_APPMANAGE:
    printf("\tMax Access: %d - ESS_PRIV_APPMANAGE\n", userInfo->MaxAccess);
    break;
case ESS_PRIV_APPLOAD:
    printf("\tMax Access: %d - ESS_PRIV_APPLOAD\n", userInfo->MaxAccess);
    break;
case ESS_PRIV_DBCREATE:
    printf("\tMax Access: %d - ESS_PRIV_DBCREATE\n", userInfo->MaxAccess);
    break;
case ESS_PRIV_DBMANAGE:
    printf("\tMax Access: %d - ESS_PRIV_DBMANAGE\n", userInfo->MaxAccess);
    break;
case ESS_PRIV_DDBLOAD:
    printf("\tMax Access: %d - ESS_PRIV_DDBLOAD\n", userInfo->MaxAccess);
    break;
case ESS_PRIV_CALC:
    printf("\tMax Access: %d - ESS_ACCESS_DBALL\n", userInfo->MaxAccess);
    break;
case ESS_PRIV_WRITE:
    printf("\tMax Access: %d - ESS_PRIV_WRITE\n", userInfo->MaxAccess);
    break;
case ESS_PRIV_READ:
    printf("\tMax Access: %d - ESS_PRIV_READ\n", userInfo->MaxAccess);
    break;
case ESS_PRIV_NONE:
    printf("\tMax Access: %d - ESS_PRIV_NONE\n", userInfo->MaxAccess);
    break;
default:
    printf("\tMax Access: Unknown\n");
}

printf("\tPassword Expiration in Dates: %d\n",userInfo->Expiration);
//EssSdCTime(NULL, userInfo->LastLogin, sizeof(time_string), time_string);
//printf("\tLast Successful Login:            %s\n", time_string);
printf("\tFailed Login Attempts Since Then: %d\n", userInfo->FailCount);
printf("\tLogin ID: %d\n", userInfo->LoginId);
printf("\n");

ESS_FUNC_M ESS_GetUserInfoEx (ESS_HCTX_T hCtx)
{
    ESS_STS_T sts  = ESS_STS_NOERR;
    ESS_STR_T userId;
    ESS_BOOL_T bUsingIdentity;
    ESS_PUSERINFOID_T userInfo = NULL;

    bUsingIdentity = ESS_TRUE;
    sts = EssGetUserInfoEx (hCtx, userId, bUsingIdentity, &userInfo);
    printf("EssGetUserInfoEx sts: %ld\n", sts);
    if (userInfo)
    {
        DisplayUserInfoID(userInfo);
    }
}

C Main API Functions

420
return (sts);
}

See Also
- EssListUserInfoEx

**EssGetUserType**

Enables you to find the application access type for a user.

**Syntax**

```
ESS_FUNC_M EssGetUserType (hCtx, UserName, UserType)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>Essbase API context handle.</td>
</tr>
<tr>
<td>UserName</td>
<td>ESS_STR_T</td>
<td>Name of user.</td>
</tr>
<tr>
<td>UserType</td>
<td>ESS_PUSER_TYPE_T</td>
<td>Application access type defined for the UserName specified.</td>
</tr>
</tbody>
</table>

**Return Value**

Returns the status of the API.

**Example**

```
ESS_FUNC_M
ESS_GetUserType (ESS_HCTX_T hCtx)
{
    ESS_FUNC_M sts = ESS_STS_NOERR;
    ESS_STR_T UserName = "jsmith";
    ESS_PUSER_TYPE_T UserType;

    sts = EssGetUserType (hCtx, UserName, UserType);
    printf("user type for the user %s is %d\n", UserName, *UserType);

    return (sts);
}
```

See Also
- EssSetUserType

**EssGetVariable**

Retrieves the value of a substitution variable.

**Syntax**

```
ESS_FUNC_M EssGetVariable (hCtx, pVariable);
```
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>Context handle to the API.</td>
</tr>
<tr>
<td>pVariable</td>
<td>&quot;ESS_VARIABLE_T&quot; on page 197</td>
<td>The pointer to the structure containing the description of the specified substitution variable.</td>
</tr>
</tbody>
</table>

**Return Value**

If successful, `EssGetVariable()` returns the value of the substitution variable in the `VarValue` field of structure `ESS_VARIABLE_T`.

**Example**

```c
/*
** EssGetVariable() gets the substitution variable value using
** the API function EssGetVariable.
*/

ESS_FUNC_M
ESS_GetVariable (ESS_HCTX_T hCtx)
{
    ESS_FUNC_M      sts = ESS_STS_NOERR;
    ESS_VARIABLE_T Variable;
    printf("\n ***************************************");
    printf("\n **** An example of using EssGetVariable");
    printf("\n ***************************************");
    /******************************/
    / * Get the Value of QuarterName */
    /******************************/
    strcpy(Variable.VarName, "QuarterName");
    strcpy(Variable.Server,  "Local");
    strcpy(Variable.AppName, "Sample");
    strcpy(Variable.DbName,  "Basic");
    sts = EssGetVariable(hCtx, &Variable);
    if (sts == ESS_STS_NOERR)
    {
        printf("\n------- Substitution Variable 'QuarterName' Information \n");
        printf("Variable name    : %s\n",   Variable.VarName);
        printf("Server name      : %s\n",   Variable.Server);
        printf("Application name : %s\n",   Variable.AppName);
        printf("Database name    : %s\n",   Variable.DbName);
        printf("Variable value   : %s\n",   Variable.VarValue);
    }
    /******************************/
    / * Get the Value of MarketName at the level of the Server/App */
    /******************************/
    if (sts == ESS_STS_NOERR)
    {
        strcpy(Variable.VarName, "MarketName");
        strcpy(Variable.Server,  "Local");
        strcpy(Variable.AppName, "Sample");
        strcpy(Variable.DbName,  "");
        sts = EssGetVariable(hCtx, &Variable);
        if (sts == ESS_STS_NOERR)
        {
            printf("\n------- Substitution Variable 'MarketName' Information \n");
        }
```
printf("Variable name    : %s\n",   Variable.VarName);
printf("Server name      : %s\n",   Variable.Server);
printf("Application name : %s\n",   Variable.AppName);
printf("Database name    : %s\n",   Variable.DbName);
printf("Variable value   : %s\n\n", Variable.VarValue);
}
}

/**********************************************************/
/* Get the Value of MarketName at the level of the Server */
/**********************************************************/
if (sts == ESS_STS_NOERR)
{
    strcpy(Variable.VarName, "MarketName");
    strcpy(Variable.Server, "Local");
    strcpy(Variable.AppName, "");
    strcpy(Variable.DbName, "");
    sts = EssGetVariable(hCtx, &Variable);
    if (sts == ESS_STS_NOERR)
    {
        printf("\n------- Substitution Variable 'MarketName' Information \n");
        printf("Variable name    : %s\n",   Variable.VarName);
        printf("Server name      : %s\n",   Variable.Server);
        printf("Application name : %s\n",   Variable.AppName);
        printf("Database name    : %s\n",   Variable.DbName);
        printf("Variable value   : %s\n\n", Variable.VarValue);
    }
}
if (sts == ESS_STS_NOERR)
    printf("\n --> No Errors in EssGetVariable\n\n\n\n");
else
    printf("\n --> Error in EssGetVariable number: %d\n\n\n\n", sts);

return (sts);
} /* End ESS_GetVariable */

Output

***************************************
**** An example of using EssGetVariable
***************************************
------- Substitution Variable 'QuarterName' Information
Variable name    : QuarterName
Server name      : Local
Application name : Sample
Database name    : Basic
Variable value   : Qtr2

------- Substitution Variable 'MarketName' Information
Variable name    : MarketName
Server name      : Local
Application name : Sample
Database name    :
Variable value   : East

------- Substitution Variable 'MarketName' Information
Variable name    : MarketName
Server name      : Local
See Also

- “ESS_VARIABLE_T” on page 197
- EssCreateVariable
- EssDeleteVariable
- EssListVariables

### EssGetVersion

Gets the full version number of the connected Essbase Server, in the form 
`Release.Version.Revision`, e.g. 11.1.2.

#### Syntax

```c
ESS_FUNC_M EssGetVersion (hCtx, pRelease, pVersion, pRevision);
```

#### Parameter Data Type Description

- **hCtx** ESS_HCTX_T API context handle.
- **pRelease** ESS_PUSHORT_T Address of variable to receive release number.
- **pVersion** ESS_PUSHORT_T Address of variable to receive version number.
- **pRevision** ESS_PUSHORT_T Address of variable to receive revision number.

#### Notes

You can call this function after connecting to a server, to ensure that the Essbase Server version supports all the features used by your program.

#### Return Value

If successful, returns the full incremental Essbase version number in `pRelease`, `pVersion` and `pRevision`.

#### Access

This function requires no special privileges.

#### Example

```c
ESS_FUNC_M
ESS_GetVersion (ESS_HCTX_T hCtx)
{
    ESS_FUNC_M     sts = ESS_STS_NOERR;
    ESS_USHORT_T   Release;
    ESS_USHORT_T   Version;
    ESS_USHORT_T   Revision;

    sts = EssGetVersion (hCtx, &Release, &Version,
```
if (!sts)
{
    printf ("\r\nEssbase Application Server - Version %d.%d.%d\r\n", Release, Version, Revision);
}
return (sts);
}

See Also
● EssInit

**EssImport**

Allows importing data from different sources to the Essbase Server.

**Syntax**

ESS_FUNC_M EssImport (hCtx, pRules, pData, ppMbrErr, pMbrUser, abortOnError);

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>pRules</td>
<td>“ESS_OBJDEF_T” on page 151</td>
<td>Pointer to the rules file object definition structure.</td>
</tr>
<tr>
<td>pData</td>
<td>“ESS_OBJDEF_T” on page 151</td>
<td>Pointer to the data file object definition structure.</td>
</tr>
<tr>
<td>ppMbrErr</td>
<td>“ESS_MBRERR_T” on page 148</td>
<td>Pointer to linked list of errors contained in ESS_MBRERR_T. Possible errors are:</td>
</tr>
<tr>
<td>pMbrUser</td>
<td>“ESS_MBRUSER_T” on page 148</td>
<td>Pointer to the SQL user structure (if data source is a SQL database). A NULL SQL user structure indicates a non SQL data source.</td>
</tr>
<tr>
<td>abortOnError</td>
<td>ESS_USHORT_T</td>
<td>If TRUE, import stops on the first error. Otherwise, it continues.</td>
</tr>
</tbody>
</table>

**Notes**

● For a non SQL source, if the *AppName* and *DbName* fields in ESS_OBJDEF_T structures for *pRules* and *pData* are NULL, *hCtx* must be a local context handle, and the ESS_OBJDEF_T *FileName* field must contain the fully qualified path to the file.

● If a local object is used, EssCreateLocalContext() must be called first.

● The memory allocated for ppMbrErr must be freed using EssFreeMbrErr().
Return Value

Returns zero if successful. Otherwise, returns an error code.

Access

This function requires the caller to have database designer privilege for the specified database (ESS_PRIV_DBDESIGN).

Example

```c
ESS_FUNC_M
ESS_Import(ESS_HCTX_T hCtx)
{
    ESS_FUNC_M     sts = ESS_STS_NOERR;
    ESS_SHORT_T    isAbortOnError;
    ESS_OBJDEF_T   Rules;
    ESS_OBJDEF_T   Data;
    ESS_MBRUSER_T  User;
    ESS_PMBERR_T   pMbrErr = NULL;

    Data.hCtx   = hCtx;
    Data.AppName = "Olap";
    Data.DbName  = "Demo";
    Data.ObjType = ESS_OBJTYPE_TEXT;
    Data.FileName = "Actuals";

    Rules.hCtx  = hCtx;
    Rules.AppName = "Olap";
    Rules.DbName = "Demo";
    Rules.ObjType = ESS_OBJTYPE_RULES;
    Rules.FileName = "Actmap";
    /**********************/
    /* Running conditions */
    /**********************/
    isAbortOnError = ESS_TRUE;

    sts = EssImport (hCtx, &Rules, &Data, &pMbrErr,
                       NULL, isAbortOnError);
    if(pMbrErr)
        EssFreeMbrErr(hCtx, pMbrErr);
}
```

/* When a SQL data source is defined in the rules file, define */
/* the variables in the ESS_OBJDEF_T Data structure as follows: */
/* Data.hCtx   = hCtx;   */
/* Data.AppName = NULL;  */
/* Data.DbName  = NULL;  */
/* Data.ObjType = ESS_OBJTYPE_NONE; */
/* Data.FileName = NULL; */
/* */
/* Also, provide strings for the variables in the ESS_MBRUSER_T */
/* User structure; for example: */
/* User.User     = "Dbusernm"; */
/* User.Password = "Dbpasswd"; */
/* */
/* Use a blank string for User and Password, if the SQL source */
/* does not require user and password information; for example: */
/* User.User     = "";                                          */
/* User.Password = "";                                          */
/*                                                            */
/* Also, define sts as follows:                                */
/*    sts = EssImport (hCtx, &Rules, &Data, &pMbrErr,          */
/*           &User, isAbortOnError);                           */
/*                                                            */
/**-----------------------------------------------------------------*/
See Also

- EssExport
- EssBuildDimension
- EssFreeMbrErr

EssImportASO

Allows importing data from different sources to an Essbase aggregate storage database.

Syntax

ESS_FUNC (hCtx, pRules, pData, ppMbrErr, pUser, usabortOnError, ulBufferId);

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>pRules</td>
<td>“ESS_OBJDEF_T” on page 151</td>
<td>Pointer to the rules file object definition structure.</td>
</tr>
<tr>
<td>pData</td>
<td>“ESS_OBJDEF_T” on page 151</td>
<td>Pointer to the data file object definition structure.</td>
</tr>
<tr>
<td>ppMbrErr</td>
<td>“ESS_MBRERR_T” on page 148</td>
<td>Pointer to linked list of errors contained in ESS_MBRERR_T. Possible errors are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_MBRERR_BADDIM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_MBRERR_BADGEN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_MBRERR_UNKNOWN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_MBRERR_BADACCESS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_MBRERR_BADSYNTAX</td>
</tr>
<tr>
<td>pUser</td>
<td>“ESS_MBRUSER_T” on page 148</td>
<td>Pointer to the SQL user structure (if data source is a SQL database). A NULL SQL user structure indicates a non SQL data source.</td>
</tr>
<tr>
<td>usabortOnError</td>
<td>ESS_USHORT_T</td>
<td>If TRUE, import stops on the first error. Otherwise, it continues.</td>
</tr>
<tr>
<td>ulBufferId</td>
<td>ESS_ULONG_T</td>
<td>ID of a data load buffer (a number between 1 and 999,999). To destroy a buffer before a data load is complete, you must use the same ulBufferId number that was used to initialize the buffer.</td>
</tr>
</tbody>
</table>
Notes

- For a non SQL source, if the `AppName` and `DbName` fields in ESS_OBJDEF_T structures for `pRules` and `pData` are NULL, `hCtx` must be a local context handle, and the ESS_OBJDEF_T `FileName` field must contain the fully qualified path to the file.
- If a local object is used, `EssCreateLocalContext()` must be called first.
- The memory allocated for `ppMbrErr` must be freed using `EssFreeMbrErr()`.

Return Value

Returns zero if successful; otherwise, returns an error code.

Access

This function requires the caller to have database designer privilege for the specified database (ESS_PRIV_DBDESIGN).

Example

```c
void TestImportASO(ESS_HCTX_T hCtx, ESS_STR_T AppName, ESS_STR_T DbName)
{
    ESS_STS_T       sts = ESS_STS_NOERR;
    ESS_SHORT_T     isAbortOnError;
    ESS_OBJDEF_T    Rules;
    ESS_OBJDEF_T    Data;
    ESS_PMBRERR_T   pMbrErr = NULL;
    ESS_PMBRUSER_T  pMbrUser = NULL;
    ESS_ULONG_T     ulBufferId;
    ESS_ULONG_T     ulDuplicateAggregationMethod;
    ESS_ULONG_T     ulOptionsFlags;
    ESS_ULONG_T     ulSize;
    ESS_ULONG_T     ulBufferCnt;
    ESS_ULONG_T     ulCommitType;
    ESS_ULONG_T     ulActionType;
    ESS_ULONG_T     ulOptions;
    ESS_ULONG_T ulBufferIdAry[1];

    ulDuplicateAggregationMethod = ESS_ASO_DATA_LOAD_BUFFER_DUPLECTES_ADD;
    ulOptionsFlags = ESS_ASO_DATA_LOAD_BUFFER_IGNORE_MISSING_VALUES;
    ulSize = 100;
    ulBufferId = 10;
    sts = EssLoadBufferInit(hCtx, AppName, DbName, ulBufferId,
                        ulDuplicateAggregationMethod, ulOptionsFlags, ulSize);
    printf("EssLoadBufferInit sts: %ld\n", sts);

    /* Server object */
    Rules.hCtx     = hCtx;
    Rules.AppName  = AppName;
    Rules.DbName   = DbName;
    Rules.ObjType  = ESS_OBJTYPE_RULES;
    Rules.FileName = "Dataload";
    Data.hCtx      = hCtx;
    Data.AppName   = AppName;
    Data.DbName    = DbName;
    Data.ObjType   = ESS_OBJTYPE_TEXT;
    Data.FileName  = "Dataload";
    isAbortOnError = ESS_TRUE;
}
```
sts = EssImportASO (hCtx, &Rules, &Data, &pMbrErr, pMbrUser,
    isAbortOnError, ulBufferId);
printf("EssImportASO sts: %ld\n",sts);
if(pMbrErr)
    EssFreeMbrErr(hCtx, pMbrErr);

/* Commit and delete the buffer */
ulBufferCnt = 1;
ulBufferIdAry[0] = ulBufferId;
ulCommitType = ESS_ASO_DATA_LOAD_BUFFER_STORE_DATA;
ulActionType = ESS_ASO_DATA_LOAD_BUFFER_COMMIT;
printf("\nCommit data to the main slice:\n");
ulOptions = ESS_ASO_DATA_LOAD_INCR_TO_MAIN_SLICE;
sts = EssLoadBufferTerm(hCtx, AppName, DbName, ulBufferCnt,
    ulBufferIdAry, ulCommitType, ulActionType, ulOptions);
printf("EssLoadBufferTerm sts: %ld\n",sts);
}

See Also

- EssLoadBufferInit
- EssBeginDataloadASO
- EssSendString
- EssEndDataload
- EssLoadBufferTerm
- EssUpdateFileASO
- EssUpdateFileUTF8ASO
- EssListExistingLoadBuffers
- EssMergeDatabaseData

**EssIncrementalBuildDim**

Builds dimensions with the specified rules file and data source. Can be called multiple times within the incremental dimension build protocol.

EssBeginIncrementalBuildDim() must be called before EssIncrementalBuildDim() gets called.

Syntax

ESS_FUNC_M EssIncrementalBuildDim(hCtx, RulesObj, DataObj, MbrUser, ErrorName,
    bOverwrite, usBuildOption, szTmpOtlFile)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>Essbase API context handle.</td>
</tr>
<tr>
<td>RulesObj</td>
<td>ESS_POBJDEF_T</td>
<td>Pointer to rules file object definition structure.</td>
</tr>
<tr>
<td>DataObj</td>
<td>ESS_POBJDEF_T</td>
<td>Pointer to data file object definition structure.</td>
</tr>
<tr>
<td>MbrUser</td>
<td>ESS_PMBRUSER_T</td>
<td>SQL user structure (if data source is SQL database). A NULL SQL user structure indicates a non SQL data source.</td>
</tr>
</tbody>
</table>

C Main API Function Reference 429
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ErrorName</td>
<td>ESS_STR_T</td>
<td>Name of error output file on client.</td>
</tr>
<tr>
<td>bOverwrite</td>
<td>ESS_BOOL_T</td>
<td>Indicates overwrite or append error message to the error file.</td>
</tr>
<tr>
<td>usBuildOption</td>
<td>ESS_USHORT_T</td>
<td>Valid values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_INCDIMBUILD_BUILD Build members only.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_INCDIMBUILD_VERIFY Build members and verify the outline.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_INCDIMBUILD_SAVEOTL Build members and save the outline to a temp outline file.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_INCDIMBUILD_ALL Build members, verify the outline, and restructure.</td>
</tr>
<tr>
<td>szTmpOtlFile</td>
<td>ESS_STR_T</td>
<td>The temp outline file name. Essbase creates a temporary outline file with extension &quot;otb&quot; if the resulting outline in this round of dimension build has outline verification errors.</td>
</tr>
</tbody>
</table>

**Return Value**

Returns zero if successful; error code if unsuccessful.

**Example**

See [EssBeginIncrementalBuildDim](#).

**See Also**

- [EssIncrementalBuildDim](#)
- [EssBeginIncrementalBuildDim](#)
- [EssBeginStreamBuildDim](#)
- [EssEndIncrementalBuildDim](#)
- [EssEndStreamBuildDim](#)

---

### EssInit

Initializes the API and message database.

**Syntax**

```c
ESS_FUNC_M EssInit (pInitStruct, phInstance);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pInitStruct</td>
<td>“ESS_INIT_T” on page 141</td>
<td>Pointer to API initialization structure</td>
</tr>
<tr>
<td>phInstance</td>
<td>ESS_PHINST_T</td>
<td>Pointer to Essbase API instance handle</td>
</tr>
</tbody>
</table>

---
Notes

- You must call this function before any other Essbase API functions.
- If any field in the initialization structure is NULL or zero (as appropriate), the API uses a default value for those parameters.

Return Value

The ESS_INIT_T structure passed to this function includes a number of initialization parameters, including the name of the message database, the maximum size of client buffer that can be allocated, pointers to the user-defined memory free allocation, error callback functions, the name and location of your help file, and version number.

EssInit() returns the phInstance instance handle that allows multiple applications to access the API independently (for DLLs only). The instance handle should be preserved and passed to EssLogin(), EssTerm() and the memory allocation functions.

Access

This function requires no special privileges.

Example

ESS_VOID_T ESS_Init()
{
    ESS_HINST_T     hInst;
    ESS_INIT_T InitStruct =        /* Define init */
        /* structure */
        { /* Initialize the API */
            ESS_API_VERSION,       /* Version of API */
            USER,      /* user-defined message context */
            0,         /* max handles */
            0L,        /* max buffer size */
            "C:\\Essbase", /* local path */
            /* The following parameters use defaults */
            NULL,      /* message db path */
            NULL,      /* allocation function pointer */
            NULL,      /* reallocation function pointer */
            NULL,      /* free function pointer */
            NULL,      /* error handling function pointer */
            NULL,      /* path name of user-defined */
                        /* Application help file */
                /* Reserved for internal use. */
                /* Set to NULL */
        };
    if ((sts = EssInit (&InitStruct, &hInst)) != ESS_STS_NOERR)
        /* error initializing API */
        exit ((ESS_USHORT_T) sts);
}

See Also

- EssLogin
- EssAutoLogin
- EssTerm
## EssKillRequest

Terminates specific Essbase user sessions or requests.

### Syntax

```c
ESS_FUNC_M EssKillRequest (hCtx, pRequestInfoStruct);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>pRequestInfoStruct</td>
<td>ESS_PREQUESTINFO_T</td>
<td>Pointer to the Request Information structure.</td>
</tr>
</tbody>
</table>

### Notes

`EssKillRequest()` uses the information in “ESS_REQUESTINFO_T” on page 179 regarding current sessions and requests terminate a specific user session. This function can also be used to terminate (without logging out the user) any active requests being made to an application, a database, or the system during a user session.

A session is the time in seconds between an Essbase user's login and logout.

A request is a query sent to Essbase by a user or by another process; for example, starting an application, or restructuring a database outline. Each session can process only one request at a time; therefore, sessions and requests have a one-to-one relationship.

This function terminates the sessions/requests specified by the UserName, AppName, and DbName specified in the ESS_REQUESTINFO_T structure. If those fields are null, this function terminates all sessions/requests initiated by this process (user). The application program is responsible for allocating and freeing the memory used by ESS_REQUESTINFO_T.

### Return Value

Returns zero if successful; error code if unsuccessful.

### Example

```c
#include
#include

ESS_FUNC_M ESS_ListRequest ()
{
    ESS_FUNC_M     sts     = ESS_STS_NOERR;
    ESS_STR_T     rString = NULL;
    ESS_HCTX_T    hCtx;
    ESS_USHORT_T  Items;
    ESS_PAPPDB_T   pAppsDbs = NULL;
    ESS_HINST_T    hInst ;
    ESS_ACCESS_T   Access;
    ESS_USHORT_T   numRequest;
    ESS_PREQUESTINFO_T requestInfo;

    ESS_INIT_T InitStruct =        /* Define init */
        /* structure */

    ESS_API_VERSION,     /* Version of API */
```
EssInit (&InitStruct, &hInst);

sts = EssLogin (hInst, "local", "admin", "password", &Items, &pAppsDbs, &hCtx);

sts = EssSetActive ( hCtx, "sample", "basic", &Access );

sts = EssListRequests( hCtx, NULL, NULL, NULL, &numRequest, &requestInfo);

printf ( "Total requests on the server %d\n", numRequest );

if ( !sts && requestInfo )
{
    ESS_USHORT_T index = 0;

    while ( index < numRequest )
    {
        printf ( "login ID = %ul\n", requestInfo[index].LoginId );
        printf ( "user name = %s\n", requestInfo[index].UserName );
        printf ( "login machine = %s\n", requestInfo[index].LoginSourceMachine );
        printf ( "AppName = %s\n", requestInfo[index].AppName );
        printf ( "DbName = %s\n", requestInfo[index].DbName );
        printf ( "DbRequestCode = %u\n", requestInfo[index].DbRequestCode );
        printf ( "RequestString = %s\n", requestInfo[index].RequestString );
        printf ( "TimeStarted = %ul\n", requestInfo[index].TimeStarted );
        printf ( "State = %d\n", requestInfo[index].State );
        printf ( "\n\n--------------------------------------\n\n", requestInfo[index].State );
        sts = EssKillRequest (hCtx, &requestInfo[index] );

        index++;
    }

    EssFree ( hInst, requestInfo );
}

EssLogout (hCtx);
EssTerm (hInst);
return(sts);
void main()
{
    ESS_ListRequest();
}

See Also

- `EssKillRequestEx`
- `EssListRequests`
- “ESS_REQUESTINFO_T” on page 179
- “ESS_REQ_STATE_T” on page 185

**EssKillRequestEx**

Terminates specific Essbase user sessions or requests. Similar to `EssKillRequest`, but the input structure can include user directories and unique identity attributes.

**Syntax**

```c
ESS_FUNC_M EssKillRequestEx (hCtx, pRequestInfoStruct);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle (input).</td>
</tr>
<tr>
<td>pRequestInfoStruct</td>
<td>ESS_REQUESTINFOEX_T</td>
<td>Pointer to the Request Information structure (input).</td>
</tr>
</tbody>
</table>

**Notes**

This function uses the information in `ESS_REQUESTINFOEX_T` regarding current sessions and requests to terminate a specific user session. This function can also be used to terminate (without logging out the user) any active requests being made to an application, a database, or the system during a user session.

A session is the time in seconds between an Essbase user’s login and logout.

A request is a query sent to Essbase by a user or by another process; for example, starting an application, or restructuring a database outline. Each session can process only one request at a time; therefore, sessions and requests have a one-to-one relationship.

This function terminates the sessions/requests specified by the UserName, AppName, and DbName specified in the `ESS_REQUESTINFOEX_T` structure. If those fields are null, this function terminates all sessions/requests initiated by this process (user). The application program is responsible for allocating and freeing the memory used by `ESS_REQUESTINFOEX_T`.

**Return Value**

Returns zero if successful; error code if unsuccessful.

**Example**

```c
void ListRequestsEx ()
{
```
ESS_STS_T sts = ESS_STS_NOERR;
ESS_STR_T userId;
ESS_BOOL_T bIsIdentity;
ESS_Ushort_T numRequest;
ESS_PREQUESTINFOEX_T requestInfo;
ESS_Ushort_T index = 0;

userId = "admin";
bIsIdentity = ESS_FALSE;
sts = EssListRequestsEx(hCtx, userId, bIsIdentity, "Sample", "Sample1", &numRequest, &requestInfo);
printf("\nEssListRequestsEx sts: %ld\n", sts);
printf ("Total requests on the server: %d\n", numRequest );
if ( !sts && requestInfo )
{
    while ( index < numRequest )
    {
        printf ( "login ID = %ul\n", requestInfo[index].LoginId );
        printf ( "user name = %s\n", requestInfo[index].UserName );
        printf ( "login machine = %s\n", requestInfo[index].LoginSourceMachine );
        printf ( "AppName = %s\n", requestInfo[index].AppName );
        printf ( "DbName = %s\n", requestInfo[index].DbName );
        printf ( "DbRequestCode = %u\n", requestInfo[index].DbRequestCode );
        printf ( "RequestString = %s\n", requestInfo[index].RequestString );
        printf ( "TimeStarted = %ul\n", requestInfo[index].TimeStarted );
        printf ( "State = %d\n", requestInfo[index].State );
        printf ( "\n\n--------------------------------------\n\n", requestInfo[index].State );

        sts = EssKillRequestEx (hCtx, &requestInfo[index] );
        index++;
    }
}

EssFree ( hInst, requestInfo );

userId = "native://nvid=f0ed2a6d7fb07688:5a342200:1265973105c:-7f46?USER ";
bIsIdentity = ESS_TRUE;
sts = EssListRequestsEx(hCtx, userId, bIsIdentity, "Sample", "Sample1", &numRequest, &requestInfo);
printf("\nEssListRequestsEx sts: %ld\n", sts);
printf ("Total requests on the server: %d\n", numRequest );
if ( !sts && requestInfo )
{
    while ( index < numRequest )
    {
        printf ( "login ID = %ul\n", requestInfo[index].LoginId );
        printf ( "user name = %s\n", requestInfo[index].UserName );
        printf ( "login machine = %s\n", requestInfo[index].LoginSourceMachine );
        printf ( "AppName = %s\n", requestInfo[index].AppName );
        printf ( "DbName = %s\n", requestInfo[index].DbName );
        printf ( "DbRequestCode = %u\n", requestInfo[index].DbRequestCode );
        printf ( "RequestString = %s\n", requestInfo[index].RequestString );
        printf ( "TimeStarted = %ul\n", requestInfo[index].TimeStarted );
        printf ( "State = %d\n", requestInfo[index].State );
        printf ( "\n\n--------------------------------------\n\n", requestInfo[index].State );
    }
requestInfo[index].State );

    sts = EssKillRequestEx (hCtx, &requestInfo[index] );

    index++;
}

    EssFree ( hInst, requestInfo );
}

See Also

- EssListRequestsEx

EssListAliases

Lists all the alias tables in the active database.

Syntax

EZS_FUNC_M EssListAliases (hCtx, pCount ppAliasList);

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>pCount</td>
<td>ESS_PUSHORT_T</td>
<td>Address of variable to receive count of alias tables.</td>
</tr>
<tr>
<td>ppAliasList</td>
<td>ESS_PPALIASNAME_T</td>
<td>Address of pointer to receive an allocated array of alias table names.</td>
</tr>
</tbody>
</table>

Notes

The memory allocated for ppAliasList should be freed using EssFree().

Return Value

If successful, this function returns a count of alias tables in pCount, and an allocated array of alias table names in ppAliasList.

Access

This function requires the caller to have at least read access (ESS_PRIV_READ) to the database, and to have selected it as the active database using EssSetActive().

Example

EZS_FUNC_M
EZS_ListAliases (ESS_HCTX_T hCtx,
    ESS_HINST_T hInst
    )
{
    EZS_FUNC_M       sts = ESS_STS_NOERR;
    EZS_USHORT_T     Count;
    EZS_USHORT_T     ind;
    EZS_PALIASNAME_T Altlist = NULL;
sts = EssListAliases (hCtx, &Count, &Altlist);
if (!sts)
{
    if (Count && Altlist)
    {
        printf ("\r\n-----List of Aliases-----\r\n\r\n");
        for (ind = 0; ind < Count; ind++)
        {
            if (Altlist [ind] != NULL)
                printf ("%s\r\n", Altlist[ind]);
        EssFree (hInst, Altlist);
    }
    else
        printf ("\r\nAlias List is Empty\r\n\r\n");
}
return (sts);
}

See Also

- EssDisplayAlias
- EssSetActive

**EssListApplications**

Lists all applications which are accessible to the caller.

**Syntax**

```c
ESS_FUNC_M EssListApplications (hCtx, pCount, ppAppList);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle</td>
</tr>
<tr>
<td>pCount</td>
<td>ESS_PUSHORT_T</td>
<td>Address of variable to receive count of returned applications</td>
</tr>
<tr>
<td>ppAppList</td>
<td>ESS_PPAPPNAME_T</td>
<td>Address of pointer to receive allocated array of application name strings</td>
</tr>
</tbody>
</table>

**Notes**

The memory allocated for `ppAppList` must be freed using `EssFree()`.

**Return Value**

If successful, this function returns a count of the number of accessible applications in `pCount`, and an array of application name strings in `ppAppList`. There are 'count' number of items in the array.

**Access**

This function requires no special privileges; note however that server applications will only be listed if the caller has access to them.
Example

```c
ESS_FUNC_M
ESS_ListApps (ESS_HCTX_T hCtx,
               ESS_HINST_T hInst
        )
{
    ESS_FUNC_M    sts = ESS_STS_NOERR;
    ESS_PAPPNAME_T strp = NULL;
    ESS_USHORT_T   Items;
    ESS_USHORT_T   ind;

    sts = EssListApplications (hCtx, &Items, &strp);
    if (!sts)
    {
        if (Items && strp)
        {
            printf ("Applications availables\r\n"
            for (ind = 0; ind < Items; ind++)
            {
                if (strp [ind] != NULL)
                    printf ("%s\r\n", strp [ind]);
            }
            EssFree (hInst, strp);
        }
        else
            printf ("Application List is Empty\r\n\r\n");
    }
    printf ("\r\n");
    return (sts);
}
```

See Also

- EssListDatabases
- EssListObjects

**EssListCalcFunctions**

Lists all calculator functions available in the active application. The list of available functions includes all native functions and all custom-defined functions (CDFs) and custom-defined macros (CDMs).

**Syntax**

```c
ESS_FUNC_M EssListCalcFunctions (hCtx, pCalcFunc);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>pCalcFunc</td>
<td>ESS_PSTR_T</td>
<td>Pointer to the string containing the available calculator functions. The string is in the form of XML.</td>
</tr>
</tbody>
</table>
Notes

EssListCalcFunctions() requires supervisor privilege (usually granted to the administrator). The user must also have database access to receive this list. To avoid an error, the user must have both supervisor privilege and access to the database to run a program with EssListCalcFunctions().

The contents of the string returned by EssGetCalcList is formatted as XML and must be either rendered in an XML utility or parsed to display only the actual text. All XML tags are enclosed in angle brackets (for example, <xml_tag>).

Here is a pared-down example of a typical XML output file:

ESSBASE API v.62000
1051034: Logging in user admin
1051035: Last login on Tuesday, May 22, 2001 10:31:19 AM
<list>
  <group name="Boolean">
    <function>
      <name><![CDATA[@ISACCTYPE]]></name>
      <syntax><![CDATA[@ISACCTYPE(tag)]></syntax>
      <comment><![CDATA[returns TRUE if the current member has the associated accounts tag]]></comment>
    </function>
  </group>
  <group name="Relationship Functions">
    <function>
      <name><![CDATA[@ANCESTVAL]]></name>
      <syntax><![CDATA[@ANCESTVAL (dimName, genLevNum [, mbrName])]]></syntax>
      <comment><![CDATA[returns the ancestor values of a specified member combination]]></comment>
    </function>
  </group>
  <group name="Custom">
  </group>
</list>

Return Value

Returns zero if successful; error code if unsuccessful.

Example

#include <iostream.h>
#include <fstream.h>
#include "windows.h"
#include "essbase.h"
#include "essapi.h"
#include "essotl.h"
#include "stdio.h"

/* globals - handles to different ESS objects */
ESS_HINST_T   hInst = 0;
ESS_HCTX_T     hCtx = 0;

ESS_APPNAME_T  szAppName;
ESS_DBNAME_T    szDbName;

ESS_HOUTLINE_T hOutline = 0;

/* end globals */

/* forward declarations of functions */
void apiInit();
void apiTerm();

ESS_STS_T apiAutoLogin();
ESS_STS_T apiLogout();

void apiInit()
{


}
ESS_STS_T apiAutoLogin ()
{
    ESS_CHAR_T SvrName[ESS_SVRNAMELEN];
    ESS_CHAR_T UserName[ESS_USERNAMELEN];
    ESS_CHAR_T Password[ESS_PASSWORDLEN];

    ESS_USHORT_T   Option;
    ESS_ACCESS_T   Access ;

    /* Initialize parameters */
    strcpy(SvrName,"localhost");
    strcpy(UserName," ");
    strcpy(Password," ");
    strcpy(szAppName," ");
    strcpy(szDbName," ");

    Option = AUTO_DEFAULT;

    /* Login to Essbase Server */
    return EssAutoLogin (hInst, SvrName, UserName, Password,
                        szAppName, szDbName, Option, &Access, &hCtx);
}

void apiTerm()
{
    ESS_STS_T sts = ESS_STS_NOERR;
    if ( hCtx )
        sts = apiLogout();
    if ( !sts && hInst )
        sts = EssTerm(hInst);
    if ( sts )
    {
        printf( "API shutdown failure: %d\n", sts );
        exit(1);
    }
}

ESS_STS_T apiLogout()
{
    return EssLogout(hCtx);
}

int main(int argc, char **argv)
{
    ESS_STS_T       status;
    ESS_STR_T       pszCalcFunctionList;
    apiInit();
    status = apiAutoLogin();
}
if ( status )
    return 1;

status = EssListCalcFunctions( hCtx, &pszCalcFunctionList );
if ( status )
    return 1;

printf( "%s\n", pszCalcFunctionList );

apiTerm();
return 0;
}

See Also
● EssGetFilterList

EssListConnections

Lists all users who are connected to the currently logged in server or application.

Syntax

ESS_FUNC_M EssListConnections (hCtx, pCount, ppUserList);

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle</td>
</tr>
<tr>
<td>pCount</td>
<td>ESS_PUSHORT_T</td>
<td>Variable to receive count of users</td>
</tr>
<tr>
<td>ppUserList</td>
<td>&quot;ESS_USERINFO_T, ESS_GROUPINFO_T&quot; on page 192</td>
<td>Pointer to an array of user information structures. This array is allocated by the API</td>
</tr>
</tbody>
</table>

Notes
● pCount contains the number of elements in the ppUserList array.
● If hCtx is a Supervisor, ppUserList is a list of users logged in to the server. If hCtx is an Application Designer, ppUserList is a list of users connected to any application for which hCtx is an Application Designer.
● Use EssFree() to free the buffer allocated for ppUserList.

Return Value

Returns 0 if successful.

Access

This function requires the caller to have Supervisor or Application Designer privilege.

Example

ESS_FUNC_M ESS_ListUserConnections (ESS_HCTX_T hCtx, ESS_HINST_T hInst)
{
    ESS_FUNC_M sts = ESS_STS_NOERR;
ESS_USHORT_T usrcnt;
ESS_PUSERINFO_T users;
sts = EssListConnections(hCtx, &usrcnt, &users);
if(!sts)
   EssFree(hInst, users);
return(sts);
}

See Also

- EssListConnectionsEx

EssListConnectionsEx

Lists all users who are connected to the currently logged in server or application. Similar to EssListConnections, but includes users hosted in a user directory.

Syntax

ESS_FUNC_M EssListConnectionsEx (hCtx, pCount, ppUserList);

Parameter | Data Type | Description
--- | --- | ---
hCtx | ESS_HCTX_T | API context handle (input).
pCount | ESS_PUSHORT_T | Address of variable to receive count of users (output).
ppUserList | ESS_PPUSERINFOID_T | Pointer to an array of user information structures (output). The information structures can include user directories and unique identity attributes.

Notes

- *pCount* contains the number of elements in the *ppUserList* array.
- If *hCtx* is a Supervisor, *ppUserList* is a list of users logged in to the server. If *hCtx* is an Application Designer, *ppUserList* is a list of users connected to any application for which *hCtx* is an Application Designer.
- Use EssFree() to free the buffer allocated for *ppUserList*.

Return Value

Returns 0 if successful.

Access

This function requires the caller to have Supervisor or Application Designer privilege.

Example

```c
void DisplayUserInfoID2(ESS_USERINFOID_T userInfo)
{
    ESS_STS_T sts  = ESS_STS_NOERR;
    ESS_BOOL_T       isDefined = ESS_TRUE;

    printf("\tUser Name: %s\n", userInfo.Name);
    printf("\tProvider Name: %s\n", userInfo.ProviderName);
```
printf("\tConnparam: %s\n", userInfo.connparam);
printf("\tDescription: %s\n", userInfo.Description);
printf("\tEMail Identification: %s\n", userInfo.EMailID);

if (userInfo.LockedOut)
    printf("\tLocked out: Yes\n");
else
    printf("\tLocked out: No\n");

if (userInfo.PwdChgNow)
    printf("\tChange the password now: Yes\n");
else
    printf("\tChange the password now: No\n");

printf("\tConnected Application: %s\n", userInfo.AppName);
printf("\tConnected Database: %s\n", userInfo.DbName);

if (userInfo.Login)
    printf("\tLogged in: Yes\n");
else
    printf("\tLogged in: No\n");

switch(userInfo.Access)
{
    case ESS_ACCESS_ADMIN:
        printf("\tAccess: %d - ESS_ACCESS_ADMIN\n", userInfo.Access);
        break;
    case ESS_ACCESS_APPALL:
        printf("\tAccess: %d - ESS_ACCESS_APPALL\n", userInfo.Access);
        break;
    case ESS_ACCESS_DBALL:
        printf("\tAccess: %d - ESS_ACCESS_DBALL\n", userInfo.Access);
        break;
    case ESS_ACCESS_APPCREATE:
        printf("\tAccess: %d - ESS_ACCESS_APPCREATE\n", userInfo.Access);
        break;
    case ESS_ACCESS_APPMANAGE:
        printf("\tAccess: %d - ESS_ACCESS_APPMANAGE\n", userInfo.Access);
        break;
    case ESS_ACCESS_DBCREATE:
        printf("\tAccess: %d - ESS_ACCESS_DBCREATE\n", userInfo.Access);
        break;
    case ESS_ACCESS_DBMANAGE:
        printf("\tAccess: %d - ESS_ACCESS_DBMANAGE\n", userInfo.Access);
        break;
    case ESS_ACCESS_CALC:
        printf("\tAccess: %d - ESS_ACCESS_CALC\n", userInfo.Access);
        break;
    case ESS_ACCESS_WRITE:
        printf("\tAccess: %d - ESS_ACCESS_WRITE\n", userInfo.Access);
        break;
    case ESS_ACCESS_READ:
        printf("\tAccess: %d - ESS_ACCESS_READ\n", userInfo.Access);
        break;
    case ESS_PRIV_USERCREATE:
        printf("\tAccess: %d - ESS_PRIV_USERCREATE\n", userInfo.Access);
        break;
}
case ESS_PRIV_APPCREATE:
    printf("%d - ESS_PRIV_APPCREATE\n", userInfo.Access);
    break;

case ESS_PRIV_APPMANAGE:
    printf("%d - ESS_PRIV_APPMANAGE\n", userInfo.Access);
    break;

case ESS_PRIV_APPLOAD:
    printf("%d - ESS_PRIV_APPLOAD\n", userInfo.Access);
    break;

case ESS_PRIV_DBCREATE:
    printf("%d - ESS_PRIV_DBCREATE\n", userInfo.Access);
    break;

case ESS_PRIV_DBMANAGE:
    printf("%d - ESS_PRIV_DBMANAGE\n", userInfo.Access);
    break;

case ESS_PRIV_DBLOAD:
    printf("%d - ESS_PRIV_DBLOAD\n", userInfo.Access);
    break;

case ESS_PRIV_CALC:
    printf("%d - ESS_ACCESS_DBALL\n", userInfo.Access);
    break;

case ESS_PRIV_WRITE:
    printf("%d - ESS_PRIV_WRITE\n", userInfo.Access);
    break;

case ESS_PRIV_READ:
    printf("%d - ESS_PRIV_READ\n", userInfo.Access);
    break;

case ESS_PRIV_NONE:
    printf("%d - ESS_PRIV_NONE\n", userInfo.Access);
    break;

default:
    printf("Unknown\n");
}

switch(userInfo.MaxAccess)
{
    case ESS_ACCESS_ADMIN:
        printf("%d - ESS_ACCESS_ADMIN\n", userInfo.MaxAccess);
        break;

    case ESS_ACCESS_APPALL:
        printf("%d - ESS_ACCESS_APPALL\n", userInfo.MaxAccess);
        break;

    case ESS_ACCESS_DBALL:
        printf("%d - ESS_ACCESS_DBALL\n", userInfo.MaxAccess);
        break;

    case ESS_ACCESS_APPCREATE:
        printf("%d - ESS_ACCESS_DBALL\n", userInfo.MaxAccess);
        break;

    case ESS_ACCESS_APPMANAGE:
        printf("%d - ESS_ACCESS_APPMANAGE\n", userInfo.MaxAccess);
        break;

    case ESS_ACCESS_DBCREATE:
        printf("%d - ESS_ACCESS_DBCREATE\n", userInfo.MaxAccess);
        break;

    case ESS_ACCESS_DBMANAGE:
        printf("%d - ESS_ACCESS_DBMANAGE\n", userInfo.MaxAccess);
        break;

    default:
        printf("Unknown\n");
}
case ESS_ACCESS_CALC:
    printf("\tMax Access: %d - ESS_ACCESS_CALC\n", userInfo.MaxAccess);
    break;

case ESS_ACCESS_WRITE:
    printf("\tMax Access: %d - ESS_ACCESS_WRITE\n", userInfo.MaxAccess);
    break;

case ESS_ACCESS_READ:
    printf("\tMax Access: %d - ESS_ACCESS_READ\n", userInfo.MaxAccess);
    break;

case ESS_PRIV_USERCREATE:
    printf("\tMax Access: %d - ESS_PRIV_USERCREATE\n", userInfo.MaxAccess);
    break;

case ESS_PRIV_APPCREATE:
    printf("\tMax Access: %d - ESS_PRIV_APPCREATE\n", userInfo.MaxAccess);
    break;

case ESS_PRIV_APPMANAGE:
    printf("\tMax Access: %d - ESS_PRIV_APPMANAGE\n", userInfo.MaxAccess);
    break;

case ESS_PRIV_APPLOAD:
    printf("\tMax Access: %d - ESS_PRIV_APPLOAD\n", userInfo.MaxAccess);
    break;

case ESS_PRIV_DBCREATE:
    printf("\tMax Access: %d - ESS_PRIV_DBCREATE\n", userInfo.MaxAccess);
    break;

case ESS_PRIV_DBMANAGE:
    printf("\tMax Access: %d - ESS_PRIV_DBMANAGE\n", userInfo.MaxAccess);
    break;

case ESS_PRIV_DBLOAD:
    printf("\tMax Access: %d - ESS_PRIV_DBLOAD\n", userInfo.MaxAccess);
    break;

case ESS_PRIV_CALC:
    printf("\tMax Access: %d - ESS_ACCESS_DBALL\n", userInfo.MaxAccess);
    break;

case ESS_PRIV_WRITE:
    printf("\tMax Access: %d - ESS_PRIV_WRITE\n", userInfo.MaxAccess);
    break;

case ESS_PRIV_READ:
    printf("\tMax Access: %d - ESS_PRIV_READ\n", userInfo.MaxAccess);
    break;

default:
    printf("\tMax Access: Unknown\n");

    printf("\tPassword Expiration in Dates: %d\n", userInfo.Expiration);
    //EssSdCTime(NULL, userInfo.LastLogin, sizeof(time_string), time_string);
    //printf("\tLast Successful Login:            %s\n", time_string);
    printf("\tFailed Login Attempts Since Then: %d\n", userInfo.FailCount);
    printf("\tLogin ID: %ld\n", userInfo.LoginId);
    printf(" \n");
}

ESS_FUNC_M ESS_ListUserConnectionsEx (ESS_HCTX_T hCtx, ESS_HINST_T hInst)
{  
ESS_STS_T sts = ESS_STS_NOERR;
ESS_HCTX_T hLocalCtx1;
ESS_HCTX_T hLocalCtx2;
ESS_USHORT_T usercount, i = 0;
ESS_USERINFOID_T userInfo = ESS_NULL;
ESS_USHORT_T Items;
ESS_PAPPDB_T pAppsDbs = ESS_NULL;

usercount = 0;
memset(&userInfo, '\0', sizeof(userInfo));
sts = EssListConnectionsEx(hCtx, &usercount, &userInfo);
printf("EssListConnectionsEx sts: %ld\n", sts);
if(!sts)
{  
    printf("\nConnection count(s): %d\n", usercount);
    for(i = 0; i < usercount; i++)
    {  
        DisplayUserInfoID2(userInfo[i]);
    }
}

    return(sts);
}

EssListCurrencyDatabases

Lists all currency databases within a specific application which are accessible to the caller.

Syntax

ESS_FUNC_M EssListCurrencyDatabases (hCtx, AppName, pCount, ppDbList);

Parameter  Data Type          Description
hCtx        ESS_HCTX_T        API context handle
AppName     ESS_STR_T         Application name
pCount      ESS_PUSHORT_T     Address of variable to receive count of currency databases
ppDbList    "ESS_APPDB_T" on page 114 Address of pointer to receive allocated array of application/databasename structures

Notes

- This function can only be used to list currency databases within an application on the server, not the client.
- The ppDbList argument returns an array of structures containing matching pairs of application and database name strings.
- The memory allocated for ppDbList should be freed using EssFree().
Return Value

If successful, this function returns a count of the number of accessible currency databases in `pCount`, and a list of applications/currency database names in `ppDbList`.

Access

This function requires no special privileges; note however that server currency databases will only be listed if the caller has access to them.

Example

```c
ESS_FUNC_M
ESS_ListCurrencyDatabases (ESS_HCTX_T hCtx, ESS_HINST_T hInst)
{
    ESS_FUNC_M     sts = ESS_STS_NOERR;
    ESS_USHORT_T  Items;
    ESS_USHORT_T  ind;
    ESS_STR_T     AppName;
    ESS_PAPPDB_T  pDbsList = NULL;
    AppName = "Sample";
    sts = EssListCurrencyDatabases(hCtx, AppName, &Items, &pDbsList);

    if(!sts)
    {
        if(Items && pDbsList)
        {
            printf("\n---- Currency Databases ----\n\n");
            for (ind = 0; ind<Items; ind++)
            {
                if((pDbsList+ind) !=NULL)
                {
                    if(pDbsList[ind].DbName != NULL)
                    {
                        printf("%s",AppName);
                        printf(" ==> ");
                        printf("%s",pDbsList[ind].DbName);
                        printf("\n\r\n");
                    }
                }
            }n
            EssFree(hInst, pDbsList);
        } else
            printf("\r\nCurrency Database List is Empty\n\r\n");
        return (sts);
    }
}

See Also

- EssGetDatabaseInfo
- EssGetDatabaseState
- EssListApplications
- EssListDatabases
- EssListObjects

448  C Main API Functions
**EssListDatabases**

Lists all databases which are accessible to the caller, either within a specific application, or on an entire server.

**Syntax**

```
ESS_FUNC_M EssListDatabases (hCtx, AppName, pCount, ppDbList);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle</td>
</tr>
<tr>
<td>AppName</td>
<td>ESS_STR_T</td>
<td>Application name</td>
</tr>
<tr>
<td>pCount</td>
<td>ESS_PUSHORT_T</td>
<td>Address of variable to receive count of applications and databases</td>
</tr>
<tr>
<td>ppDbList</td>
<td>“ESS_APPDB_T” on page 114</td>
<td>Address of pointer to receive allocated array of application/databasename structures</td>
</tr>
</tbody>
</table>

**Notes**

- If the AppName argument is NULL, this function lists all the accessible applications and databases on the server.
- The `ppDbList` argument returns an array of structures containing matching pairs of application and databasename strings.
- The memory allocated for `ppDbList` must be freed using `EssFree()`.

**Return Value**

If successful, this function returns a count of the number of accessible databases in `pCount`, and a list of the application and database names in `ppDbList`.

**Access**

This function requires no special privileges; note however that server databases will only be listed if the caller has access to them.

**Example**

```
ESS_FUNC_M
ESS_ListDbs (ESS_HCTX_T hCtx,
             ESS_HINST_T hInst
)
{
    ESS_FUNC_M    sts = ESS_STS_NOERR;
    ESS_USHORT_T  Items;
    ESS_USHORT_T  ind;
    ESS_PAPPDB_T  pAppsDbs = NULL;
    sts = EssListDatabases (hCtx, NULL, &Items,
                             &pAppsDbs);
    if (!sts)
    {
        if (Items && pAppsDbs)
        {
            printf ("\r\n-----Applications/databases available-----\r\n");
        }
    }
```

for (ind = 0; ind < Items; ind++)
{
    if ((pAppsDbs+ind) != NULL)
    {
        if ((pAppsDbs[ind].AppName != NULL) &&
            (pAppsDbs[ind].DbName != NULL))
        {
            printf ("%s", pAppsDbs[ind].AppName);
            printf (" ==> ");
            printf ("%s", pAppsDbs[ind].DbName);
            printf ("\n\r");
        }
    }
    EssFree (hInst, pAppsDbs);
}
else
    printf ("\r\ndatabaseList is Empty\n\r\n");
}

return(sts);

See Also

- EssGetDatabaseInfo
- EssGetDatabaseState
- EssListApplications
- EssListCurrencyDatabases
- EssListObjects

**EssListExistingLoadBuffers**

Returns the list of structures that describe existing data load buffers for an aggregate storage database.

This function returns the count of existing buffers and an array of descriptor structures. The memory for the array must be freed using EssFree.

**Syntax**

```
ESS_FUNC_M EssListExistingLoadBuffers (hCtx, AppName, DbName, pCount, paLoadBuffers);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle</td>
</tr>
<tr>
<td>AppName</td>
<td>ESS_STR_T</td>
<td>Use NULL. Function always applies to the currently selected database.</td>
</tr>
<tr>
<td>DbName</td>
<td>ESS_STR_T</td>
<td>Use NULL. Function always applies to the currently selected database.</td>
</tr>
<tr>
<td>pCount</td>
<td>ESS_PULONG_T</td>
<td>Address of variable to receive count of load buffers</td>
</tr>
</tbody>
</table>

450  C Main API Functions
### paLoadBuffers

- **Data Type**: "ESS_LOAD_BUFFER_T" on page 145
- **Description**: Pointer to load buffer information structure

### Return Value

Returns zero if successful; otherwise, returns an error code.

### Example

```c
void TestListExistingLoadBuffers(ESS_HCTX_T hCtx, ESS_STR_T AppName, ESS_STR_T DbName)
{
    ESS_STS_T       sts = ESS_STS_NOERR;
    ESS_LOAD_BUFFER_T *LoadBuffers;
    ESS_ULONG_T i;
    ESS_ULONG_T     Count;
    /* EssListExistingLoadBuffers */

    sts = EssListExistingLoadBuffers(hCtx, AppName, DbName, &Count, &LoadBuffers);
    printf("EssListExistingLoadBuffers sts: %ld\n",sts);
    printf("\tNumber of buffers: %d", Count);
    if(Count > 0)
    {
        for(i = 0; i < Count; i++)
        {
            printf("\n\tBuffer Id: %d", LoadBuffers[i].ulBufferId);
            printf("\n\tDuplicate Agg Method: %d", LoadBuffers[i].ulDuplicateAggregationMethod);
            printf("\n\tOption Flags: %d", LoadBuffers[i].ulOptionFlags);
            printf("\n\tSize (1-100): %d", LoadBuffers[i].ulSize);
            printf("\n\tInternal: %d", LoadBuffers[i].bInternal);
            printf("\n\tActive: %d", LoadBuffers[i].bActive);
            printf("\n")
        }
    }
}
```

### See Also

- EssLoadBufferInit
- EssBeginDataloadASO
- EssSendString
- EssEndDataload
- EssLoadBufferTerm
- EssImportASO
- EssUpdateFileASO
- EssUpdateFileUTF8ASO
- EssMergeDatabaseData

### EssListDbFiles

Retrieves information on specified index and data files.
Syntax

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx;</td>
<td>ESS_HCTX_T</td>
<td>Context handle</td>
</tr>
<tr>
<td>szAppName;</td>
<td>ESS_STR_T</td>
<td>Application name</td>
</tr>
<tr>
<td>szDbName;</td>
<td>ESS_STR_T</td>
<td>Database name</td>
</tr>
<tr>
<td>usFileType;</td>
<td>ESS_USHORT_T</td>
<td>One of the following file types to be returned:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESS_FILETYPE_INDEX</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESS_FILETYPE_DATA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESS_FILETYPE_INDEX</td>
</tr>
<tr>
<td>pNmbrofFiles;</td>
<td>ESS_PUSHORT_T</td>
<td>Pointer to the number of index and data files returned</td>
</tr>
<tr>
<td>ppDbInfoArray;</td>
<td>“ESS_DBFILEINFO_T” on page 123</td>
<td>Pointer to an array of database file information structures returned</td>
</tr>
</tbody>
</table>

Return Value

- If successful,
- EssListDbFiles() returns 0
- pNmbrofFiles contains a pointer to the number of index and data files returned
- ppDbInfoArray contains a pointer to an array of database file information structures returned

Example

```c
#define ESS_STS_T ESS_STS_T
#define ESS_HCTX_T ESS_HCTX_T
#define ESS_STR_T ESS_STR_T
#define ESS_USHORT_T ESS_USHORT_T

ESS_STS_T ListDbFiles( ESS_HCTX_T hCtx )
{
    ESS_STS_T         sts         = ESS_STS_NOERR;
    ESS_APPNAME_T     pszAppName;
    ESS_DBNAME_T      pszDbName;
    ESS_PDBFILEINFO_T aDbFileInfo = NULL;
    ESS_PDBFILEINFO_T pDbFileInfo = NULL;
    ESS_USHORT_T      usFileType  = ( ESS_FILETYPE_INDEX | ESS_FILETYPE_DATA );
    ESS_USHORT_T      usFileCount = 0;
    ESS_USHORT_T      usFileIx;

    /****************************************************************************
     * Prompt for the type of files to list: index, data or both, *              
     * and assign the user's file type choice to usFileType                    
     ****************************************************************************
    
    /****************************************************************************
     * This function uses ESS_FILETYPE_INDEX | ESS_FILETYPE_DATA               
     * as the default value                                                      
     ****************************************************************************/

    .
    .
    .
```
Prompt for application and database names, and assign the user's choices to pszAppName and pszDbName, respectively.

Get an array of persistent database file information from Essbase for the selected file type, application and database.

sts = EssListDbFiles( hCtx, pszAppName, pszDbName, usFileType, &usFileCount, &aDbFileInfo );

if ( sts )
{
    goto exit;
}

Format and display the information in the persistent database file information array.

if ( ( usFileCount ) && ( aDbFileInfo ) )
{
    printf( "Application Name: %s\n", aDbFileInfo[ 0 ].AppName );
    printf( "Database Name: %s\n", aDbFileInfo[ 0 ].DbName );

    for ( ( usFileIx = 0, usFileType = 0 );
        usFileIx < usFileCount;
        usFileIx++ )
    {
        pDbFileInfo = &( aDbFileInfo[ usFileIx ] );

        printf( "\nFile %lu:\n", pDbFileInfo->FileSequenceNum );
        printf( "   File Name: %s\n", pDbFileInfo->FilePath );
        printf( "   File Type: " );
        if ( pDbFileInfo->FileType == ESS_FILETYPE_INDEX )
        {
            printf( "INDEX\n" );
        }
        else
        {
            printf( "DATA\n" );
        }
    }
}
printf( "   File Number:        %lu of %lu\n",
      pDbFileInfo->FileSequenceNum, pDbFileInfo->FileCount );

printf( "   File Size:          %lu\n",
      pDbFileInfo->FileSize );

printf( "   File Opened:        %c\n",
      ( pDbFileInfo->FileOpen ) ? 'Y' : 'N' );
}

} /* FOR usFileIx */

/***********************
* Free the memory allocated for the persistent *
* database file information array               *
*************************/
free( aDbFileInfo );

else
{

printf( "Application Name:      %s\n",
       AppName );

printf( "Database Name:         %s\n",
       DbName );

switch ( usFileType )
{
  case ESS_FILETYPE_INDEX:
    printf( "\nNo existing INDEX files.\n" );
    break;
  case ESS_FILETYPE_DATA:
    printf( "\nNo existing DATA files.\n" );
    break;
  case ( ESS_FILETYPE_INDEX | ESS_FILETYPE_DATA ):
    printf( "\nNo existing INDEX or DATA files.\n" );
    break;
      default:
        printf( "\nNo existing database files of the selected type.\n" );
        break;
} /* SWITCH usFileType */

printf( "\n" );

exit:

    return ( sts );
}

See Also

- “ESS_DBFILEINFO_T” on page 123
EssListDrillThruURLs

Lists the drill-through URL names within the active database outline.

See “Drill-through URL Limits” on page 1729.

Syntax

ESS_FUNC_M EssListDrillThruURLs (hCtx, &pCountOfUrls, &pUrls);

Parameter | Data Type | Description
--- | --- | ---
hCtx | ESS_HCTX_T | API context handle
pCountOfUrls | ESS_PUSHORT_T | Count of drill-through URLs
pUrls | ESS_PPDURLINFO_T | List of URLs

Notes

The ESS_DURLINFO_T structure array must be deallocated by the caller using EssFreeStructure() with the ESS_DT_STRUCT_URLINFO option.

Return Value

- If successful, lists drill-through URLs in the active database outline.
- If unsuccessful, returns an error code.

Access

- Caller must have database Read privilege (ESS_PRIV_READ) for the specified database.
- Caller must have selected the specified database as the active database using EssSetActive().

Example

```c
static void DisplayUrlDefn (ESS_PPDURLINFO_T pUrls )
{
    ESS_UINT_T i;

    printf("\tUrlname : %s\n", pUrls->cpURLName);
    if (pUrls->bIsLevel0)
        printf("\tUrl Is Level-0 slice : Yes\n");
    else
        printf("\tUrl Is Level-0 slice : No\n");

    printf("\tUrlXmlsize : %i\n", pUrls->iURLXmlSize);
    printf("\tUrlXml : %s\n", (ESS_STR_T) pUrls->cpURLXml);

    printf("\tNumber of drill region(s) : %d\n", pUrls->iCountOfDrillRegions);
    for (i = 0; i < pUrls->iCountOfDrillRegions; i++ )
    {
        printf("\t\tDrillRegion[%d] : %s\n", i, pUrls->cppDrillRegions[i] );
    }
    printf("\n");
}

ESS_STS_T sts = ESS_STS_NOERR;
```
ESS_USHORT_T usCountOfURLs, i;
ESS_PDURLINFO_T listOfURLs;
ESS_DURLINFO_T url;
/* Valid case*/
sts = EssListDrillThruURLs(hCtx, &usCountOfURLs, &listOfURLs);
printf("EssListDrillThruURLs sts: %ld\n", sts);
if(!sts)
{
    printf("\tCount of URL: %d\n", usCountOfURLs);
    printf("\tList of URL(s):\n");
    for(i = 0; i < usCountOfURLs; i++)
    {
        DisplayUrlDefn (&listOfURLs[i]);
    }
}
EssFreeStructure (hInst, ESS_DT_STRUCT_URLINFO, usCountOfURLs, listOfURLs);

**EssListExtUsers**

Lists users who are externally authenticated through Shared Services.

**Syntax**

ESS_FUNC_M EssListExtUsers (hCtx, AppName, DbName, Protocol, Count, ppUserList);

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>AppName</td>
<td>ESS_STR_T</td>
<td>Application name. If NULL, lists all users.</td>
</tr>
<tr>
<td>DbName</td>
<td>ESS_STR_T</td>
<td>Database name. If NULL, lists users for all databases within application.</td>
</tr>
<tr>
<td>Protocol</td>
<td>ESS_STR_T</td>
<td>External authentication protocol: CSS, for Shared Services mode. Even if the protocol is not specified, this function returns a list of users who are externally authenticated through Shared Services.</td>
</tr>
<tr>
<td>Count</td>
<td>ESS_PUSHORT_T</td>
<td>Address of variable to receive count of users.</td>
</tr>
<tr>
<td>ppUserList</td>
<td>“ESS_USERINFOEX_T” on page 195</td>
<td>Address of pointer to receive an allocated array of user info structures. The AppName and DbName fields of the returned user info structures contain NULL values.</td>
</tr>
</tbody>
</table>

**Notes**

- If both AppName and DbName are not NULL, only users with access to the specified application and database are listed. If DbName is NULL, only users with access to the specified application are listed. If AppName is NULL, all users that exist on the server are listed.
- The AppName and DbName fields of the returned ESS_USERINFO_T structures contain NULL values.
- The memory allocated for ppUserList should be freed using EssFree().

456  C Main API Functions
**Return Value**

If successful, returns a count of the number of users in `pCount`, and list of users with access to the specified application and database in `ppUserList`.

**Access**

This function requires no special privileges.

**See Also**

- EssSetExtUser
- EssCreateExtUser
- EssGetExtUser

---

**EssListFilters**

Lists all filters for a database.

**Syntax**

```c
ESS_FUNC_M EssListFilters (hCtx,AppName, DbName, Count, ppFilterList);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>AppName</td>
<td>ESS_STR_T</td>
<td>Application name.</td>
</tr>
<tr>
<td>DbName</td>
<td>ESS_STR_T</td>
<td>Database name.</td>
</tr>
<tr>
<td>pCount</td>
<td>ESS_PUSHORT_T</td>
<td>Address of variable to receive count of filter names.</td>
</tr>
<tr>
<td>ppFilterList</td>
<td>ESS_PPFTRNAME_T</td>
<td>Address of pointer to receive an allocated array of filter name strings.</td>
</tr>
</tbody>
</table>

**Notes**

The memory allocated for `ppFilterList` should be freed using EssFree().

**Return Value**

If successful, returns the count of filters in the database in `pCount`, and an array of filter names in `ppFilterList`.

**Access**

This function requires the caller to have database Design privilege (ESS_PRIV_DBDESIGN) for the specified database.

**Example**

```c
ESS_FUNC_M
ESS_ListFilters (ESS_HCTX_T hCtx, ESS_HINST_T hInst) {
    ESS_FUNC_M    sts = ESS_STS_NOERR;
    ESS_STR_T     AppName;
    ESS_STR_T     DbName;
```
ESS_USHORT_T    Count = 0;
ESS_USHORT_T    ind;
ESS_PFTRNAME_T  pFilterList = NULL;

AppName = "Sample";
DbName  = "Basic";

sts = EssListFilters(hCtx, AppName, DbName,
                      &Count, &pFilterList);

if(!sts)
{
    if(Count && pFilterList)
    {
        printf("\r\n-------Filter List-------\n\r\n");
        for (ind = 0; ind < Count; ind++)
            printf("%s\r\n",pFilterList[ind]);
        EssFree(hInst, pFilterList);
    }
    else
        printf("\r\nFilter List is empty\r\n\n");
}
return (sts);

See Also

- EssGetFilter
- EssSetFilter

EssListFilterUsers

Lists all users using a filter.

Syntax

ESS_FUNC_M EssListFilterUsers (hCtx, dbName, AppName, UserCount, ppUserInfo);

Parameter  | Data Type  | Description
---------- |------------ |-------------
hCtx       | ESS_HCTX_T  | API context handle.       
dbName     | ESS_STR_T   | Database name          
AppName    | ESS_STR_T   | Application name        
UserCount  | ESS_PUSHORT_T | Count of users using the filter
ppUserInfo | ESS_PPUSERINFO_T | Pointer to array of UserInfo structures

Return Value

Returns zero if successful; error code if unsuccessful.
See Also

- EssListFilters

**EssListGroups**

Lists all groups who have access to a particular Essbase Server, application or database.

**Syntax**

```c
ESS_FUNC_M EssListGroups (hCtx,AppName,DbName,pCount,ppGroupList);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>AppName</td>
<td>ESS_STR_T</td>
<td>Application name. If NULL, lists all groups.</td>
</tr>
<tr>
<td>DbName</td>
<td>ESS_STR_T</td>
<td>Database name. If NULL, lists groups for all databases within application.</td>
</tr>
<tr>
<td>pCount</td>
<td>ESS_PUSHORT_T</td>
<td>Address of variable to receive count of groups.</td>
</tr>
<tr>
<td>ppGroupList</td>
<td>“ESS_USERINFO_T, ESS_GROUPINFO_T” on page 192</td>
<td>Address of pointer to receive an allocated array of group info structures.</td>
</tr>
</tbody>
</table>

**Notes**

- If both `AppName` and `DbName` are not NULL, only groups with access to the specified application and database will be listed. If `DbName` is NULL, only groups with access to the specified application will be listed. If `AppName` is NULL, all groups on the logged in server will be listed.
- The memory allocated for `ppGroupList` should be freed using `EssFree()`.

**Return Value**

If successful, returns a count of the number of groups in `pCount`, and list of groups with access to the specified application and database in `ppGroupList`.

**Access**

This function requires no special privileges.

**Example**

```c
ESS_FUNC_M
ESS_ListGroups (ESS_HCTX_T hCtx,
                ESS_HINST_T hInst)
{
    ESS_FUNC_M sts = ESS_STS_NOERR;
    ESS_USHORT_T Count;
    ESS_PGROUPINFO_T Groups = NULL;
    ESS_USHORT_T ind;
    sts = EssListGroups (hCtx, NULL, NULL, &Count, &Groups);
```
if (!sts)
{
    if (Count && Groups)
    {
        printf ("\r\n----Group List-----\r\n\n");
        for (ind = 0; ind < Count; ind++)
            printf ("Name->%s\r\n", Groups [ind].Name);
        EssFree (hInst, Groups);
    }
    else
        printf ("\r\nGroup List is Empty\r\n\n");
}

return (sts);

See Also

- EssListGroupsInfoEx
- EssGetGroup
- EssListUsers

**EssListGroupsInfoEx**

Lists all groups who have access to a particular Essbase Server, application or database. Similar to **EssListGroups**, but the group list structure can include user directories and unique identity attributes.

**Syntax**

```
ESS_FUNC_M EssListGroupsInfoEx (hCtx, AppName, DbName, pCount, ppGroupList);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle (input).</td>
</tr>
<tr>
<td>AppName</td>
<td>ESS_STR_T</td>
<td>Application name (input). If NULL, lists all groups.</td>
</tr>
<tr>
<td>DbName</td>
<td>ESS_STR_T</td>
<td>Database name (input). If NULL, lists groups for all databases within the application.</td>
</tr>
<tr>
<td>pCount</td>
<td>ESS_PUSHORT_T</td>
<td>Address of variable to receive count of groups (output).</td>
</tr>
<tr>
<td>ppGroupList</td>
<td>ESS_PPGROUPINFOID_T</td>
<td>Address of pointer to receive an allocated array of group info structures (output). The group list structure can include user directories and unique identity attributes.</td>
</tr>
</tbody>
</table>

**Notes**

- If both **AppName** and **DbName** are not NULL, only groups with access to the specified application and database will be listed. If **DbName** is NULL, only groups with access to the specified application will be listed. If **AppName** is NULL, all groups on the logged in server will be listed.

- The memory allocated for **ppGroupList** should be freed using **EssFree()**.
Return Value

If successful, returns a count of the number of groups in pCount, and list of groups with access to the specified application and database in ppGroupList.

Access

This function requires no special privileges.

Example

```c
void DisplayGroupsInfoEx(ESS_GROUPINFOID_T groupInfo)
{
    ESS_STS_T sts  = ESS_STS_NOERR;
    ESS_BOOL_T isDefined = ESS_TRUE;

    printf("\tUser Name: %s\n", groupInfo.Name);
    printf("\tProvider Name: %s\n", groupInfo.ProviderName);
    printf("\tIdentity: %s\n", groupInfo.connparam);
    printf("\tDescription: %s\n", groupInfo.Description);
    printf("\tEMail Identification: %s\n", groupInfo.EMailID);

    if (groupInfo.LockedOut)
        printf("\tLocked out: Yes\n");
    else
        printf("\tLocked out: No\n");

    if (groupInfo.PwdChgNow)
        printf("\tChange the password now: Yes\n");
    else
        printf("\tChange the password now: No\n");

    printf("\tPassword: %s\n", groupInfo.Password);
    printf("\tApplication: %s\n", groupInfo.AppName);
    printf("\tDatabase: %s\n", groupInfo.DbName);

    if (groupInfo.Login)
        printf("\tLogged in: Yes\n");
    else
        printf("\tLogged in: No\n");

    switch(groupInfo.Access)
    {
        case ESS_ACCESS_ADMIN:
            printf("\tAccess: %d - ESS_ACCESS_ADMIN\n", groupInfo.Access);
            break;
        case ESS_ACCESS_APPALL:
            printf("\tAccess: %d - ESS_ACCESS_APPALL\n", groupInfo.Access);
            break;
        case ESS_ACCESS_DBALL:
            printf("\tAccess: %d - ESS_ACCESS_DBALL\n", groupInfo.Access);
            break;
        case ESS_ACCESS_APPCREATE:
            printf("\tAccess: %d - ESS_ACCESS_APPCREATE\n", groupInfo.Access);
            break;
        case ESS_ACCESS_APPMANAGE:
            printf("\tAccess: %d - ESS_ACCESS_APPMANAGE\n", groupInfo.Access);  
            break;
    }
}
```

break;
case ESS_ACCESS_DBCREATE:
    printf("Access: %d - ESS_ACCESS_DBCREATE\n", groupInfo.Access);
    break;
case ESS_ACCESS_DBMANAGE:
    printf("Access: %d - ESS_ACCESS_DBMANAGE\n", groupInfo.Access);
    break;
case ESS_ACCESS_CALC:
    printf("Access: %d - ESS_ACCESS_CALC\n", groupInfo.Access);
    break;
case ESS_ACCESS_WRITE:
    printf("Access: %d - ESS_ACCESS_WRITE\n", groupInfo.Access);
    break;
case ESS_ACCESS_READ:
    printf("Access: %d - ESS_ACCESS_READ\n", groupInfo.Access);
    break;
case ESS_PRIV_USERCREATE:
    printf("Access: %d - ESS_PRIV_USERCREATE\n", groupInfo.Access);
    break;
case ESS_PRIV_APPCREATE:
    printf("Access: %d - ESS_PRIV_APPCREATE\n", groupInfo.Access);
    break;
case ESS_PRIV_APPMANAGE:
    printf("Access: %d - ESS_PRIV_APPMANAGE\n", groupInfo.Access);
    break;
case ESS_PRIV_APPLOAD:
    printf("Access: %d - ESS_PRIV_APPLOAD\n", groupInfo.Access);
    break;
case ESS_PRIV_DBCREATE:
    printf("Access: %d - ESS_PRIV_DBCREATE\n", groupInfo.Access);
    break;
case ESS_PRIV_DBMANAGE:
    printf("Access: %d - ESS_PRIV_DBMANAGE\n", groupInfo.Access);
    break;
case ESS_PRIV_DBLOAD:
    printf("Access: %d - ESS_PRIV_DBLOAD\n", groupInfo.Access);
    break;
case ESS_PRIV_CALC:
    printf("Access: %d - ESS_ACCESS_DBALL\n", groupInfo.Access);
    break;
case ESS_PRIV_WRITE:
    printf("Access: %d - ESS_PRIV_WRITE\n", groupInfo.Access);
    break;
case ESS_PRIV_READ:
    printf("Access: %d - ESS_PRIV_READ\n", groupInfo.Access);
    break;
case ESS_PRIV_NONE:
    printf("Access: %d - ESS_PRIV_NONE\n", groupInfo.Access);
    break;
default:
    printf("Access: Unknown\n");
}

switch(groupInfo.MaxAccess)
{
    case ESS_ACCESS_ADMIN:
        printf("Max Access: %d - ESS_ACCESS_ADMIN\n", groupInfo.MaxAccess);
        break;
}
break;
case ESS_ACCESS_APPALL:
    printf("\tMax Access: %d - ESS_ACCESS_APPALL\n", groupInfo.MaxAccess);
    break;
case ESS_ACCESS_DBALL:
    printf("\tMax Access: %d - ESS_ACCESS_DBALL\n", groupInfo.MaxAccess);
    break;
case ESS_ACCESS_APPCREATE:
    printf("\tMax Access: %d - ESS_ACCESS_APPCREATE\n", groupInfo.MaxAccess);
    break;
case ESS_ACCESS_APPMANAGE:
    printf("\tMax Access: %d - ESS_ACCESS_APPMANAGE\n", groupInfo.MaxAccess);
    break;
case ESS_ACCESS_DBCREATE:
    printf("\tMax Access: %d - ESS_ACCESS_DBCREATE\n", groupInfo.MaxAccess);
    break;
case ESS_ACCESS_DBMANAGE:
    printf("\tMax Access: %d - ESS_ACCESS_DBMANAGE\n", groupInfo.MaxAccess);
    break;
case ESS_ACCESS_CALC:
    printf("\tMax Access: %d - ESS_ACCESS_CALC\n", groupInfo.MaxAccess);
    break;
case ESS_ACCESS_WRITE:
    printf("\tMax Access: %d - ESS_ACCESS_WRITE\n", groupInfo.MaxAccess);
    break;
case ESS_ACCESS_READ:
    printf("\tMax Access: %d - ESS_ACCESS_READ\n", groupInfo.MaxAccess);
    break;
case ESS_PRIV_USERCREATE:
    printf("\tMax Access: %d - ESS_PRIV_USERCREATE\n", groupInfo.MaxAccess);
    break;
case ESS_PRIV_APPCREATE:
    printf("\tMax Access: %d - ESS_PRIV_APPCREATE\n", groupInfo.MaxAccess);
    break;
case ESS_PRIV_APPMANAGE:
    printf("\tMax Access: %d - ESS_PRIV_APPMANAGE\n", groupInfo.MaxAccess);
    break;
case ESS_PRIV_DBCREATE:
    printf("\tMax Access: %d - ESS_PRIV_DBCREATE\n", groupInfo.MaxAccess);
    break;
case ESS_PRIV_DBMANAGE:
    printf("\tMax Access: %d - ESS_PRIV_DBMANAGE\n", groupInfo.MaxAccess);
    break;
case ESS_PRIV_CALC:
    printf("\tMax Access: %d - ESS_ACCESS_DBALL\n", groupInfo.MaxAccess);
    break;
case ESS_PRIV_WRITE:
    printf("\tMax Access: %d - ESS_PRIV_WRITE\n", groupInfo.MaxAccess);
    break;
case ESS_PRIV_READ:
    printf("\tMax Access: %d - ESS_PRIV_READ\n", groupInfo.MaxAccess);
break;
case ESS_PRIV_NONE:
    printf("\tMax Access: %d - ESS_PRIV_NONE\n", groupInfo.MaxAccess);
    break;
default:
    printf("\tMax Access: Unknown\n");
}
printf("\tPassword Expiration in Days: %d\n",groupInfo.Expiration);
printf("\tFailed Login Attempts Since Then: %d\n", groupInfo.FailCount);
printf("\tLogin ID: %d\n", groupInfo.LoginId);
printf("\tProtocol: %s\n", groupInfo.protocol);
printf("\tConnection Parameter: %s\n", groupInfo.connparam);
printf("\n");

} // ESS_FUNC_M ESS_ListGroupsInfoEx (ESS_HCTX_T hCtx, ESS_HINST_T hInst)

{  
    ESS_STS_T sts  = ESS_STS_NOERR;
    ESS_USHORT_T count, i;
    ESS_PGROUPINFOID_T pGroupList = ESS_NULL;

    sts = EssListGroupsInfoEx(hCtx, AppName, DbName, &count, &pGroupList);
    printf("EssListGroupsInfoEx sts: %ld\n", sts);
    if(!sts)
    {
        printf("\nNumber of group(s): %d\n", count);
        for(i = 0; i < count; i++)
        {
            DisplayGroupsInfoEx(pGroupList[i]);
        }
    }

    return (sts);
}

See Also
- EssGetGroupInfoEx
- EssListUsersInfoEx

## EssListLocks

Lists all users who are connected to a specific application and database, together with a count of data blocks which they currently have locked.

### Syntax

```c
ESS_FUNC_M EssListLocks (hCtx, AppName, DbName, pCount, ppLockList);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
</tbody>
</table>
Parameter | Data Type | Description
---|---|---
AppName | ESS_STR_T | Application name.
DbName | ESS_STR_T | Databasename.
pCount | ESS_PUSHORT_T | Address of variable to receive count of users.
ppLockList | “ESS_LOCKINFO_T” on page 146 | Address of pointer to receive an allocated array of user lock info structures.

Notes
- This function is a "snapshot", in that only those users who are connected to the server when this function is called will be listed.
- The memory allocated for ppLockList should be freed using EssFree().

Return Value
If successful, returns a count of the number of connected users in pCount, and list of user lock structures in ppLockList.

Access
This function requires the caller to have database Design privilege (ESS_PRIV_DBDESIGN) for the specified database.

Example
```c
ESS_FUNC_M
ESS_ListLocks (ESS_HCTX_T hCtx,
                ESS_HINST_T hInst
)
{
    ESS_FUNC_M          sts;
    ESS_USHORT_T     Count;
    ESS_PLOCKINFO_T  plockinfo = NULL;
    ESS_STR_T        AppName;
    ESS_STR_T        DbName;
    AppName = "Sample";
    DbName  = "Basic";
    sts = EssListLocks (hCtx, AppName, DbName,
                        &Count, &plockinfo);
    if (!sts)
    {
        if (Count && plockinfo)
            EssFree (hInst, plockinfo);
        else
            printf ("\n\nExclusive Lock List on %s:%s is empty\n", AppName, DbName);
    }
    return (sts);
}
```

See Also
- EssListLocksEx
- EssListConnections
EssListLocksEx

Lists all users who are connected to a specific application and database, together with a count of data blocks which they currently have locked. Similar to EssListLocks, but includes users hosted in a user directory.

Syntax

\[ \text{ESS_FUNC_M EssListLocksEx (hCtx,AppName,DbName,pCount,ppLockList);} \]

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle (input).</td>
</tr>
<tr>
<td>AppName</td>
<td>ESS_STR_T</td>
<td>Application name (input).</td>
</tr>
<tr>
<td>DbName</td>
<td>ESS_STR_T</td>
<td>Database name (input).</td>
</tr>
<tr>
<td>pCount</td>
<td>ESS_PUSHORT_T</td>
<td>Address of variable to receive the user count (output).</td>
</tr>
<tr>
<td>ppLockList</td>
<td>ESS_PPLOCKINFOEX_T</td>
<td>Address of pointer to receive an allocated array of user lock information structures (output). The information structures can include user directories and unique identity attributes.</td>
</tr>
</tbody>
</table>

Notes

- This function is a "snapshot", in that only those users who are connected to the server when this function is called will be listed.
- The memory allocated for ppLockList should be freed using EssFree().

Return Value

If successful, returns a count of the number of connected users in pCount, and list of user lock structures in ppLockList.

Access

This function requires the caller to have database Design privilege (ESS_PRIV_DBDESIGN) for the specified database.

Example

```c
void DisplayLock(ESS_LOCKINFOEX_T lockinfo)
{
    ESS_STS_T sts = ESS_STS_NOERR;

    printf(\"tUser Name: %s\n", lockinfo.UserName);
    printf(\"tProvider Name: %s\n", lockinfo.ProviderName);
    printf(\"tConnection Parameter: %s\n", lockinfo.connparam);
    printf(\"tNumber of Locks: %d\n", lockinfo.nLocks);
    printf(\"tTime: %ld\n", lockinfo.Time);
}"
```


```c
printf("\tLoginId: %ld\n", lockinfo.LoginId);
printf("\n");
}

ESS_FUNC_M ESS_ListLocksEx (ESS_HCTX_T hCtx, ESS_HINST_T hInst)
{
    ESS_STS_T sts;
    ESS_USHORT_T count, i;
    ESS_PLOCKINFOEX_T plockinfo = NULL;
    ESS_ACCESS_T Access;

    sts = EssSetActive(hCtx, AppName, DbName, &Access);
    printf("EssSetActive sts: %ld\n", sts);

    sts = EssListLocksEx (hCtx, AppName, DbName, &count, &plockinfo);
    printf("EssListLocksEx sts: %ld\n", sts);
    if(!sts)
    {
        printf("Number of lock info returned: %d\n", count);
        for(i = 0; i < count; i++)
        {
            DisplayLock(plockinfo[i]);
        }
    }
    return (sts);
}

See Also

- EssListConnectionsEx
- EssListUsersInfoEx

**EssListLogins**

Returns the list of login instances in the current session.

**Syntax**

```c
ESS_FUNC_M EssListLogins (hCtx, count, logins);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>count</td>
<td>ESS_PUSHORT_T</td>
<td>Pointer to the number of logins returned from the server.</td>
</tr>
<tr>
<td>logins</td>
<td>ESS_PPCONNECTINFO_T</td>
<td>Pointer to an array of a ESS_CONNECTINFO_T structure containing connection information.</td>
</tr>
</tbody>
</table>

**Notes**

You can call **EssListLogins()** more than once for the same user name and server. The API returns a unique context handle for each login to the server.
**Return Value**
If successful, returns login information and a count of current logins.

**Access**
Before calling this function, you must first initialize the API and obtain a valid instance handle by calling `EssInit()`.

**See Also**
- `EssListLoginsEx`
- `EssAutoLogin`
- `EssInit`
- `EssListDatabases`
- `EssLogout`
- `EssSetActive`

---

**EssListLoginsEx**

Returns the list of login instances in the current session. Similar to `EssListLogins`, but includes users hosted in a user directory.

**Syntax**

```c
ESS_FUNC_M EssListLoginsEx (hCtx, count, logins);
```

**Parameter Data Type Description**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle (input).</td>
</tr>
<tr>
<td>count</td>
<td>ESS_PUSHORT_T</td>
<td>Pointer to the number of logins returned from the server (output).</td>
</tr>
<tr>
<td>logins</td>
<td>ESS_PPCONNECTINFOEX_T</td>
<td>Pointer to an array of an ESS_CONNECTINFOEX_T structure containing connection information (output). The information structure can include user directories and unique identity attributes.</td>
</tr>
</tbody>
</table>

**Notes**
You can call this function more than once for the same user name and server. The API returns a unique context handle for each login to the server.

**Return Value**
If successful, returns login information and a count of current logins.

**Access**
Before calling this function, you must first initialize the API and obtain a valid instance handle by calling `EssInit()`.

**Example**

```c
void DisplayLoginInfo(ESS_CONNECTINFOEX_T login)
{
```
ESS_STS_T sts = ESS_STS_NOERR;

printf("\tName: \s
", login.Name);
printf("\tApp Name: \s
", login.AppName);
printf("\tDb Name: \s
", login.DbName);
printf("\tLogin MachineName: \s
", login.LoginMachine);
printf("\tLogin Ip: %ld\n", login.LoginIP);
printf("\tLast login time: %ld\n", login.LastLogin);
printf("\n");

}

ESS_FUNC_M ESS_ListLoginsEx (ESS_HCTX_T hCtx, ESS_HINST_T hInst)
{
    ESS_STS_T sts = ESS_STS_NOERR;
    ESS_HCTX_T hLocalCtx1;
    ESS_HCTX_T hLocalCtx2;
    ESS_USHORT_T count, i;
    ESS_PCONNECTINFOEX_T logins;
    ESS_USHORT_T Items;
    ESS_PAPDB_T pAppsDbs = ESS_NULL;
    ESS_ACCESS_T Access;

    sts = EssListLoginsEx(hCtx, &count, &logins);
    printf("EssListLogins sts: %ld\n", sts);
    if(!sts)
    {
        printf("\nConnection count(s): %d\n", count);
        for(i = 0; i < count; i++)
        {
            DisplayLoginInfo(logins[i]);
        }

        sts = EssFree (hInst, logins);
        printf("EssFree sts: %ld\n", sts);
    }

    return(sts);
}

See Also
● EssAutoLogin
● EssInit
● EssListDatabases
● EssLogout
● EssSetActive

EssListObjects
Lists all objects of the specified types on the server or locally on the client.
Syntax

`ESS_FUNC_M EssListObjects (hCtx, ObjType, AppName, DbName, pCount, ppObjList);`

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle. Can be local context handle returned by EssCreateLocalContext().</td>
</tr>
<tr>
<td>ObjType</td>
<td>ESS_OBJTYPE_T</td>
<td>Object type (may be multiple types joined by bitwise OR (</td>
</tr>
<tr>
<td>AppName</td>
<td>ESS_STR_T</td>
<td>Application name.</td>
</tr>
<tr>
<td>DbName</td>
<td>ESS_STR_T</td>
<td>Databasename. If NULL, lists objects in the application subdirectory.</td>
</tr>
<tr>
<td>pCount</td>
<td>ESS_PUSHORT_T</td>
<td>Address of variable to receive the count of objects of the appropriate type(s).</td>
</tr>
<tr>
<td>ppObjList</td>
<td>“ESS_OBJINFO_T” on page 152</td>
<td>Address of pointer to receive allocated array of object info structures.</td>
</tr>
</tbody>
</table>

Notes

- The memory allocated for `ppObjList` should be freed using `EssFree()`.
- This function does not guarantee a consistent order of the returned objects, as this may vary by operating system.

Return Value

If successful, returns a count of the number of objects of the appropriate type(s) in `pCount`, and an array of matching object structures in `ppObjList`.

Access

This function requires no special privileges; note however that server objects will only be listed if the caller has the appropriate level of access to the application and/or database (depending on the object type).

Example

```c
ESS_FUNC_M
ESS_ListObjects (ESS_HCTX_T hCtx,
                  ESS_HINST_T hInst)
{
    ESS_FUNC_M         sts = ESS_STS_NOERR;
    ESS_POBJINFO_T    pObject, pNextObject = NULL;
    ESS_SHORT_T       objType = 0;
    ESS_USHORT_T      objCnt;
    ESS_USHORT_T      objInd;
    ESS_STR_T         AppName;
    ESS_STR_T         DbName;
    Appname = "Sample";
    DbName = "Basic";
    objType = ESS_OBJTYPE_OUTLINE;
    sts = EssListObjects (hCtx, objType, AppName,
                          DbName, &objCnt, &pObject);
    // Further processing...
}
```
if (!sts)
{
    if (objCnt && pObject)
    {
        pNextObject = pObject;
        for (objInd = 0; objInd < objCnt; objInd++)
        {
            if (pNextObject)
            {
                printf("Name: %s \nUser: %s \nTime Stamp: %ld\n",
                        pNextObject->Name,
                        pNextObject->User,
                        pNextObject->TimeStamp);
                pNextObject = pNextObject + 1;
            }
        }
        EssFree (hInst, pObject);
    }
    else
        printf("\n\nObject List is Empty\n\n\n");
    return(sts);
}

See Also

- EssGetObject
- EssGetObjectInfo

**EssListRequests**

Returns information about active sessions and requests.

**Syntax**

```c
ESS_FUNC_M EssListRequests (hCtx, UserName, AppName, DbName, RequestCount, pRequestInfo);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>UserName</td>
<td>ESS_STR_T</td>
<td>User name</td>
</tr>
<tr>
<td>AppName</td>
<td>ESS_STR_T</td>
<td>Application name</td>
</tr>
<tr>
<td>DbName</td>
<td>ESS_STR_T</td>
<td>Database name</td>
</tr>
<tr>
<td>RequestCount</td>
<td>ESS_PUSHORT_T</td>
<td>Number of requests (output)</td>
</tr>
<tr>
<td>ppRequestInfoStruct</td>
<td>ESS_PPREQUESTINFO_T</td>
<td>Request type (output)</td>
</tr>
</tbody>
</table>

**Notes**

- A session is the time in seconds between a user’s login and logout.
A request is a query sent to Essbase by a user or by another process; for example, starting an application or restructuring a database outline. Each session can process only one request at a time; therefore, sessions and requests have a one-to-one relationship.

Some of the listed requests may have been recently terminated, but are still listed as active due to network delay.

This function returns information on requests/sessions initiated by the process specified by the UserName, AppName, and DbName. If these parameters are null or empty, then all the processes in the system are listed. This function returns the number of current requests and one ESS_REQUESTINFO_T structure for each request.

The returned ppRequestInfoStruct needs to be freed by calling EssFree.

Return Value

Returns zero if successful; error code if unsuccessful.

Example

```c
#include <stdio.h>
#include <essapi.h>

ESS_FUNC_M ESS_ListRequest ()
{
    ESS_FUNC_M       sts     = ESS_STS_NOERR;
    ESS_STR_T     rString = NULL;
    ESS_HCTX_T    hCtx;
    ESS_USHORT_T   Items;
    ESS_PAPPD_T    pAppsDbs = NULL;
    ESS_HINST_T    hInst;
    ESS_ACCESS_T   Access;
    ESS_USHORT_T   numRequest;
    ESS_PREQUESTINFO_T requestInfo;

    ESS_INIT_T InitStruct =        /* Define init */
                          /* structure */
    {
        ESS_API_VERSION,       /* Version of API */
        NULL,      /* user-defined message context */
        0,         /* max handles */
        0L,        /* max buffer size */
        NULL, /* local path */
        /* The following parameters use defaults */
        NULL, /* message db path */
        NULL, /* allocation function pointer */
        NULL, /* reallocation function pointer */
        NULL, /* free function pointer */
        NULL, /* error handling function pointer */
        NULL, /* path name of user-defined */
        /* Application help file */
        NULL, /* Reserved for internal use. */
        /* Set to NULL */
    };

    EssInit (&InitStruct, &hInst);

    sts = EssLogin (hInst, "local", "admin", "password", &Items, &pAppsDbs, &hCtx);
```
sts = EssListRequests( hCtx, NULL, NULL, NULL, &numRequest, &requestInfo);

printf ( "Total requests on the server %d\n", numRequest );

if ( !sts && requestInfo )
{
    ESS_USHORT_T index = 0;

    while ( index < numRequest )
    {
        printf ( "login ID = %ul\n", requestInfo[index].LoginId );
        printf ( "user name = %s\n", requestInfo[index].UserName );
        printf ( "login machine = %s\n", requestInfo[index].LoginSourceMachine );
        printf ( "AppName = %s\n", requestInfo[index].AppName );
        printf ( "DbName = %s\n", requestInfo[index].DbName );
        printf ( "DbRequestCode = %u\n", requestInfo[index].DbRequestCode );
        printf ( "RequestString = %s\n", requestInfo[index].RequestString );
        printf ( "TimeStarted = %ul\n", requestInfo[index].TimeStarted );
        printf ( "State = %d\n", requestInfo[index].State );

        requestInfo[index].State = 0;

        sts = EssKillRequest (hCtx, &requestInfo[index] );

        index++;
    }

    EssFree ( hInst, requestInfo );
}

EssLogout (hCtx);
EssTerm (hInst);
return(sts);

void main()
{
    ESS_ListRequest ();
}

See Also

- EssListRequestsEx
- EssKillRequest
- “ESS_REQUESTINFO_T” on page 179
- “ESS_REQ_STATE_T” on page 185

**EssListRequestsEx**

Returns information about active sessions and requests. Similar to EssListRequests, but includes users hosted in a user directory.
Syntax

ESS_FUNC_M EssListRequestsEx (hCtx, UserId, bIsIdentity,AppName, DbName, RequestCount, pRequestInfo);

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle (input).</td>
</tr>
<tr>
<td>UserId</td>
<td>ESS_STR_T</td>
<td>User name or identity (input). If an identity, includes a unique identity</td>
</tr>
<tr>
<td></td>
<td></td>
<td>string identifying the user in a user directory.</td>
</tr>
<tr>
<td>bIsIdentity</td>
<td>ESS_BOOL_T</td>
<td>Indicates if a user identity or name is used (input). If TRUE, indicates</td>
</tr>
<tr>
<td></td>
<td></td>
<td>that UserId is a unique identity attribute. If FALSE, UserId is a user name.</td>
</tr>
<tr>
<td>AppName</td>
<td>ESS_STR_T</td>
<td>Application name (input).</td>
</tr>
<tr>
<td>DbName</td>
<td>ESS_STR_T</td>
<td>Database name (input).</td>
</tr>
<tr>
<td>RequestCount</td>
<td>ESS_PUSHORT_T</td>
<td>Number of requests (output).</td>
</tr>
<tr>
<td>pRequestInfo</td>
<td>ESS_PREQUESTINFOEX_T</td>
<td>Request type (output).</td>
</tr>
</tbody>
</table>

Notes

- A session is the time in seconds between a user's login and logout.
- A request is a query sent to Essbase by a user or by another process; for example, starting an application or restructuring a database outline. Each session can process only one request at a time; therefore, sessions and requests have a one-to-one relationship.
- Some of the listed requests may have been recently terminated, but are still listed as active due to network delay.
- This function returns information on requests/sessions initiated by the process specified by the UserId, AppName, and DbName. If these parameters are null or empty, then all the processes in the system are listed. This function returns the number of current requests and one ESS_REQUESTINFOEX_T structure for each request.
- The returned ppRequestInfoStruct needs to be freed by calling EssFree.

Return Value

Returns zero if successful; error code if unsuccessful.

Access

This function requires the caller to have database designer privilege (ESS_PRIV_DBDESIGN) for the specified database.

Example

```c
void ListRequestsEx ()
{
    ESS_STS_T sts = ESS_STS_NOERR;
    ESS_STR_T userId;
    ESS_BOOL_T bIsIdentity;
    ESS_USHORT_T numRequest;
```
ESS_PREQUESTINFOEX_T requestInfo;
ESS_USHORT_T index = 0;

userId = "admin";
bIsIdentity = ESS_FALSE;
sts = EssListRequestsEx(hCtx, userId, bIsIdentity, "Sample", "Sample1", &numRequest, &requestInfo);
printf("\nEssListRequestsEx sts: %ld\n", sts);
printf ( "Total requests on the server: %d\n", numRequest );
if ( !sts && requestInfo )
{
    while ( index < numRequest )
    {
        printf ( "login ID = %ul\n", requestInfo[index].LoginId );
        printf ( "user name = %s\n", requestInfo[index].UserName );
        printf ( "login machine = %s\n", requestInfo[index].LoginSourceMachine );
        printf ( "AppName = %s\n", requestInfo[index].AppName );
        printf ( "DbName = %s\n", requestInfo[index].DbName );
        printf ( "DbRequestCode = %u\n", requestInfo[index].DbRequestCode );
        printf ( "RequestString = %s\n", requestInfo[index].RequestString );
        printf ( "TimeStarted = %ul\n", requestInfo[index].TimeStarted );
        printf ( "State = %d\n", requestInfo[index].State );
        printf ( "\n\n--------------------------------------\n\n", requestInfo[index].State );
        sts = EssKillRequestEx (hCtx, &requestInfo[index] );
        index++;
    }
    EssFree ( hInst, requestInfo );
}

userId = " native://nvid=f0ed2a6d7fb07688:5a342200:1265973105c:-7f46?USER ";
bIsIdentity = ESS_TRUE;
sts = EssListRequestsEx(hCtx, userId, bIsIdentity, "Sample", "Sample1", &numRequest, &requestInfo);
printf("\nEssListRequestsEx sts: %ld\n", sts);
printf ( "Total requests on the server: %d\n", numRequest );
if ( !sts && requestInfo )
{
    while ( index < numRequest )
    {
        printf ( "login ID = %ul\n", requestInfo[index].LoginId );
        printf ( "user name = %s\n", requestInfo[index].UserName );
        printf ( "login machine = %s\n", requestInfo[index].LoginSourceMachine );
        printf ( "AppName = %s\n", requestInfo[index].AppName );
        printf ( "DbName = %s\n", requestInfo[index].DbName );
        printf ( "DbRequestCode = %u\n", requestInfo[index].DbRequestCode );
        printf ( "RequestString = %s\n", requestInfo[index].RequestString );
        printf ( "TimeStarted = %ul\n", requestInfo[index].TimeStarted );
        printf ( "State = %d\n", requestInfo[index].State );
        printf ( "\n\n--------------------------------------\n\n", requestInfo[index].State );
        sts = EssKillRequestEx (hCtx, &requestInfo[index] );
}
EssListSpoolFiles

Lists all trigger log files for a database.

Syntax

```c
ESS_FUNC_M EssListSpoolFiles (hCtx, AppName, DbName, pCount, ppFileList);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>AppName</td>
<td>ESS_STR_T</td>
<td>Application name.</td>
</tr>
<tr>
<td>DbName</td>
<td>ESS_STR_T</td>
<td>Database name.</td>
</tr>
<tr>
<td>pCount</td>
<td>ESS_PUSHORT_T</td>
<td>Address of variable to receive count of spool file names.</td>
</tr>
<tr>
<td>ppFileList</td>
<td>ESS_PPOBJINFO_T</td>
<td>Address of pointer to receive an allocated array of spool file name objects.</td>
</tr>
</tbody>
</table>

Notes

The memory allocated for `EssListSpoolFiles()` should be freed using `EssFree()`.

Return Value

If successful, returns the count of spool files in the database in `pCount`, and an array of spool file names in `ppFileList`.

Access

This function requires the caller to have database designer privilege (ESS_PRIV_DBDESIGN) for the specified database.

See Also

- EssDisplayTriggers
- EssGetSpoolFile
- EssDeleteAllSplFiles
- EssDeleteSplFile
- EssMdxTrig
**EssListSSMigrFailedGroups**

Displays groups that did not successfully migrate to Shared Services.

**Syntax**

```c
ESS_FUNC_M EssListSSMigrFailedGroups (hCtx, count, pNativeUserList);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>count</td>
<td>ESS_PUSHORT_T</td>
<td>Address of variable to receive count of migration failed groups.</td>
</tr>
<tr>
<td>pNativeUserList</td>
<td>ESS_PPUSERNAME_T</td>
<td>Address of pointer to receive an allocated array of migration failed groups.</td>
</tr>
</tbody>
</table>

**Notes**

If Essbase has not been migrated to Shared Services, this function is not supported and returns an error.

**Return Value**

Returns 0 if successful; otherwise, returns an error.

**Example**

```c
ESS_FUNC_M ESS_SS_ListSSMigrFailedGroups(ESS_HCTX_T hCtx, ESS_HINST_T hInst)
{
    ESS_STS_T         sts = ESS_STS_NOERR;
    ESS_USERNAME_T    pNativeUserList = NULL;
    ESS_USHORT_T      Count = 0,
                      index;

    sts = EssListSSMigrFailedGroups(hCtx, &Count, &pNativeUserList);
    if (!sts)
    {
        if (Count && pNativeUserList)
        {
            printf ("\n------- Group List -------\n\\n");
            for (index = 0; index < Count; index++)
            {  
                if (pNativeUserList[index])
                    printf ("%s\n", pNativeUserList[index]);
            }
            EssFree(hInst, pNativeUserList);
        }
        else
            printf("\nGroup list is empty\n\\n");
    }
    else
        printf("Failed to get Shared Services migration failed Groups list.\n\\n");

    return (sts);
}
```
See also an extended Appendix B

See Also
- EssListSSMigrFailedUsers

**EssListSSMigrFailedUsers**
Displays users that did not successfully migrate to Shared Services.

**Syntax**

```
ESS_FUNC_M EssListSSMigrFailedUsers (hCtx, count, pNativeUserList);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>count</td>
<td>ESS_PUSHORT_T</td>
<td>Address of variable to receive count of migration failed users.</td>
</tr>
<tr>
<td>pNativeUserList</td>
<td>ESS_PPUSERNAME_T</td>
<td>Address of pointer to receive an allocated array of migration failed users.</td>
</tr>
</tbody>
</table>

**Return Value**
Returns 0 if successful; otherwise, returns an error.

**Example**

```
ESS_FUNC_M ESS_SS_ListSSMigrFailedUsers(ESS_HCTX_T hCtx, ESS_HINST_T hInst)
{
    ESS_STS_T         stst = ESS_STS_NOERR;
    ESS_PUSERNAME_T   pNativeUserList = NULL;
    ESS_USHORT_T      Count = 0,
    index;

    stst = EssListSSMigrFailedUsers(hCtx, &Count, &pNativeUserList);
    if (!stst)
    {
        if (Count && pNativeUserList)
        {
            printf ("\n------- User List -------\n\n");

            for (index = 0; index < Count; index++)
            {
                if (pNativeUserList[index])
                    printf ("%s\n", pNativeUserList[index]);
            }

            EssFree(hInst, pNativeUserList);
        }
        else
            printf("\nUser list is empty\n\n");
    }
    else
        printf("Failed to get Shared Services migration failed Users list.\n");
}
```

478  C Main API Functions
return (sts);
}

See also an extended Appendix B

See Also

- EssListSSMigrFailedGroups

## EssListTransactions

Returns transaction messages to a client buffer or to a comma-separated file. You can export comma-separated files to relational databases for processing with third-party tools.

### Syntax

```c
ESS_FUNC_M EssListTransactions(hCtx, TimeSrc, InpTime,
    ListOption, FileName, pCount, ppResults);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>Login context must be set active before calling this API.</td>
</tr>
<tr>
<td>TimeSrc</td>
<td>ESS_USHORT_T</td>
<td>The option that specifies where to get the start time for display transactions.</td>
</tr>
<tr>
<td>InpTime</td>
<td>ESS_TIME32_T</td>
<td>Input the time if TimeSrc is ESS_TRLOG_TIME_SPECIFIED. The time is a ULONG representing the number of seconds since January 1, 1970.</td>
</tr>
<tr>
<td>ListOption</td>
<td>ESS_USHORT_T</td>
<td>The option that specifies the destination of the output. See “List Option Constants (C)” on page 105</td>
</tr>
<tr>
<td>FileName</td>
<td>ESS_STR_T</td>
<td>If ListOption is either of the LIST_TRANSACTIONS_* options, then the content is written to this file on the server machine:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- You can enter a full path.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Default: $ARBORPATH/app</td>
</tr>
<tr>
<td>pCount</td>
<td>ESS_PULONG_T</td>
<td>Number of entries returned</td>
</tr>
<tr>
<td>ppResults</td>
<td>ESS_PPTRANSACTION_ENTRY_T</td>
<td>The entries returned if ListOption is ESS_LIST_TRANSACTIONS_TOCLIENT</td>
</tr>
</tbody>
</table>

### Return Value

- 0—If successful
  - pCount contains the number of returned entries
  - ppResults contains the returned entries if ListOption is ESS_LIST_TRANSACTIONS_TOCLIENT
- Error number—If unsuccessful
**Access**

You must have an active database using `set active` before calling `list transactions`.

The caller must have Essbase Administrator access to the database.

**Example**

```c
void ListAndReplayTransactions()
{
    ESS_FUNC_M                sts = ESS_STS_NOERR;
    ESS_USHORT_T                TimeSrc;
    ESS_TIME32_T                timestamp = 0;
    ESS_USHORT_T                listOption;
    ESS_STR_T                    FileName = ESS_NULL;
    ESS ULONG_T                Count = 0;
    ESS PTRANSACTION_ENTRY_T    pResults;
    ESS CHAR_T                listTime[ESS_TIMESIZE];
    ESS TRANSACTION REPLAY INP_T    ReplayInp;
    ESS PSEQID_T                pSeqIds = ESS_NULL;
    ESS OBJDEF_T                Data;
    ESS_STR_T                    Script;
    ESS SHORT_T                isAbortOnError;
    ESS PMBRERR_T            pMbrErr = NULL;
    ESS PROSTATE_T        pState;

    /* Load data from server */
    Data.hCtx      = hCtx;
    Data.AppName   = AppName;
    Data.DbName    = DbName;
    Data.ObjType   = ESS_OBJTYPE_TEXT;
    Data.FileName  = "Calcdat";
    isAbortOnError = ESS_TRUE;
    sts = EssImport (hCtx, ESS_NULL, &Data,
                    &pMbrErr, NULL, isAbortOnError);
    printf("EssImport sts: %ld\r\n",sts);

    /* List and replay with a specified time */
    TimeSrc = 1;
    strcpy(listTime, "09/18/2007:00:00:00");
    /* mm/dd/yyyy:hh:mm:ss */
    timestamp = adtGenericGetTime(listTime);
    listOption = ESS_LIST_TRANSACTIONS_TOCLIENT;
    sts = EssListTransactions(hCtx, TimeSrc,
                              timestamp, listOption,
                              FileName, &Count, &pResults);

    /* This function converts listTime to the number of
       seconds since January 1, 1970. */

    printf("EssListTransactions sts: %ld\r\n",sts);
    if (Count && pResults)
        PrintTransactionLog(Count, pResults);
    memset(&ReplayInp, 0, sizeof((ESS_TRANSACTION REPLAY INP_T)));
    ReplayInp.InpType = ESS_REPLAY_BASED_GIVENTIME;
    ReplayInp.value.InpTime = timestamp;
}
```

480  **C Main API Functions**
sts = EssReplayTransactions (hCtx, AppName, DbName, 
            ReplayInp, pSeqIds);
printf("EssReplayTransactions sts: %ld\n",sts);
printf("\n\n");

/* Run a calc*/
Script = "CALC ALL;";
sts = EssCalc(hCtx, ESS_TRUE, Script);
printf("EssCalc sts: %ld\n",sts);
if (!sts)
{
    sts = EssGetProcessState (hCtx, &pState);
    while (!sts && (pState.State != ESS_STATE_DONE))
        sts = EssGetProcessState (hCtx, &pState);
}

/* List and replay with last replay time */
TimeSrc = 2;
timestamp = 0;
sts = EssListTransactions(hCtx, TimeSrc, 
                timestamp, listOption, 
                FileName, &Count, &pResults);

/* This function converts listTime to the number of 
seconds since January 1, 1970. */
printf("EssListTransactions sts: %ld\n",sts);
if (Count && pResults)
    PrintTransactionLog(Count, pResults);
memset(&ReplayInp, 0, sizeof 
        (ESS_TRANSACTION_REPLAY_INP_T));
ReplayInp.InpType = ESS_REPLAY_BASED_LASTREPLAYTIME;
sts = EssReplayTransactions (hCtx, AppName, 
            DbName, ReplayInp, pSeqIds);
printf("EssReplayTransactions sts: %ld\n",sts);

if(pSeqIds)
    EssFree(hInst, pSeqIds);
if(pResults)
    EssFree(hInst, pResults);
if(pMbrErr)
    EssFree(hInst, pMbrErr);
}

See Also

- “ESS_SEQID_T” on page 186
- “ESS_DISKVOLUME_REPLACE_T” on page 137
- “ESS_TRANSACTION_ENTRY_T” on page 187
- “ESS_TRANSACTION_REPLAY_INP_T” on page 188
- “ESS_TRANSACTION_REQSPECIFIC_T” on page 188
- EssReplayTransactions

EssListUsers

Lists all users who have access to a particular Essbase Server, application or database.
Syntax

ESS_FUNC_M EssListUsers (hCtx,AppName,DbName,pCount,ppUserList);

Parameter  Data Type  Description
hCtx  ESS_HCTX_T  API context handle.
AppName  ESS_STR_T  Application name. If NULL, lists all users.
DbName  ESS_STR_T  Database name. If NULL, lists users for all databases within the application.
pCount  ESS_PUSHORT_T  Address of variable to receive count of users.

ppUserList  “ESS_USERINFO_T,
ESS_GROUPINFO_T” on page 192  Address of pointer to receive an allocated array of user info structures. The AppName and DbName fields of the returned user info structures contain NULL values.

Notes

- If both AppName and DbName are not NULL, only users with access to the specified application and database are listed. If DbName is NULL, only users with access to the specified application are listed. If AppName is NULL, all users that exist on the server are listed.

- The AppName and DbName fields of the returned ESS_USERINFO_T structures contain NULL values.

- The memory allocated for ppUserList should be freed using EssFree().

Return Value

If successful, returns a count of the number of users in pCount, and list of users with access to the specified application and database in ppUserList.

Access

This function requires no special privileges.

Example

ESS_STS_T
ESS_ListUsers (ESS_HCTX_T hCtx,
ESS_HINST_T hInst
)
{
    ESS_STS_T  st;
    ESS_USHORT_T  Count;
    ESS_PUSERINFO_T  Users = NULL;
    ESS_USHORT_T  ind;

    st = EssListUsers (hCtx, NULL, NULL, &Count,
                       &Users);
    if (!st)
    {
        if (Count & Users)
        {
            printf ("\n\n-------User List-------\n\n\n");
        }
for (ind = 0; ind < Count; ind++)
{
    printf ("Name->%s Application->%s database->%s\r\n",
            Users[ind].Name, Users[ind].AppName,
            Users[ind].DbName);
    }
    EssFree (hInst, Users);
}  
else
printf ("\r\nUsers list is empty\r\n\r\n");
}
return (sts);
}

See Also
- EssListUsersInfoEx
- EssGetUser
- EssListConnections
- EssListGroups
- EssListLocks

## EssListUsersEx

Lists all users who have access to a particular Essbase Server, application or database.

### Syntax

```
ESS_FUNC_M EssListUsersEx (hCtx,AppName,DbName,SecurityProvider,pCount, ppUserList);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>AppName</td>
<td>ESS_STR_T</td>
<td>Application name. If NULL, lists all users.</td>
</tr>
<tr>
<td>DbName</td>
<td>ESS_STR_T</td>
<td>Database name. If NULL, lists users for all databases within application.</td>
</tr>
<tr>
<td>SecurityProvider</td>
<td>ESS_STR_T</td>
<td>Name of the external authentication mechanism.</td>
</tr>
<tr>
<td>pCount</td>
<td>ESS_PUSHORT_T</td>
<td>Address of variable to receive count of users.</td>
</tr>
<tr>
<td>ppUserList</td>
<td>ESS_USERINFOEX_T</td>
<td>Address of pointer to receive an allocated array of user info structures. The AppName and DbName fields of the returned user info structures contain NULL values.</td>
</tr>
</tbody>
</table>

### Notes

- If both AppName and DbName are not NULL, only users with access to the specified application and database are listed. If DbName is NULL, only users with access to the specified application are listed. If AppName is NULL, all users that exist on the server are listed.
- The AppName and DbName fields of the returned ESS_USERINFO_T structures contain NULL values.
The memory allocated for *ppUserList* should be freed using *EssFree()*.

**Return Value**

If successful, returns a count of the number of users in *pCount*, and list of users with access to the specified application and database in *ppUserList*.

**Access**

This function requires no special privileges.

**Example**

```c
ESS_STS_T
ESS_ListUsers (ESS_HCTX_T  hCtx,
                ESS_HINST_T hInst
             )
{
    ESS_STS_T        sts;
    ESS_USHORT_T     Count;
    ESS_PUSERINFO_T  Users = NULL;
    ESS_USHORT_T     ind;

    sts = EssListUsersEx (hCtx, NULL, NULL, &Count,
                          &Users);
    if (!sts)
    {
        if (Count & Users)
        {
            printf ("\r
-------User List-------\r\n\r
" );
            for (ind = 0; ind < Count; ind++)
            {
                printf ("Name->%s Application->%s database->%s\r\n",
                        Users[ind].Name, Users[ind].AppName,
                        Users[ind].DbName);
            }
            EssFree (hInst, Users);
        }
        else
        printf ("\r
Users list is empty\r\n\r
"");
    }
    return (sts);
}
```

**See Also**

- *EssGetUser*
- *EssListConnections*
- *EssListGroup*
- *EssListLocks*
- *EssCreateExtUser*
- *EssGetUserEx*
- *EssSetUserEx*
- "ESS_USERINFOEX_T" on page 195
**EssListUsersInfoEx**

Lists all users who have access to a particular Essbase Server, application or database. Similar to *EssListUsers*, but the user list structure can include user directories and unique identity attributes.

**Syntax**

```
ESS_FUNC_M EssListUsersInfoEx (hCtx,AppName,DbName,pCount,ppUserList);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle (input).</td>
</tr>
<tr>
<td>AppName</td>
<td>ESS_STR_T</td>
<td>Application name (input). If NULL, lists all users.</td>
</tr>
<tr>
<td>DbName</td>
<td>ESS_STR_T</td>
<td>Database name (input). If NULL, lists users for all databases within the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>application.</td>
</tr>
<tr>
<td>pCount</td>
<td>ESS_PUSHORT_T</td>
<td>Address of variable to receive count of users (output).</td>
</tr>
<tr>
<td>ppUserList</td>
<td>ESS_PPUSERINFOID_T</td>
<td>Address of pointer to receive an allocated array of user info structures (output).</td>
</tr>
</tbody>
</table>

**Return Value**

If successful, returns a count of the number of users in `pCount`, and a list of users with access to the specified application and database in `ppUserList`.

**Access**

This function requires no special privileges.

**Example**

```c
void DisplayUserInfoID2(ESS_USERINFOID_T userInfo)
{
    ESS_STS_T sts = ESS_STS_NOERR;
    ESS_BOOL_T isDefined = ESS_TRUE;

    printf("\tUser Name: %s\n", userInfo.Name);
    printf("\tProvider Name: %s\n", userInfo.ProviderName);
    printf("\tConnparam: %s\n", userInfo.connparam);
    printf("\tDescription: %s\n", userInfo.Description);
    printf("\tEmail Identification: %s\n", userInfo.EMailID);

    if (userInfo.LockedOut)
        printf("\tLocked out: Yes\n");
    else
        printf("\tLocked out: No\n");

    if (userInfo.PwdChgNow)
        printf("\tChange the password now: Yes\n");
    else
        printf("\tChange the password now: No\n");

    printf("\tConnected Application: %s\n", userInfo.AppName);
}
```
printf("\tConnected Database: %s\n", userInfo.DbName);

if (userInfo.Login)
    printf("\tLogged in: Yes\n");
else
    printf("\tLogged in: No\n");

switch(userInfo.Access)
{
    case ESS_ACCESS_ADMIN:
        printf("\tAccess: %d - ESS_ACCESS_ADMIN\n", userInfo.Access);
        break;
    case ESS_ACCESS_APPALL:
        printf("\tAccess: %d - ESS_ACCESS_APPALL\n", userInfo.Access);
        break;
    case ESS_ACCESS_DBALL:
        printf("\tAccess: %d - ESS_ACCESS_DBALL\n", userInfo.Access);
        break;
    case ESS_ACCESS_APPCREATE:
        printf("\tAccess: %d - ESS_ACCESS_APPCREATE\n", userInfo.Access);
        break;
    case ESS_ACCESS_APPMANAGE:
        printf("\tAccess: %d - ESS_ACCESS_APPMANAGE\n", userInfo.Access);
        break;
    case ESS_ACCESS_DBCREATE:
        printf("\tAccess: %d - ESS_ACCESS_DBCREATE\n", userInfo.Access);
        break;
    case ESS_ACCESS_DBMANAGE:
        printf("\tAccess: %d - ESS_ACCESS_DBMANAGE\n", userInfo.Access);
        break;
    case ESS_ACCESS_CALC:
        printf("\tAccess: %d - ESS_ACCESS_CALC\n", userInfo.Access);
        break;
    case ESS_ACCESS_WRITE:
        printf("\tAccess: %d - ESS_ACCESS_WRITE\n", userInfo.Access);
        break;
    case ESS_ACCESS_READ:
        printf("\tAccess: %d - ESS_ACCESS_READ\n", userInfo.Access);
        break;
    case ESS_PRIV_USERCREATE:
        printf("\tAccess: %d - ESS_PRIV_USERCREATE\n", userInfo.Access);
        break;
    case ESS_PRIV_APPCREATE:
        printf("\tAccess: %d - ESS_PRIV_APPCREATE\n", userInfo.Access);
        break;
    case ESS_PRIV_APPMANAGE:
        printf("\tAccess: %d - ESS_PRIV_APPMANAGE\n", userInfo.Access);
        break;
    case ESS_PRIV_APPLOAD:
        printf("\tAccess: %d - ESS_PRIV_APPLOAD\n", userInfo.Access);
        break;
    case ESS_PRIV_DBCREATE:
        printf("\tAccess: %d - ESS_PRIV_DBCREATE\n", userInfo.Access);
        break;
    case ESS_PRIV_DBMANAGE:
        printf("\tAccess: %d - ESS_PRIV_DBMANAGE\n", userInfo.Access);
        break;
}
case ESS_PRIV_DBLOAD:
    printf("\tAccess: %d - ESS_PRIV_DBLOAD\n", userInfo.Access);
    break;

case ESS_PRIV_CALC:
    printf("\tAccess: %d - ESS_ACCESS_DBALL\n", userInfo.Access);
    break;

case ESS_PRIV_WRITE:
    printf("\tAccess: %d - ESS_PRIV_WRITE\n", userInfo.Access);
    break;

case ESS_PRIV_READ:
    printf("\tAccess: %d - ESS_PRIV_READ\n", userInfo.Access);
    break;

case ESS_PRIV_NONE:
    printf("\tAccess: %d - ESS_PRIV_NONE\n", userInfo.Access);
    break;

default:
    printf("\tAccess: Unknown\n");
}

switch(userInfo.MaxAccess)
{
    case ESS_ACCESS_ADMIN:
        printf("\tMax Access: %d - ESS_ACCESS_ADMIN\n", userInfo.MaxAccess);
        break;

    case ESS_ACCESS_APPALL:
        printf("\tMax Access: %d - ESS_ACCESS_APPALL\n", userInfo.MaxAccess);
        break;

    case ESS_ACCESS_DBALL:
        printf("\tMax Access: %d - ESS_ACCESS_DBALL\n", userInfo.MaxAccess);
        break;

    case ESS_ACCESS_APPCREATE:
        printf("\tMax Access: %d - ESS_ACCESS_APPCREATE\n", userInfo.MaxAccess);
        break;

    case ESS_ACCESS_APPMANAGE:
        printf("\tMax Access: %d - ESS_ACCESS_APPMANAGE\n", userInfo.MaxAccess);
        break;

    case ESS_ACCESS_DBCREATE:
        printf("\tMax Access: %d - ESS_ACCESS_DBCREATE\n", userInfo.MaxAccess);
        break;

    case ESS_ACCESS_DBMANAGE:
        printf("\tMax Access: %d - ESS_ACCESS_DBMANAGE\n", userInfo.MaxAccess);
        break;

    case ESS_ACCESS_CALC:
        printf("\tMax Access: %d - ESS_ACCESS_CALC\n", userInfo.MaxAccess);
        break;

    case ESS_ACCESS_WRITE:
        printf("\tMax Access: %d - ESS_ACCESS_WRITE\n", userInfo.MaxAccess);
        break;

    case ESS_ACCESS_READ:
        printf("\tMax Access: %d - ESS_ACCESS_READ\n", userInfo.MaxAccess);
        break;

    case ESS_PRIV_USERCREATE:
        printf("\tMax Access: %d - ESS_PRIV_USERCREATE\n", userInfo.MaxAccess);
        break;

    case ESS_PRIV_APPCREATE:
        printf("\tMax Access: %d - ESS_PRIV_APPCREATE\n", userInfo.MaxAccess);
        break;
case ESS_PRIV_APPMANAGE:
    printf("\tMax Access: %d - ESS_PRIV_APPMANAGE\n", userInfo.MaxAccess);
    break;
case ESS_PRIV_APPLOAD:
    printf("\tMax Access: %d - ESS_PRIV_APPLOAD\n", userInfo.MaxAccess);
    break;
case ESS_PRIV_DBCREATE:
    printf("\tMax Access: %d - ESS_PRIV_DBCREATE\n", userInfo.MaxAccess);
    break;
case ESS_PRIV_DBMANAGE:
    printf("\tMax Access: %d - ESS_PRIV_DBMANAGE\n", userInfo.MaxAccess);
    break;
case ESS_PRIV_DBLOAD:
    printf("\tMax Access: %d - ESS_PRIV_DBLOAD\n", userInfo.MaxAccess);
    break;
case ESS_PRIV_CALC:
    printf("\tMax Access: %d - ESS_ACCESS_DBALL\n", userInfo.MaxAccess);
    break;
case ESS_PRIV_WRITE:
    printf("\tMax Access: %d - ESS_PRIV_WRITE\n", userInfo.MaxAccess);
    break;
case ESS_PRIV_READ:
    printf("\tMax Access: %d - ESS_PRIV_READ\n", userInfo.MaxAccess);
    break;
case ESS_PRIV_NONE:
    printf("\tMax Access: %d - ESS_PRIV_NONE\n", userInfo.MaxAccess);
    break;
default:
    printf("\tMax Access: Unknown\n");
}

printf("\tPassword Expiration in Dates: %d\n", userInfo.Expiration);
if(!sts)
    printf("\tNumber of users: %d\n\n", count);
for(i = 0; i < count; i++)
    DisplayUserInfoID2(pUserList[i]);

ESS_STS_T ESS_ListUsersInfo (ESS_HCTX_T hCtx, ESS_HINST_T hInst)
{
    ESS_STS_T sts = ESS_STS_NOERR;
    ESS_USHORT_T count, i;
    ESS_PUSERINFOID_T pUserList;

    sts = EssListUsersInfoEx(hCtx, AppName, ",", &count, &pUserList);
    printf("EssListUsersInfoEx sts: %ld\n", sts);
    if(!sts)
    {
        printf("\tNumber of users: %d\n\n", count);
        for(i = 0; i < count; i++)
            DisplayUserInfoID2(pUserList[i]);
    }
return (sts);
}

See Also

- EssCreateExtUser
- EssListGroupsInfoEx

## EssListVariables

EssListVariables() lists substitution variables at the server, application, and database levels, according to the input criteria.

### Syntax

```c
ESS_FUNC_M EssListVariables (hCtx, pCriteria, pNumVars, ppVarList);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>Context handle to the API.</td>
</tr>
<tr>
<td>pCriteria</td>
<td>&quot;ESS_VARIABLE_T&quot; on page 197</td>
<td>The pointer to the structure containing the description of the substitution variables being listed.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The members VarName and VarValue are ignored.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- If the Server member is given but AppName and DbName are empty, then the function will list substitution variables at the server level only.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- If the Server and AppName members are given but DbName is empty, it lists all variables at both the given server and application levels.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- If all three of Server, AppName, and DbName members are given, it lists variables from all three specified levels.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- If a field is empty, then that field is treated as a &quot;don't care.&quot;</td>
</tr>
<tr>
<td>pNumVars</td>
<td>ESS_PULONG_T</td>
<td>The pointer to an unsigned long value indicating the number of variables being returned in the ppVarList parameter.</td>
</tr>
<tr>
<td>ppVarList</td>
<td>&quot;ESS_VARIABLE_T&quot; on page 197</td>
<td>The pointer to an array of substitution variable structures. It is the responsibility of the caller to free this array by calling EssFree.</td>
</tr>
</tbody>
</table>

### Return Value

If successful, returns zero.

### Example

```c
/*
** ESS_ListVariables() lists the substitution variables using
** the API EssListVariables.
*/
ESS_FUNC_M
ESS_ListVariables (ESS_HCTX_T hCtx)
{

    ESS_FUNC_M    sts = ESS_STS_NOERR;
    ESS_VARIABLE_T pVariables;
    ESSULONG_T    ulCount, i;
```
ESS_VARIABLE_T Variable;
printf("\n *****************************************");
printf("\n **** An example of using EssListVariables\n");
printf("\n *****************************************");
/*------------------------------------------------------------------*/
/* List Variables at the level of the Server/App/Db                */
/* Variables under that specific server will be listed            */
/* Variables under that specific server/ App will be listed       */
/* Variables under that specific server/ App /DB will be listed    */
/*------------------------------------------------------------------*/
strcpy(Variable.VarName,  "");   // ignored by EssListVariables
strcpy(Variable.Server,  "local");
strcpy(Variable.AppName, "Sample");
strcpy(Variable.DbName,  "Basic");
sts = EssListVariables(hCtx, &Variable, &ulCount, &pVariables);
if (sts == ESS_STS_NOERR)
{
    printf("\n--- Number of Substitution Variables at the Server, App and Db
    level is: %ld\n", ulCount);
    for (i = 0; i < ulCount; i++)
    {
        printf("Variable name : %s\n", pVariables[i].VarName);
        printf("Server name : %s\n", pVariables[i].Server);
        printf("Application name : %s\n", pVariables[i].AppName);
        printf("Database name : %s\n", pVariables[i].DbName);
        printf("Variable value : %s\n\n", pVariables[i].VarValue);
    }
}
/*------------------------------------------------------------------*/
/* Variables under that specific Server will be listed            */
/* Variables under that specific Server/App will be listed        */
/*------------------------------------------------------------------*/
if (sts == ESS_STS_NOERR)
{
    strcpy(Variable.VarName,  "");   // ignored by EssListVariables
    strcpy(Variable.Server,  "local");
    strcpy(Variable.AppName, "Sample");
    strcpy(Variable.DbName,  "");
    sts = EssListVariables(hCtx, &Variable, &ulCount, &pVariables);
    if (sts == ESS_STS_NOERR)
    {
        printf("\n--- Number of Substitution Variables at the Server and App
        level is: %ld\n", ulCount);
        for (i = 0; i < ulCount; i++)
        {
            printf("Variable name : %s\n", pVariables[i].VarName);
            printf("Server name : %s\n", pVariables[i].Server);
            printf("Application name : %s\n", pVariables[i].AppName);
            printf("Database name : %s\n", pVariables[i].DbName);
            printf("Variable value : %s\n\n", pVariables[i].VarValue);
        }
    }
}
/*------------------------------------------------------------------*/
/* List Variables at the level of the Server                      */
/*------------------------------------------------------------------*/
if (sts == ESS_STS_NOERR)
{
    strcpy(Variable.VarName, ""); // ignored by EssListVariables
    strcpy(Variable.Server, "local");
    strcpy(Variable.AppName, "");
    strcpy(Variable.DbName, "");
    if (sts == ESS_STS_NOERR)
        sts = EssListVariables(hCtx, &Variable, &ulCount, &pVariables);
    if (sts == ESS_STS_NOERR)
    {
        printf("\n--- Number of Substitution Variables at the Server level is:
%ld\n", ulCount);
        for (i = 0; i < ulCount; i++)
        {
            printf("Variable name    : %s\n",   pVariables[i].VarName);
            printf("Server name      : %s\n",   pVariables[i].Server);
            printf("Application name : %s\n",   pVariables[i].AppName);
            printf("Database name    : %s\n",   pVariables[i].DbName);
            printf("Variable value   : %s\n\n", pVariables[i].VarValue);
        }
    }
}
if (sts == ESS_STS_NOERR)
    printf("\n --> No Errors in EssListVariables


");
else
    printf("\n --> Error in EssListVariables number: %d\n\n
", sts);
return (sts);
} /* end ESS_ListVariables */

Output

*************************************************************
**** An example of using EssListVariables
*************************************************************
--- Number of Substitution Variables at the Server, App and Db level is: 3
  Variable name    : QuarterName
  Server name      : local
  Application name : Sample
  Database name    : Basic
  Variable value   : Qtr2
  Variable name    : MarketName
  Server name      : local
  Application name : Sample
  Database name    :
  Variable value   : East
  Variable name    : MarketName
  Server name      : local
  Application name :
  Database name    :
  Variable value   : Market

--- Number of Substitution Variables at the Server and App level is: 2
  Variable name    : MarketName
  Server name      : local
  Application name :
  Database name    :
Variable value : East
Variable name : MarketName
Server name : local
Application name :
Database name :
Variable value : Market

--- Number of Substitution Variables at the Server level is: 1
Variable name : MarketName
Server name : local
Application name :
Database name :
Variable value : Market

--> No Errors in EssListVariables

See Also

- “ESS_VARIABLE_T” on page 197
- EssCreateVariable
- EssDeleteVariable
- EssGetVariable

**EssLoadAlias**

Creates and permanently loads an alias table for the active database from a structured text file.

**Syntax**

```c
ESS_FUNC_M EssLoadAlias (hCtx, AliasName, FileName);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>AliasName</td>
<td>ESS_STR_T</td>
<td>Name of alias table to load.</td>
</tr>
<tr>
<td>FileName</td>
<td>ESS_STR_T</td>
<td>Full path name of structured alias names file on the server.</td>
</tr>
</tbody>
</table>

**Notes**

- This function cannot complete successfully if `AliasName` already exists. Before you can load an alias table with the same name as an existing table, you must delete the existing alias table.
- The alias table file format is described in the *Oracle Essbase Database Administrator’s Guide*.

**Return Value**

None.

**Access**

This function requires the caller to have at least read access (ESS_PRIV_READ) to the database, and to have selected it as their active database using `EssSetActive()`.
Example

ESS_FUNC_M
ESS_LoadAlias (ESS_HCTX_T hCtx)
{
    ESS_FUNC_M sts = ESS_STS_NOERR;
    ESS_STR_T TableName;
    ESS_STR_T FileName;
    TableName = "NewAlias";
    FileName = "NEW.ALT'';
    sts = EssLoadAlias (hCtx, TableName, FileName);
    return (sts);
}

See Also

- EssListAliases
- EssSetActive

### EssLoadApplication

Starts an application on the server.

**Syntax**

ESS_FUNC_M EssLoadApplication (hCtx, AppName);

**Parameter** | **Data Type** | **Description**
--- | --- | ---
| hCtx | ESS_HCTX_T | API context handle. |
| AppName | ESS_STR_T | Name of application to load. |

**Notes**

To load an application, the connected user must have load access to the application.

**Return Value**

None.

**Access**

This function requires the caller to have Application Load/Unload privilege (ESS_PRIV_APPLOAD) for the specified application.

**Example**

ESS_FUNC_M
ESS_LoadApp (ESS_HCTX_T hCtx)
{
    ESS_FUNC_M sts = ESS_STS_NOERR;
    ESS_STR_T AppName;
    AppName = "Sample";
    sts = EssLoadApplication (hCtx, AppName);
return (sts);
}

See Also

- `EssLoadDatabase`
- `EssUnloadApplication`

## EssLoadBufferInit

Creates a temporary data load buffer, which provides temporary storage for tuples during a data load into an aggregate storage database. Applies only to aggregate storage databases.

### Syntax

```c
ESS_FUNC_M EssLoadBufferInit (hCtx, AppName, DbName, ulBufferId,
ulDuplicateAggregationMethod, ulOptionFlags, ulSize);
```

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>AppName</td>
<td>ESS_STR_T</td>
<td>Name of the application for which to create the load buffer.</td>
</tr>
<tr>
<td>DbName</td>
<td>ESS_STR_T</td>
<td>Name of the database for which to create the load buffer.</td>
</tr>
<tr>
<td>ulBufferId</td>
<td>ESS_ULONG_T</td>
<td>ID number for the data load buffer (a number between 1 and 999,999, inclusive). If the ID is already in use, the operation fails.</td>
</tr>
</tbody>
</table>
**Parameter** | **Data Type** | **Description**
---|---|---
ulDuplicateAggregationMethod | ESS_ULONGLONG | One of the following constants for combining multiple values for the same cell within the buffer:

- ESS_ASO_DATA_LOAD_BUFFER_DUPLICATES_ADD: Add values when the buffer contains multiple values for the same cell.
  
  #define ESS_ASO_DATA_LOAD_BUFFER_DUPLICATES_ADD 0

- ESS_ASO_DATA_LOAD_BUFFER_DUPLICATES_ASSUME_EQUAL: Verify that multiple values for the same cells are identical; if they are, ignore the duplicate values. If the values for the same cell differ, stop the data load with an error message.
  
  #define ESS_ASO_DATA_LOAD_BUFFER_DUPLICATES_ASSUME_EQUAL 1

- ESS_ASO_DATA_LOAD_BUFFER_DUPLICATES_USE_LAST: Combines duplicate cells by using the value of the cell that was loaded last into the load buffer. This option is intended for relatively small data loads of up to 10,000s of cells.
  
  #define ESS_ASO_DATA_LOAD_BUFFER_DUPLICATES_USE_LAST 2

When using data load buffers with the use_last option, data loads are significantly slower, even if there are not any duplicate values.

**Caution!** The use_last method has significant performance impact, and is not intended for large data loads. If your data load is larger than one million cells, consider separating the numeric data into a separate data load process (from any typed measure data). The separate data load can use the add method instead.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ulOptionFlags</td>
<td>ESS_ULONG_T</td>
<td>One or more of the following load buffer options:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <strong>ESS_ASO_DATA_LOAD_BUFFER_IGNORE_MISSING_VALUES</strong>: Ignores <code>#MISSING</code> values in the incoming data stream.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>#define <strong>ESS_ASO_DATA_LOAD_BUFFER_IGNORE_MISSING_VALUES</strong> 0x00000001</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <strong>ESS_ASO_DATA_LOAD_BUFFER_IGNORE_ZERO_VALUES</strong>: Ignores zeros in the incoming data stream.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>#define <strong>ESS_ASO_DATA_LOAD_BUFFER_IGNORE_ZERO_VALUES</strong> 0x00000002</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <strong>ESS_ASO_DATA_LOAD_BUFFER_WAIT_FOR_RESOURCES</strong>: Tells Essbase to wait up to the amount of time specified by the ASOLOADBUFFERWAIT configuration setting in essbase.cfg for resources to become available in order to process load buffer operations.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>#define <strong>ESS_ASO_DATA_LOAD_BUFFER_WAIT_FOR_RESOURCES</strong> 0x00000004</td>
</tr>
<tr>
<td>ulSize</td>
<td>ESS_ULONG_T</td>
<td>Percentage of total load buffer resources this load buffer may use. Possible values: 0 to 100.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For a value of 0, Essbase uses a self-determined, default load buffer size.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If the total size of all load buffers exceeds 100, the operation fails.</td>
</tr>
</tbody>
</table>

Notes

- Multiple buffers can exist on a single aggregate storage database; however, only one data load may use a given load buffer at a time.

Return Value

Returns zero if successful; otherwise, returns an error code.

Example

```c
void TestBeginDataloadASO(ESS_HCTX_T hCtx, ESS_STR_T AppName, ESS_STR_T DbName)
{
    ESS_STS_T       sts = ESS_STS_NOERR;
    ESS_BOOL_T      Store;
    ESS_BOOL_T      Unlock;
    ESS_BOOL_T      abortOnError;
    ESS_STR_T       loadString;
    ESS_OBJDEF_T    rulesFile;
    ESS_PMBRERR_T   pMbrErr;
    ESS_ULONG_T     ulBufferId;
    ESS_ULONG_T     ulDuplicateAggregationMethod;
    ESS_ULONG_T     ulOptionsFlags;
```
ESS_ULONG_T     ulSize;
ESS_ULONG_T     ulBufferCnt;
ESS_ULONG_T     ulCommitType ;
ESS_ULONG_T     ulActionType;
ESS_ULONG_T     ulOptions;
ESS_ULONG_T     ulBufferIdAry[1];

/* EssLoadBufferInit */
ulDuplicateAggregationMethod = ESS_ASO_DATA_LOAD_BUFFER_DUPLICATES_ADD;
ulOptionsFlags = ESS_ASO_DATA_LOAD_BUFFER_IGNORE_MISSING_VALUES;
ulSize = 100;
ulBufferId = 201;
sts = EssLoadBufferInit(hCtx,AppName,DbName,ulBufferId,ulDuplicateAggregationMethod,
                       ulOptionsFlags,ulSize);
printf("EssLoadBufferInit sts: %ld\n", sts);

/* EssBeginDataloadASO, EssSendString, EssEndDataload */
Store = ESS_TRUE;
Unlock = ESS_FALSE;
abortOnError = ESS_FALSE;
loadString = "Mar Sale \"Curr Year\" \"Original Price\" \"017589\" \"13668\" Cash \"No Promotion\" \"1 to 13 Years\" \"Under 20,000\" \"Digital Cameras\" 111";

sts = EssBeginDataloadASO (hCtx, Store, Unlock, abortOnError, ESS_NULL, ulBufferId);
printf("EssBeginDataloadASO sts: %ld\n",sts);
sts = EssSendString(hCtx,loadString);
printf("EssSendString sts: %ld\n",sts);
sts = EssEndDataload(hCtx,&pMbrErr);
printf("EssEndDataload sts: %ld\n",sts);

/* EssLoadBufferTerm */
ulBufferCnt = 1;
ulBufferIdAry[0] = ulBufferId;
ulCommitType = ESS_ASO_DATA_LOAD_BUFFER_STORE_DATA;
ulActionType = ESS_ASO_DATA_LOAD_BUFFER_COMMIT;
printf("Commit data to main slice and destroy buffer:\n");
ulOptions = ESS_ASO_DATA_LOAD_INCR_TO_MAIN_SLICE;
sts = EssLoadBufferTerm(hCtx,AppName,DbName,ulBufferCnt,ulBufferIdAry,ulCommitType,
                       ulActionType,ulOptions);
printf("EssLoadBufferTerm sts: %ld\n",sts);

}
EssLoadBufferTerm

Destroys the temporary data-load memory buffer(s) allocated by EssLoadBufferInit for loading data into an aggregate storage database. Optionally, the data can be committed first.

Applies only to aggregate storage databases.

Syntax

```c
ESS_FUNC_M EssLoadBufferTerm (hCtx,AppName,DbName,ulBufferCnt,*ulBufferIdAry,
        ulCommitType,
        ulActionType, ulOptions);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>AppName</td>
<td>ESS_STR_T</td>
<td>Name of the application.</td>
</tr>
<tr>
<td>DbName</td>
<td>ESS_STR_T</td>
<td>Name of the database.</td>
</tr>
<tr>
<td>ulBufferCnt</td>
<td>ESS_ULONG_T</td>
<td>Number of buffers in the list.</td>
</tr>
<tr>
<td>*ulBufferIdAry</td>
<td>ESS_ULONG_T</td>
<td>Array of buffer IDs that will be affected by this operation.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ulCommitType</td>
<td>ESS_ULONG_T</td>
<td>One of the following constants for combining the values stored in the buffer with the values already stored in the database:</td>
</tr>
</tbody>
</table>
|               |             | - ESS_ASO_DATA_LOAD_BUFFER_STORE_DATA: Replace existing cell values in the database with the new values from the load buffer. Cells in the database that do not have corresponding values in the buffer are not updated.  
#define ESS_ASO_DATA_LOAD_BUFFER_STORE_DATA   0  |
|               |             | - ESS_ASO_DATA_LOAD_BUFFER_ADD_DATA: Add new values to the existing ones.  
#define ESS_ASO_DATA_LOAD_BUFFER_ADD_DATA   1  |
|               |             | - ESS_ASO_DATA_LOAD_BUFFER_SUBTRACT_DATA: Subtract new values from the existing ones.  
#define ESS_ASO_DATA_LOAD_BUFFER_SUBTRACT_DATA   2  |
|               |             | - ESS_ASO_DATA_LOAD_BUFFER_OVERRIDE_ALL_DATA: Atomically destroy all existing data cells in the database (even cells in the database that do not have corresponding values in the load buffer) and load the contents of the load buffer in one operation.  
#define ESS_ASO_DATA_LOAD_BUFFER_OVERRIDE_ALL_DATA   3  |
|               |             | When using the override all data option, the ulOptions setting is ignored. Essbase always writes the data currently stored in the buffer to the main slice in the database.                  |
|               |             | - ESS_ASO_DATA_LOAD_BUFFER_OVERRIDE_INCREMENTAL_DATA: Atomically destroy all data cells currently stored in any incremental slice and load the contents of the load buffer.  
#define ESS_ASO_DATA_LOAD_BUFFER_OVERRIDE_INCREMENTAL_DATA   4  |
|               |             | When using the override incremental data option and the ulOptions setting is main slice, Essbase ignores the ulOptions setting and writes the data currently stored in the buffer to a new slice in the database.                  |
|               |             | When committing multiple buffers, the values from different buffers are always combined using the add operation, regardless of this ulCommitType setting or how the buffers themselves are configured.                  |
|               |             | **Note:** If the ulActionType setting is abort, the ulCommitType setting is ignored. |
| ulActionType  | ESS_ULONG_T | One of the following constants:                                                                                                                                 |
|               |             | - ESS_ASO_DATA_LOAD_BUFFER_COMMIT: Load the data from the load buffer to the database; then destroy the buffer.  
#define ESS_ASO_DATA_LOAD_BUFFER_COMMIT   1  |
|               |             | - ESS_ASO_DATA_LOAD_BUFFER_ABORT: Destroy the load buffer. All data in the buffer is lost.  
#define ESS_ASO_DATA_LOAD_BUFFER_ABORT   2  |
<p>|               |             | When using the abort option, the ulCommitType and ulOptions settings are ignored |</p>
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ulOptions</td>
<td>ESS_ULONG_T</td>
<td>One of the following constants:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_ASO_DATA_LOAD_INCR_TO_MAIN_SLICE: Write the data currently stored in the buffer to the main slice in the database.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>#define ESS_ASO_DATA_LOAD_INCR_TO_MAIN_SLICE 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>When using the incremental to main slice option, and the ulCommitType setting is override incremental data, Essbase ignores the ulOptions setting and writes the data currently stored in the buffer to a new slice in the database.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_ASO_DATA_LOAD_INCR_TO_NEW_SLICE: Write the data currently stored in the buffer to a new slice in the database. This operation speeds up the data load.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>#define ESS_ASO_DATA_LOAD_INCR_TO_NEW_SLICE 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_ASO_DATA_LOAD_INCR_TO_NEW_SLICE_LIGHTWEIGHT: Write the data currently stored in the buffer to a new slice in the database, as a lightweight operation. This option is intended only for very small data loads of up to 1,000s of cells that occur concurrently (for example, spreadsheet lock and send operations).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>#define ESS_ASO_DATA_LOAD_INCR_TO_NEW_SLICE_LIGHTWEIGHT 2</td>
</tr>
</tbody>
</table>

**Note:** If the ulCommitType setting is override all data, the ulOptions setting is ignored. Essbase always writes the data currently stored in the buffer to the main slice in the database. If the ulActionType setting is abort, the ulOptions setting is ignored.

**Notes**

This function destroys the specified set of load buffers (usually a single load buffer). If the specified action type is "commit," data currently stored in the buffer is applied to the database before the buffers are destroyed.

**Return Value**

Returns zero if successful; otherwise, returns an error code.

**Example**

```c
void TestBeginDataloadASO(ESS_HCTX_T hCtx, ESS_STR_T AppName, ESS_STR_T DbName) {
    ESS_STS_T       sts = ESS_STS_NOERR;
    ESS_BOOL_T      Store;
    ESS_BOOL_T      Unlock;
    ESS_BOOL_T      abortOnError;
    ESS_STR_T       loadString;
    ESS_OBJDEF_T    rulesFile;
    ESS_PMBRERR_T   pMbrErr;
    ESS_ULONG_T     ulBufferId;
    ESS_ULONG_T     ulDuplicateAggregationMethod;
    ESS_ULONG_T     ulCommitType ;
    ESS_ULONG_T     ulActionType;
```
ESS ULONG_T ulOptions;
ESS ULONG_T ulBufferIdAry[1];

/* EssLoadBufferInit */
ulDuplicateAggregationMethod = ESS_ASO_DATA_LOAD_BUFFER_DUPLICATES_ADD;
ulOptionsFlags = ESS_ASO_DATA_LOAD_BUFFER_IGNORE_MISSING_VALUES;
ulSize = 100;
ulBufferId = 201;
sts = EssLoadBufferInit(hCtx, AppName, DbName, ulBufferId,
ulDuplicateAggregationMethod,
ulOptionsFlags, ulSize);
printf("EssLoadBufferInit sts: %ld\n", sts);

/* EssBeginDataloadASO, EssSendString, EssEndDataload */
Store = ESS_TRUE;
Unlock = ESS_FALSE;
abortOnError = ESS_FALSE;
loadString = "Mar Sale \"Curr Year\" \"Original Price\" \"017589\" \"13668\"";
Cash \"No Promotion\" \"1 to 13 Years\" \"Under 20,000\" \"Digital Cameras\" 111;
sts = EssBeginDataloadASO (hCtx, Store, Unlock, abortOnError, ESS_NULL, ulBufferId);
printf("EssBeginDataloadASO sts: %ld\n",sts);
sts = EssSendString(hCtx, loadString);
printf("EssSendString sts: %ld\n",sts);
sts = EssEndDataload (hCtx, &pMbrErr);
printf("EssEndDataload sts: %ld\n",sts);

/* EssLoadBufferTerm */
ulBufferCnt = 1;
ulBufferIdAry[0] = ulBufferId;
ulCommitType = ESS_ASO_DATA_LOAD_BUFFER_STORE_DATA;
ulActionType = ESS_ASO_DATA_LOAD_BUFFER_COMMIT;
printf("\nCommit data to main slice and destroy buffer:\n");
ulOptions = ESS_ASO_DATA_LOAD_INCR_TO_MAIN_SLICE;
sts = EssLoadBufferTerm(hCtx, AppName, DbName, ulBufferCnt, ulBufferIdAry,
ulCommitType,
ulActionType, ulOptions);
printf("EssLoadBufferTerm sts: %ld\n",sts);

See Also
- EssLoadBufferInit
- EssBeginDataloadASO
- EssSendString
- EssEndDataload
- EssImportASO
- EssUpdateFileASO
- EssUpdateFileUTF8ASO
- EssListExistingLoadBuffers
- EssMergeDatabaseData
**EssLoadDatabase**

Starts a database within an Application on the server.

**Syntax**

```c
ESS_FUNC_M EssLoadDatabase (hCtx,AppName,DbName);
```

**Parameter | Data Type | Description**

- **hCtx** | ESS_HCTX_T | API context handle.
- **AppName** | ESS_STR_T | Application name.
- **DbName** | ESS_STR_T | Name of database to load.

**Return Value**

None.

**Access**

This function requires the caller to have databaseLoad/Unload privilege (ESS_PRIV_APPLOAD).

**Example**

```c
ESS_FUNC_M
ESS_LoadDb (ESS_HCTX_T  hCtx)
{
    ESS_FUNC_M    sts;
    ESS_STR_T   AppName;
    ESS_STR_T    DbName;

   AppName = "Sample";
   DbName  = "Basic";
   sts = EssLoadDatabase(hCtx,AppName,DbName);

    return (sts);
}
```

**See Also**

- [EssLoadApplication](#)
- [EssUnloadDatabase](#)

**EssLocateIBH**

Locates invalid block headers within the database. At the end of the locate process, a server-based IBH log file is created that can be used later in [EssFixIBH](#) to fix the errors.

**Syntax**

```c
ESS_FUNC_M EssLocateIBH (hCtx, dbName);
```
### Parameter Data Type Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle. Can be local context handle returned by <code>EssCreateLocalContext()</code>.</td>
</tr>
<tr>
<td>dbName</td>
<td>ESS_STR_T</td>
<td>Name of the database.</td>
</tr>
</tbody>
</table>

#### See Also
- `EssFixIBH`
- `EssGetIBH`

### EssLockObject

Locks an object on the server or the client object system to prevent other users from updating it.

#### Syntax

```c
ESS_FUNC_M EssLockObject (hCtx, ObjType, AppName, DbName, ObjName);
```

#### Parameter Data Type Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle. Can be local context handle returned by <code>EssCreateLocalContext()</code>.</td>
</tr>
<tr>
<td>ObjType</td>
<td>ESS_OBJTYPE_T</td>
<td>Object type (must be single type). Refer to &quot;Bitmask Data Types (C)&quot; on page 96 for a list of possible values.</td>
</tr>
<tr>
<td>AppName</td>
<td>ESS_STR_T</td>
<td>Application name.</td>
</tr>
<tr>
<td>DbName</td>
<td>ESS_STR_T</td>
<td>databasename. If NULL, uses the application subdirectory.</td>
</tr>
<tr>
<td>ObjName</td>
<td>ESS_STR_T</td>
<td>Name of object to lock.</td>
</tr>
</tbody>
</table>

#### Notes
- To lock an object, the object must already exist and not be locked by another user.
- This function does not retrieve the object. Use `EssGetObject()` to retrieve the object.

#### Return Value

None.

#### Access

This function requires the caller to have application or database Design privilege (ESS_PRIV_APPDESIGN or ESS_PRIV_DBDESIGN) for the specified application or database containing the object.

#### Example

```c
ESS_FUNC_M
EssLockObject (ESS_HCTX_T hCtx) {
    ESS_FUNC_M         sts = ESS_STS_NOERR;
    ESS_STR_T         AppName;
    ESS_STR_T         DbName;
```
ESS_STR_T ObjName;
ESS_OBJTYPE_T ObjType;
AppName = "Sample";
DbName = "Basic";
ObjName = "Basic";
ObjType = ESS_OBJTYPE_OUTLINE;

sts = EssLockObject (hCtx, ObjType, AppName,
                   DbName, ObjName);
if (!sts)
    printf("The Object "%s" is locked\n",
           ObjName);
    return (sts);
}

See Also
- EssGetObject
- EssGetObjectInfo
- EssListObjects
- EssPutObject
- EssUnlockObject

EssLogin

Logs in a user to an Essbase Server. This function should normally be called after executing a successful call to EssInit, and prior to making any other API calls which require a context handle argument.

Syntax

ESS_FUNC_M EssLogin (hInstance, Server, UserName, Password, pDbCount, ppDbList, phCtx);

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>ESS_HINST_T</td>
<td>API instance handle.</td>
</tr>
<tr>
<td>Server</td>
<td>ESS_STR_T</td>
<td>Network server name string.</td>
</tr>
</tbody>
</table>

The server name can be expressed as hostname, hostname:port, or as a URL representing the APS servlet endpoint with the Essbase failover cluster name; for example:

http://myhost:13080/aps/Essbase?clustername=Esslbase-Cluster1

For secure mode (SSL), the URL syntax is

http[s]://host:port/aps/Essbase?
ClusterName=logicalName&SecureMODE=yesORno

For example,

### Parameter | Data Type | Description
--- | --- | ---
UserName | ESS_STR_T | User name string.
Password | ESS_STR_T | Password string.
pDbCount | ESS_PUSHORT_T | Address of variable to receive count of accessible applications/databases.
ppDbList | "ESS_APPDB_T" on page 114 | Address of pointer to receive allocated array of application/databasename structures.
phCtx | ESS_PHCTX_T | Pointer to an Essbase Server context handle.

### Notes
- If you are programming in Microsoft Windows, you should consider using the EssAutoLogin function instead of EssLogin.
- Memory allocated for `ppDbList` must be freed using EssFree.
- You can call EssLogin more than once for the same user name and server. The API returns a unique context handle for each login to the specified server.

### Return Value
If successful, returns an Essbase Server context handle in `phCtx`, which can be used as an argument in subsequent calls to other API functions. Also returns a count of databases accessible to the specified user in `pCount`, and a list of accessible applications and databases in `ppDbList`.

### Access
Before calling this function, you must first initialize the API and obtain a valid instance handle by calling EssInit.

### Example
```c
ESS_FUNC_M
ESS_Login (ESS_HINST_T hInst)
{
    ESS_FUNC_M    sts = ESS_STS_NOERR;
    ESS_HCTX_T   hCtx;
    ESS_USHORT_T Items;
    ESS_USHORT_T ind;
    ESS_PAPPDB_T pAppsDbs = NULL;
    ESS_STR_T    SvrName;
    ESS_STR_T    User;
    ESS_STR_T    Password;

    SvrName  = "POPLAR";
    User     = "Joseph";
    Password = "Password";

    sts = EssLogin (hInst, SvrName, User, Password,
                    &Items, &pAppsDbs, &hCtx);
    if (!sts)
    {
        for (ind = 0; ind < Items; ind++)
        {
```
if ((pAppsDbs+ind) != NULL) {
    if ( ((pAppsDbs[ind].AppName != NULL) &&
          (pAppsDbs[ind].DbName  != NULL))
    {
        printf("%s\n", pAppsDbs[ind].AppName);
        printf("%s\n", pAppsDbs[ind].DbName);
    }
    }
} return(sts);

See Also

- EssAutoLogin
- EssLoginAs
- EssLoginEx
- EssLoginExAs
- EssInit
- EssListDatabases
- EssLogout
- EssSetActive

### EssLoginAs

Logs in to Essbase Server as another user. Logging in as another user can help administrators create scheduled reports with user-appropriate permissions.

This function should normally be called after executing a successful call to EssInit, and prior to making any other API calls which require a context handle argument.

**Syntax**

```c
ESS_FUNC_M EssLoginAs (hInstance, Server, UserName, Password, UserNameAs, pDbCount, ppDbList, phCtx);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>ESS_HINST_T</td>
<td>API instance handle.</td>
</tr>
</tbody>
</table>
### Parameter | Data Type | Description
--- | --- | ---
Server | ESS_STR_T | Network server name string.
The server name can be expressed as `hostname`, `hostname:port`, or as a URL representing the APS servlet endpoint with the Essbase failover cluster name; for example:
http://myhost:13080/aps/Essbase?clustername=Essbase-Cluster1
For secure mode (SSL), the URL syntax is
http[s]://host:port/aps/Essbase?
ClusterName=logicalName&SecureMODE=yesORno
For example,

UserName | ESS_STR_T | User name string.

Password | ESS_STR_T | Password string.

UserNameAs | ESS_STR_T | User name string for the user you want to impersonate.

pDbCount | ESS_PUSHORT_T | Address of variable to receive count of accessible applications/databases.

ppDbList | “ESS_APPDB_T” on page 114 | Address of pointer to receive allocated array of application/databasename structures.

phCtx | ESS_PHCTX_T | Pointer to an Essbase Server context handle.

### Notes
- Memory allocated for `ppDbList` must be freed using EssFree.
- You can call EssLogin more than once for the same user name and server. The API returns a unique context handle for each login to the specified server.

### Return Value
If successful, returns an Essbase Server context handle in `phCtx`, which can be used as an argument in subsequent calls to other API functions. Also returns a count of databases accessible to the specified user in `pCount`, and a list of accessible applications and databases in `ppDbList`.

### Access
You must be an administrator to log in as another user.

Before calling this function, you must first initialize the API and obtain a valid instance handle by calling EssInit.

### See Also
- EssAutoLogin
- EssLoginExAs
- EssInit
- EssListDatabases
EssLoginEx

Logs in a user to an Essbase Server using a user authentication token rather than a username and password. This function should normally be called after executing a successful call to EssInit(), and prior to making any other API calls which require a context handle argument.

Syntax

ESS_FUNC_M EssLoginEx (hInstance, Server, Token, pDbCount, ppDbList, phCtx);

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>ESS_HINST_T</td>
<td>API instance handle.</td>
</tr>
<tr>
<td>Server</td>
<td>ESS_STR_T</td>
<td>Network server name string.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The server name can be expressed as hostname, hostname:port, or as a URL representing the APS servlet endpoint with the Essbase failover cluster name; for example:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><a href="http://myhost:13080/aps/Essbase?clusternamelEx=Essbase-Cluster1">http://myhost:13080/aps/Essbase?clusternamelEx=Essbase-Cluster1</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td>For secure mode (SSL), the URL syntax is</td>
</tr>
<tr>
<td></td>
<td></td>
<td>http[s]://host:port/aps/Essbase?ClusterName=logicalName&amp;SecureMODE=yesORno</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For example,</td>
</tr>
<tr>
<td></td>
<td></td>
<td><a href="https://myhost:13080/aps/Essbase?clusternamelEx=Essbase-Cluster1&amp;SecureMODE=Yes">https://myhost:13080/aps/Essbase?clusternamelEx=Essbase-Cluster1&amp;SecureMODE=Yes</a></td>
</tr>
<tr>
<td>Token</td>
<td>ESS_STR_T</td>
<td>The token representing the username and password of an authenticated user.</td>
</tr>
<tr>
<td>pDbCount</td>
<td>ESS_PUSHORT_T</td>
<td>Address of variable to receive count of accessible applications/databases.</td>
</tr>
<tr>
<td>ppDbList</td>
<td>“ESS_APPDB_T” on page 114</td>
<td>Address of pointer to receive allocated array of application/databasename structures.</td>
</tr>
<tr>
<td>phCtx</td>
<td>ESS_PHCTX_T</td>
<td>Pointer to an Essbase Server context handle.</td>
</tr>
</tbody>
</table>

Notes

- If this function fails, the corresponding EssLogin() function is automatically called in order to try to verify a username and password for the user.
- Memory allocated for ppDbList must be freed using EssFree().

Return Value

If successful, returns an Essbase Server context handle in phCtx, which can be used as an argument in subsequent calls to other API functions. Also returns a count of databases accessible to the specified user in pCount, and a list of accessible applications and databases in ppDbList.
Access

Before calling this function, you must first initialize the API and obtain a valid instance handle by calling `EssInit()`.

See Also

- `EssLogin`
- `EssLoginAs`
- `EssLoginExAs`
- `EssAutoLogin`
- `EssInit`
- `EssListDatabases`
- `EssLogout`
- `EssSetActive`

**EssLoginExAs**

Logs in an administrator to an Essbase Server as another user, and using a user authentication token rather than the administrator username and password. This function should normally be called after executing a successful call to `EssInit`, and prior to making any other API calls which require a context handle argument.

Logging in as another user can help administrators create scheduled reports with user-appropriate permissions.

**Syntax**

```c
ESS_FUNC_M EssLoginExAs (hInstance, Server, Token, UserNameAs, pDbCount, ppDbList, phCtx);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>ESS_HINST_T</td>
<td>API instance handle.</td>
</tr>
<tr>
<td>Server</td>
<td>ESS_STR_T</td>
<td>Network server name string.</td>
</tr>
<tr>
<td>Token</td>
<td>ESS_STR_T</td>
<td>The token representing the user name and password of an authenticated user.</td>
</tr>
<tr>
<td>UserNameAs</td>
<td>ESS_STR_T</td>
<td>User name string for the user you want to impersonate.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>pDbCount</td>
<td>ESS_PUSHORT_T</td>
<td>Address of variable to receive count of accessible applications/databases.</td>
</tr>
<tr>
<td>ppDbList</td>
<td>&quot;ESS_APPDB_T&quot; on page 114</td>
<td>Address of pointer to receive allocated array of application/databases.</td>
</tr>
<tr>
<td>phCtx</td>
<td>ESS_PHCTX_T</td>
<td>Pointer to an Essbase Server context handle.</td>
</tr>
</tbody>
</table>

**Notes**

- If this function fails, the corresponding EssLoginAs function is automatically called in order to try to verify a username and password for the user.
- Memory allocated for ppDbList must be freed using EssFree.

**Return Value**

If successful, returns an Essbase Server context handle in phCtx, which can be used as an argument in subsequent calls to other API functions. Also returns a count of databases accessible to the specified user in pCount, and a list of accessible applications and databases in ppDbList.

**Access**

You must be an administrator to log in as another user.

Before calling this function, you must first initialize the API and obtain a valid instance handle by calling EssInit.

**See Also**

- EssLogin
- EssLoginAs
- EssAutoLogin
- EssInit
- EssListDatabases
- EssLogout
- EssSetActive

---

**EssLoginSetPassword**

Logs in a user, and changes the password. Use this function if the password expires, or must be changed at the next login.

**Syntax**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HInstance</td>
<td>ESS_HINST_T</td>
<td>API instance handle.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>----------------</td>
<td>-------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Server</td>
<td>ESS_STR_T</td>
<td>Network server name string. The server name can be expressed as hostname, hostname:port, or as a URL representing the APS servlet endpoint with the Essbase failover cluster name; for example: <a href="http://myhost:13080/aps/Essbase?clusternam=Essbase-Cluster1">http://myhost:13080/aps/Essbase?clusternam=Essbase-Cluster1</a> For secure mode (SSL), the URL syntax is http[s]://host:port/aps/Essbase?ClusterName=logicalName&amp;SecureMODE=yesORno For example, <a href="https://myhost:13080/aps/Essbase?clusternam=Essbase-Cluster1&amp;SecureMODE=Yes">https://myhost:13080/aps/Essbase?clusternam=Essbase-Cluster1&amp;SecureMODE=Yes</a></td>
</tr>
<tr>
<td>UserName</td>
<td>ESS_STR_T</td>
<td>User name.</td>
</tr>
<tr>
<td>Password</td>
<td>ESS_STR_T</td>
<td>Old password.</td>
</tr>
<tr>
<td>NewPassword</td>
<td>ESS_STR_T</td>
<td>New password.</td>
</tr>
<tr>
<td>pDbCount</td>
<td>ESS_PUSHORT_T</td>
<td>Number of accessible databases.</td>
</tr>
<tr>
<td>ppDbList</td>
<td>ESS_PPAPPDB_T</td>
<td>Address of the pointer to an array of accessible application-database structures.</td>
</tr>
<tr>
<td>phCtx</td>
<td>ESS_PHCTX_T</td>
<td>Pointer to the context handle.</td>
</tr>
</tbody>
</table>

**Notes**

- Call EssLoginSetPassword after you call EssLogin, and after you receive status code 1051090 (Password has expired), or 1051093 (Change password now).
- In Microsoft Windows, consider using EssAutoLogin, instead of EssLoginSetPassword.
- Free memory allocated for ppDbList using EssFree.

**Return Value**

If successful, EssLoginSetPassword returns:

- In hCtx, the context handle.
- In pDbCount, the number of databases accessible to the user.
- In ppDbList, the pointer to an array of accessible application-database structures.

**Access**

Before you call EssLoginSetPassword, call EssInit to initialize the API, and obtain a valid instance handle.

**Example**

```c
ESS_FUNC_M
ESS_LoginSetPassword (ESS_HINST_T  hInst)
```
```

ESS_FUNC_M   sts = ESS_STS_NOERR;
ESS_HCTX_T   hCtx;
ESS_USHORT_T Items;
ESS_USHORT_T ind;
ESS_PAPPDB_T pAppsDbs = NULL;
ESS_STR_T    SvrName;
ESS_STR_T    User;
ESS_STR_T    Password;
ESS_STR_T    NewPassword;

SvrName     = "POPLAR";
User        = "Joseph";
Password    = "Password";
NewPassword = "NewPassword";

sts = EssLoginSetPassword (hInst, SvrName, User, Password, NewPassword
                      &Items, &pAppsDbs, &hCtx);

if (!sts)
{
    for (ind = 0; ind < Items; ind++)
    {
        if ((pAppsDbs+ind) != NULL)
        {
            if ((pAppsDbs[ind].AppName != NULL) &&
                (pAppsDbs[ind].DbName != NULL))
            {
                printf ("%s\r\n", pAppsDbs[ind].AppName);
                printf ("%s\r\n", pAppsDbs[ind].DbName);
            }
        }
    }
    if (pAppsDbs)
        EssFree(hInst,pAppsDbs);
}
return(sts);
}

See Also

- EssAutoLogin
- EssInit
- EssListDatabases
- EssLogout
- EssSetActive

**EssLogout**

Logs out a user from an Essbase Server.

**Syntax**

```

ESS_FUNC_M EssLogout (hCtx);
```

```
### Parameter Data Type Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle to logout.</td>
</tr>
</tbody>
</table>

### Notes

- This function logs out only the login represented by the specified context handle. No other logins or contexts are affected, even if using the same user name.
- This function should only be used for login contexts. For local contexts, use the EssDeleteLocalContext() function.

### Return Value

None.

### Access

To call this function, the caller must have previously logged in successfully using either the EssLogin() or EssAutoLogin functions.

### Example

```c
ESS_FUNC_M
ESSLogout (ESS_HCTX_T hCtx)
{
    ESS_FUNC_M sts = ESS_STS_NOERR;
    sts = EssLogout (hCtx);
    return(sts);
}
```

### See Also

- EssAutoLogin
- EssDeleteLocalContext
- EssGetActive
- EssLogin
- EssLogoutUser

---

### EssLogoutUser

Allows a Supervisor or an Application Designer to disconnect another user from an Essbase Server.

### Syntax

```c
ESS_FUNC_M EssLogoutUser (hCtx, LoginId);
```

### Parameter Data Type Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle of user forcing the log out.</td>
</tr>
<tr>
<td>LoginId</td>
<td>ESS_LOGINID_T</td>
<td>Login ID of user to be logged out.</td>
</tr>
</tbody>
</table>

---
Notes

- `LoginId` can be obtained from the user information structure returned by the `EssListConnections()` function.
- This function logs out only the login represented by the specified `LoginId`. No other logins or contexts are affected.
- A Supervisor can log out anyone logged in to the server to which `hCtx` is logged in. An Application Designer can log out only those users connected to an application for which `hCtx` is an Application Designer. You can't log yourself out.

Return Value

None.

Access

To call this function, you must have Supervisor or Application Designer privilege.

Example

```c
ESS_FUNC_M ESS_LogoutUser (ESS_HCTX_T hCtx,
ESS_HINST_T hInst)
{
    ESS_FUNC_M sts = ESS_STS_NOERR;
    ESS_USHORT_T usrcnt;
    ESS_PUSERINFO_T users;
    sts = EssListConnections(hCtx, &usrcnt,
        &users);
    if(!sts)
    {
        if(usrcnt > 0)
        {
            /***************************************
             * Log out first user from the list *
             *******************************************/
            sts = EssLogoutUser(hCtx, users[0].LoginId);
            if(!sts)
                EssFree(hInst, users);
        }
        return(sts);
    }
}
```

See Also

- `EssListConnections`
- `EssLogout`

**EssLogSize**

Returns the size of the Essbase Server log file (essbase.log), or of the application log file (appname.log).
Syntax

ESS_FUNC_M EssLogSize (hCtx, AgentLog, pszAppName, pulLogSize);

Parameter | Data Type | Description
---|---|---
hCtx | ESS_HCTX_T | API context handle.
AgentLog | ESS_BOOL_T | If TRUE, the size of the Essbase Server log file (essbase.log) is returned. If FALSE, the size of the application log file (appname.log) is returned.
pszAppName | ESS_STR_T | Application name.
pulLogSize | ESS_PULONG_T | Size of log file returned.

Notes

- Use EssGetLogFile() to view message logs.
- For the locations of essbase.log and appname.log, see the Oracle Essbase Database Administrator's Guide.

Return Value

Returns a zero if successful.

Access

This function does not require the caller to have access privileges.

Example

ESS_FUNC_M ESS_LogSize (ESS_HCTX_T hCtx)
{
    ESS_STR_T pszAppName = NULL;
    ESS_UULONG_T ulLogSize = 0;
    ESS_FUNC_M sts = ESS_STS_NOERR;

    pszAppName = "Sample";

    /*
    * Get the log file size for the "Sample" application.
    */
    sts = EssLogSize(hCtx, ESS_FALSE, pszAppName, &ulLogSize);

    return(sts);
}

See Also

- EssDeleteLogFile
- EssGetLogFile
- EssWriteToFile

EssLROAddObject

Links reporting objects to a data cell in an Essbase database.
Syntax

ESS_FUNC_M EssLROAddObject (hCtx, memCount, pMemComb, usOption, pLRODesc);

Parameter | Data Type | Description
--- | --- | ---
| hCtx | ESS_HCTX_T | API context handle.
| memCount | ESS_ULONG_T | The number of members specified in pMemComb.
| pMemComb | ESS_PVOID_T | Array of the member names that define the data cell to be linked.
| usOption | ESS_USHORT_T | Option specifying where to store the object. Use one of these values:
  - ESS_STORE_OBJECT_API to store an object on the server.
  - ESS_NOSTORE_OBJECT_API to not store the object on a server.
| pLRODesc | “ESS_LRODESC_API_T” on page 108 | Pointer to object's description structure.

Notes

- The linked object can be any of the following types:
  - A flat file, such as a Word document, Excel spreadsheet, or bitmap image.
  - A cell note containing up to 599 characters of text.
  - A link to a URL.
  - A link to another Essbase database (a linked partitions feature).
- If you elect not to store the object on the server (usOption), your application is responsible for all file management tasks for the object (that is, because the object is not being stored with the Essbase database, some other program must take responsibility for it).
- The usOption parameter is ignored for cell notes, which are always stored on the server.
- The usOption parameter for a URL linked object should always be ESS_NOSTORE_OBJECT_API.
- EssLROAddObject uses the currently logged in user name as the "created by" user name for the object and ignores any user name specified in the pLRODesc object description structure.

Return Value

If successful, returns ESS_STS_NOERR. Otherwise, returns an error code.

Access

A call to this function requires write privileges (ESS_PRIV_WRITE) to the active database.

Example

```c
ESS_STS_T EssLROAddObject (ESS_HCTX_T hCtx, ESS_HINST_T hInst)
{
    ESS_STS_T   sts = ESS_STS_NOERR;
    ESS_PMBRNAME_NONUNI_T   pMemComb = NULL;
    ESS_LRODESC_API_T       lroDesc;
    ESS_USHORT_T            usOption = 0;
```
ESSULONG_T memCount;

memset (&lroDesc, 0 , sizeof(ESS_LRODESC_API_T));

lroDesc.usObjType = 0;    /* Creating a cell note */
strcpy(lroDesc.lro.note, "The profit for Colas in the East based on actuals");
usOption = ESS_NOSTORE_OBJECT_API;
strcpy(lroDesc.userName, "user1");
memCount = 5;

sts = EssAlloc(hInst, memCount*sizeof(ESS_MBRNAME_NONUNI_T),
               (ESS_PPVOID_T)&pMemComb);
if (sts)
{
    printf("could not allocate memory\n");
    return sts;
}

memset(pMemComb, 0, memCount*sizeof(ESS_MBRNAME_NONUNI_T));
strcpy( pMemComb[0], "Profit");
strcpy( pMemComb[1], "East");
strcpy( pMemComb[2], "Actual");
strcpy( pMemComb[3], "Colas");
strcpy( pMemComb[4], "Year");

sts = EssLROAddObject( hCtx, memCount, pMemComb, usOption, &lroDesc);
if (sts)
{
    printf( "Could not attach LRO\n");
}
EssFree(hInst, pMemComb);
return sts;

See Also

- "LRO Constant and Structure Definitions (C)" on page 107
- EssLROGetObject
- EssLROUpdateObject
- EssLRODeleteObject

**EssLRODeleteCellObjects**

Deletes all objects linked to a given data cell in an Essbase database. To delete a specific object linked to a cell, use **EssLRODeleteObject**.

**Syntax**

```c
ESS_FUNC_M EssLRODeleteCellObjects (hCtx, memCount, pMemComb, pulLROCount, pLRODescList);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
</tbody>
</table>
Parameter | Data Type | Description
--- | --- | ---
memCount | ESS_ULONG_T | Number of members specified in \$pMemComb\$.
pMemComb | ESS_PVOID_T | Array of member names.
pullRCount | ESS_ULONG_T | Number of LRO catalog entries deleted.
plRODescList | “ESS_LRODESC_API_T” on page 108 | List of LRO catalog entries deleted.

Notes
- This function deletes all objects linked to the specified cell along with their catalog entries.
- If the object is not stored on the server, only the cell link is destroyed; the file is not deleted.
- The caller is responsible for freeing memory allocated for \$pLRODescList\$.

Return Value
If successful, returns ESS_STS_NOERR. Otherwise, returns an error code.

Access
A call to this function requires write privileges (ESS_PRIV_WRITE) to the active database.

Example

```c
ESS_FUNC_M ESS_LRO DeleteCellObjects (ESS_HCTX_T hCtx, ESS_HINST_T hInst) {
    ESS_LRODESC_API_T plroDescList=NULL;
    ESS_PMBRNAME_NONUNI_T pMemComb = NULL;
    ESS_ULONG_T memCount;
    ESS_FUNC_M sts = ESS_STS_NOERR;
    ESS_ULONG_T ulLroCount;
    memCount = 5;
    sts = EssAlloc(hInst, memCount*sizeof(ESS_MBRNAME_NONUNI_T),
    (ESS_PPVOID_T)&pMemComb);
    if(sts)
    {
        printf("Could not allocate memory \n");
        return sts;
    }
    memset(pMemComb, 0, memCount*sizeof(ESS_MBRNAME_NONUNI_T));
    strcpy( pMemComb[0], "Profit");
    strcpy( pMemComb[1], "East");
    strcpy( pMemComb[2], "Actual");
    strcpy( pMemComb[3], "Colas");
    strcpy( pMemComb[4], "Year");
    sts = EssLRODeleteCellObjects(hCtx, memCount, pMemComb, &ulLroCount,
    &plroDescList);
    if (sts)
    {
        printf ("Could not delete cell objects. \n");
    }
    EssFree( hInst, pMemComb);
    if (plroDescList)
        EssFree(hInst, plroDescList);
} 518 C Main API Functions
return sts;
}

See Also
- “LRO Constant and Structure Definitions (C)” on page 107
- EssLROAddObject
- EssLRODeleteObject
- EssLROPurgeObjects

EssLRODeleteObject

 Deletes a specific object linked to a data cell in an Essbase database. To delete all objects linked to a cell, use EssLRODeleteCellObjects.

Syntax

ESS_FUNC_M EssLRODeleteObject (hCtx, plinkId);

Parameter | Data Type | Description
--- | --- | ---
hCtx | ESS_HCTX_T | API context handle.
plinkId | “ESS_LROHANDLE_API_T” on page 109 | Pointer to object identification structure.

Notes
- The specified object is deleted and also removed from the Catalog list.
- If the object is not stored on the server, only the cell link is destroyed; the file is not deleted.

Return Value

If successful, returns ESS_STS_NOERR. Otherwise, returns an error code.

Access

A call to this function requires write privileges (ESS_PRIV_WRITE) to the active database.

Example

ESS_FUNC_M Ess_LRO_DeleteObject (ESS_HCTX_T hCtx)
{
    ESS_FUNC_M   sts = ESS_STS_NOERR;
    ESS_LROHANDLE_API_T  linkId;
    memset(&linkId, 0, sizeof(ESS_LROHANDLE_API_T));
    linkId.hObject = 26;
    linkId.cellKey.cellOffset = 282;
    linkId.cellKey.blkOffset = 113;
    linkId.cellKey.segment = 0;
    sts = EssLRODeleteObject(hCtx, &linkId);
    if (sts)
    {
        printf("Could not delete object\n");
    }
    return sts;
}
See Also

- “LRO Constant and Structure Definitions (C)” on page 107
- EssLROAddObject
- EssLRODeleteCellObjects
- EssLROPurgeObjects

**EssLROGetCatalog**

Retrieves a list of LRO catalog entries for a given data cell in an Essbase database.

**Syntax**

```
ESS_FUNC_M EssLROGetCatalog (hCtx, memCount, pMemComb, pulLROCount, ppLRODescList)
```

**Parameter** | **Data Type** | **Description**
--- | --- | ---
**hCtx** | ESS_HCTX_T | API context handle.
**memCount** | ESS_ULONG_T | Number of members specified in **pMemComb**.
**pMemComb** | ESS_PMBRNAMECOMB_T | Array of member names.
**pulLROCount** | ESS_ULONG_T* | Number of LRO catalog entries returned to caller.
**pLRODescList** | “ESS_LRODESC_API_T” on page 108 | Address of pointer to the list of LRO catalog entries returned to caller.

**Return Value**

If successful, returns ESS_STS_NOERR. Otherwise, returns an error code.

**Access**

A call to this function requires read privileges (ESS_PRIV_READ) for the active database.

**Example**

```
ESS_FUNC_M ESS_LRO GetCatalog (ESS_HCTX_T hCtx, ESS_HINST_T hInst)
{
    ESS_PMBRNAME_NONUNI_T pMemComb = NULL;
    ESS_PLLRODESC_API_T plroDescList= NULL;
    ESS_USHORT_T usOption = 0;
    ESS_ULONG_T memCount;
    ESS_FUNC_M sts = ESS_STS_NOERR;
    ESS_ULONG_T ulLroCount;
    memCount = 5;
    sts = EssAlloc(hInst, memCount*sizeof(ESS_MBRNAME_NONUNI_T),
                   (ESS_PPVOID_T)&pMemComb);
    if (sts)
    {
        printf("Could not allocate memory \n");
        return sts;
    }
    memset(pMemComb, 0, memCount*sizeof(ESS_MBRNAME_NONUNI_T));
    strcpy (pMemComb[0], "Profit");
    strcpy (pMemComb[1], "East");
}
```
strcpy( pMemComb[2], "Actual");
strcpy( pMemComb[3], "Colas");
strcpy( pMemComb[4], "Year");
sts = EssLROGetCatalog(hCtx, memCount, pMemComb, &ulLroCount, &plroDescList);
if (sts)
{
    printf ("Could not get the catalog \n");
}
EssFree(hInst, pMemComb);
if(plroDescList)
{
    EssFree(hInst, pMemComb);
}
return sts;

See Also

- “LRO Constant and Structure Definitions (C)” on page 107
- EssLROGetCatalogBatch
- EssLROAddObject
- EssLROUpdateObject
- EssLROGetObject
- EssLRDEleteObject

EssLROGetCatalogBatch

Retrieves a list of LRO catalog entries for multiple data cells in an Essbase database.

Syntax

```
ESS_FUNC_M EssLROGetCatalogBatch (hCtx, memCount, pMemComb, cellCount, pulLROCount, ppLRODescList)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>memCount</td>
<td>ESS_ULONG_T *</td>
<td>Array of 'Number of members' specified in pMemComb, one for each cell.</td>
</tr>
<tr>
<td>pMemComb</td>
<td>ESS_PMBRNAMECOMB_T *</td>
<td>Array of 'member name' combination. Each element of array itself is an array of member names, one for each cell.</td>
</tr>
<tr>
<td>cellCount</td>
<td>ESS_ULONG_T</td>
<td>Count of LRO cells.</td>
</tr>
<tr>
<td>pulLROCount</td>
<td>ESS_ULONG_T *</td>
<td>Array of 'Number of LRO' catalog entries returned to caller. Each element in array corresponds to the number of LRO catalog entries for an input cell.</td>
</tr>
<tr>
<td>pLRODescList</td>
<td>&quot;ESS_LRODESC_API_T&quot; on page 108</td>
<td>Address of pointer to the list of LRO catalog entries returned to caller.</td>
</tr>
</tbody>
</table>
Notes

To use this function, initialize the program with the MaxBuffer field of the initialization structure ESS_INIT_T set to 0xFFFFFFFF bytes.

Return Value

If successful, returns ESS_STS_NOERR. Otherwise, returns an error code.

Access

A call to this function requires read privileges (ESS_PRIV_READ) for the active database.

Example

/*
 * ESS_GetLinkedObjectCatalogBatch() -- Gets a list of LRO description for a list of
given data cell
 * From the Database Sample.Basic, it will fetch LROs for the following Cells.
 * 1) "Jan", "Sales", "100-10", "New York", "Actual"
 * 2) "Feb", "COGS", "200-10", "Utah", "Budget"
 * 3) "Mar", "Payroll", "300-10", "Texas", "Variance"
 */
ESS_STS_T ESS_GetLinkedObjectCatalogBatch(ESS_HINST_T hInst, ESS_HCTX_T hCtx)
{
    ESS_STS_T           status = 0;
    ESS_UINT_T          memberLength = ESS_MBRNAMELEN_NONUNI;
    ESS_PMBRNAME_NONUNI_T *ppMemComb=NULL;
    ESS_PMBRNAME_NONUNI_T pMemComb = NULL;
    ESS_ULONG_T            *pulLroCount= NULL;
    ESS_PLRODESC_API_T     pLroDescList = NULL;
    ESS_PLRODESC_API_T     *ppLroDescList = NULL;
    ESS_ULONG_T             cellCount = 3;            /* Number of cells for which to
retrieve LROs */
    ESS_ULONG_T             mbrrsCount[3] = {5, 5, 5}; /* Number of members in
combinations for each cell */
    ESS_ULONG_T             i,j,k,offset;
    ESS_CHAR_T              *pMember = NULL;
    ESS_CHAR_T              response;

    status = EssAlloc(hInst, cellCount * sizeof(ESS_PMBRNAMECOMB_T),
                      (ESS_PPVOID_T)&ppMemComb);
    if (status)
        goto exit;

    /* Member combination for Cell # 1 */
    status = EssAlloc(hInst, mbrrsCount[0] * memberLength, (ESS_PPVOID_T)&(ppMemComb[0]));
    if (status)
        goto exit;
    pMemComb = ppMemComb[0];
    memset(pMemComb, 0, mbrrsCount[0]* memberLength);
    strcpy((pMemComb)[0], "Jan");
    strcpy((pMemComb)[1], "Sales");
    strcpy((pMemComb)[2], "100-10");
    strcpy((pMemComb)[3], "New York");
    strcpy((pMemComb)[4], "Actual");

    // Rest of the code...
}
/* Member combination for Cell # 2 */
status = EssAlloc(hInst, mbrsCount[1] * memberLength, (ESS_PPVOID_T)&(ppMemComb[1]));
if (status)
    goto exit;
pMemComb = ppMemComb[1];
memset(pMemComb, 0, mbrsCount[1] * memberLength);
strcpy((pMemComb)[0], "Feb");
strcpy((pMemComb)[1], "COGS");
strcpy((pMemComb)[2], "200-10");
strcpy((pMemComb)[3], "Utah");
strcpy((pMemComb)[4], "Budget");

/* Member combination for Cell # 3 */
status = EssAlloc(hInst, mbrsCount[2] * memberLength, (ESS_PPVOID_T)&(ppMemComb[2]));
if (status)
    goto exit;
pMemComb = ppMemComb[2];
memset(pMemComb, 0, mbrsCount[2] * memberLength);
strcpy((pMemComb)[0], "Mar");
strcpy((pMemComb)[1], "Payroll");
strcpy((pMemComb)[2], "300-10");
strcpy((pMemComb)[3], "Texas");
strcpy((pMemComb)[4], "Variance");

/* Will hold information about how many LROs fetched for each Cell */
status = EssAlloc(hInst, cellCount * sizeof(ESS_ULONG_T),
    (ESS_PPVOID_T)&pulLroCount);
if (status)
    goto exit;
memset(pullLroCount, 0, cellCount * sizeof(ESS_ULONG_T));

ppLroDescList = &pLroDescList;

status = EssLROGetCatalogBatch(hCtx, mbrsCount, ppMemComb, cellCount, pulLroCount,
    ppLroDescList);
if (status)
    goto exit;

for (k=0, offset=0; k<cellCount; k++)
{
    ESS_LRODESC_API_T *pLroDesc = &pLroDescList[offset];
    for (i=0; i<pullLroCount[k]; i++, offset++)
    {
        printf("***** information for linked object *****\n");
        printf("Object type - %2d\n", (pLroDesc+i)->usObjType);
        printf("Link Id : \n");
        printf("Object handle - %d\n", (pLroDesc+i)->linkId.hObject);
        printf("Cell offset - %d\n", (pLroDesc+i)->linkId.cellKey.cellOffset);
        printf("Block offset - %lf\n", (pLroDesc+i)->linkId.cellKey.blkOffset);
        printf("Segment - %lf\n", (pLroDesc+i)->linkId.cellKey.segment);
        if ((pLroDesc+i)->usObjType > 0)
        {
            printf("Object name - %s\n", (pLroDesc+i)->lro.lroInfo.objName);
            printf("Object description - %s\n", (pLroDesc+i)->lro.lroInfo.objDesc);
        }
        else
printf("Cell notes - %s", (pLroDesc+i)->lro.note);
}
printf("User name - %s", (pLroDesc+i)->userName);
printf("Security Access Level - %d\n", (pLroDesc+i)->accessLevel);
if ((pLroDesc+i)->pMemComb)
{
    printf("Member Name : \n");
    pMember = (ESS_CHAR_T *) (pLroDesc+i)->pMemComb;
    for (j=0; j < (pLroDesc+i)->memCount; j++)
    {
        printf(" %s\n", pMember);
        pMember += memberLength;
    }
    EssFree(hInst, (pLroDesc+i)->pMemComb);
}
printf("\n");
}
printf("******** complete ********\n");

exit:
if (status)
    printf("Fail Getting Catalog Information.\n");
if (ppMemComb)
{
    for(i=0; i<cellCount; i++)
    {
        EssFree(hInst, ppMemComb[i]);
    }
    EssFree(hInst, ppMemComb);
}
if (pLroDescList)
    EssFree(hInst, pLroDescList);
if (pullLroCount)
    EssFree(hInst, pullLroCount);
return(status);
}

See Also

- “LRO Constant and Structure Definitions (C)” on page 107
- EssLROGetCatalog
- EssLROAddObject
- EssLROUpdateObject
- EssLROGetObject
- EssLRODeleteObject
**EssLROGetObject**

Retrieves an object linked to a data cell in an Essbase database.

**Syntax**

```c
ESS_FUNC_M EssLROGetObject (hCtx, plinkId, targetFile, usOption, pRetLRODesc);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>plinkId</td>
<td>&quot;ESS_LROHANDLE_API_T&quot; on page 109</td>
<td>Pointer to object identification structure.</td>
</tr>
<tr>
<td>targetFile</td>
<td>ESS_STR_T</td>
<td>The name of the target file into which the object is retrieved.</td>
</tr>
<tr>
<td>usOption</td>
<td>ESS_USHORT_T</td>
<td>Option specifying whether to retrieve the object, its catalog entry, or both. Use one of the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_LRO_OBJ_API retrieves only the object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_LRO_CATALOG_API retrieves only the catalog entry.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_LRO_BOTH_API retrieves object and catalog entry.</td>
</tr>
<tr>
<td>pRetLRODesc</td>
<td>&quot;ESS_LRODESC_API_T&quot; on page 108</td>
<td>Pointer to object's description structure.</td>
</tr>
</tbody>
</table>

**Notes**

Cell notes are part of the catalog entry for an object. To retrieve a cell note, use ESS_LRO_CATALOG_API for the `usOption` parameter. The linked note is contained in structure “ESS_LRODESC_API_T” on page 108.

**Return Value**

If successful, returns ESS_STS_NOERR. Otherwise, returns an error code.

**Access**

A call to this function requires read privileges (ESS_PRIV_READ) for the active database.

**Example**

```c
ESS_FUNC_M EssLROGetObject (ESS_HCTX_T hCtx) {  
    ESS_FUNC_M              sts = ESS_STS_NOERR;
    ESS_LROHANDLE_API_T     linkId;
    ESS_LRODESC_API_T       lroDesc;
    ESS_USHORT_T            usOption = 2;  /* Default is catalog */
    ESS_CHAR_T              targetFile[ESS_ONAMELEN_API];
    memset(&lroDesc, 0, sizeof(ESS_LRODESC_API_T));
    memset(&linkId, 0, sizeof(ESS_LROHANDLE_API_T));
    /* Linked object is a LRO. (Windows Application) */
    linkId.hObject = 4;
    linkId.cellKey.cellOffset = 136;
    linkId.cellKey.blkOffset = 113.0;
    linkId.cellKey.segment = 0.0;
```
usOption = ESS_LRO_BOTH_API; /* Get the catalog and the object */
strcpy(targetFile, "c:\temp\lrofile");
sts = EssLROGetObject(hCtx, &linkId, targetFile, usOption, &lroDesc);
if (sts)
{
    printf("Could not get object\n");
}
return sts;

See Also

- “LRO Constant and Structure Definitions (C)” on page 107
- EssLROAddObject
- EssLROUpdateObject
- EssLRODeleteObject

**EssLROListObjects**

Retrieves a list of all objects linked to cells in the active database for a given user name and/or modification date.

**Syntax**

```c
ESS_FUNC_M EssLROListObjects (hCtx, userName, listDate, pulLROCount, pLRODescList);```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>userName</td>
<td>ESS_CHAR_T</td>
<td>A user name. If specified, returns a list of all objects last modified by the given user.</td>
</tr>
<tr>
<td>listDate</td>
<td>ESS_TIME_T</td>
<td>A modification date. If specified, returns a list of all objects modified before the given date. The time is a ULONG representing the number of seconds since January 1, 1970.</td>
</tr>
<tr>
<td>pulLROCount</td>
<td>ESS_ULONG_T *</td>
<td>Number of LRO catalog entries returned.</td>
</tr>
<tr>
<td>pLRODescList</td>
<td>“ESS_LRODESC_API_T” on page 108</td>
<td>Address of pointer to the list of LRO catalog entries returned.</td>
</tr>
</tbody>
</table>

**Notes**

- If you specify both the `userName` and `listDate` parameters, objects meeting both criteria are listed.
- The caller is responsible for freeing memory allocated for `pLRODescList`.

**Return Value**

If successful, returns ESS_STS_NOERR. Otherwise, returns an error code.

**Access**

A call to this function requires read privileges (ESS_PRIV_READ) to the active database.
Example

```c
ESS_FUNC_M ESS_LRO ListObjects (ESS_HCTX_T hCtx, ESS_HINST_T hInst)
{
    ESS_FUNC_M          sts = ESS_STS_NOERR;
    ESS_LRODESC_API_T   plroDescList=NULL;
    ESS_ULONG_T         ulLroCount;
    ESS_CHAR_T          userName[ESS_USERNAMELEN];
    ESS_CHAR_T          listDate[ESS_DATESIZE];
    ESS_CHAR_T          buf[ESS_DATESIZE];
    ESS_TIME_T          timestamp;
    struct tm           *pTmStruct, time_str;
    strcpy( userName, "user1");
    strcpy( listDate, "09/05/1997");
    time(&timestamp);
    pTmStruct = localtime((ESS_PLONG_T)&timestamp);
    memset(&time_str, 0, sizeof(struct tm));
    strncpy (buf, (const char *)&listDate[8], 2);
    time_str.tm_year = atoi(buf);
    strncpy(buf, listDate, 2);
    time_str.tm_mon = atoi(buf)-1;
    strncpy(buf, (const char *)&listDate[3], 2);
    time_str.tm_mday = atoi(buf);
    time_str.tm_hour = 0;
    time_str.tm_min = 0;
    time_str.tm_sec = 1;
    time_str.tm_isdst = -1;
    if ((time_str.tm_mon != pTmStruct->tm_mon) ||
        (time_str.tm_year != pTmStruct->tm_year) ||
        (time_str.tm_mday != pTmStruct->tm_mday))
    {
        time_str.tm_mday++;
        timestamp = mktime(&time_str);
    }
    sts = EssLROListObjects(hCtx, userName, timestamp, &ulLroCount, &plroDescList);
    if(sts)
    {
        printf("Could not list linked objects. \n");
    }
    if (plroDescList)
        EssFree(hInst, plroDescList);
    return sts;
}
```

See Also

- “LRO Constant and Structure Definitions (C)” on page 107
- EssLROGetCatalog
- EssLROReparseObjects

EssLROReparseObjects

Deletes all objects linked to cells in the active database for a given user name and/or modification date.
Syntax

ESST_FUNC_M EssLROPurgeObjects (hctx, userName, purgeDate, pulLROCount, pLRODescList);

Parameter | Data Type | Description
--- | --- | ---
 hCtx | ESS_HCTX_T | API context handle.
 userName | ESS_STR_T | Pointer to a user name. If specified, deletes all objects last modified by the given user.
 purgeDate | ESS_TIME_T | A modification date. If specified, returns a list of all objects modified before the given date. The date is a ULONG representing the number of seconds since January 1, 1970.
 pulLROCount | ESS_ULONG_T | Number of LRO catalog entries purged.
 pLRODescList | “ESS_LRODESC_API_T” on page 108 | Address of pointer to the list of LRO catalog entries purged.

Notes

- If you specify both the userName and purgeDate parameters, objects meeting both criteria are deleted.
- The caller is responsible for freeing memory allocated for pLRODescList.

Return Value

If successful, returns ESS_STS_NOERR. Otherwise, returns an error code.

Access

A call to this function requires design privileges (ESS_PRIV_DBDESIGN) for the active database.

Example

ESST_FUNC_M ESS_LRO PurgeObjects (ESS_HCTX_T hctx, ESS_HINST_T hInst)
{
    ESS_FUNC_M stst = ESS_STS_NOERR;
    ESS_LRODESC_API_T plroDescList=NULL;
    ESS_ULONG_T ulLroCount;
    ESS_CHAR_T userName[ESS_USERNAMELEN];
    ESS_CHAR_T purgeDate[ESS_DATESIZE];
    ESS_TIME_T timestamp;
    struct tm *pTmStruct, time_str;
    strcpy( userName, "user1");
    strcpy( purgeDate, "09/05/1997");
    time(&timestamp);
    pTmStruct = localtime((ESS_PLONG_T)&timestamp);
    memset(&time_str, 0, sizeof(struct tm));
    strncpy (buf, (const char *)&purgeDate[8], 2);
    time_str.tm_year = atoi(buf);
    strncpy(buf, listDate, 2);
    time_str.tm_mon = atoi(buf)-1;
    strncpy(buf, (const char *)&purgeDate[3], 2);
    time_str.tm_mday = atoi(buf);
    time_str.tm_hour = 0;
}
time_str.tm_min = 0;
time_str.tm_sec = 1;
time_str.tm_isdst = -1;
if ((time_str.tm_mon != pTmStruct->tm_mon) ||
    (time_str.tm_year != pTmStruct->tm_year) ||
    (time_str.tm_mday != pTmStruct->tm_mday))
{
    time_str.tm_mday++;
timestamp = mktime(&time_str);
}
sts = EssLROPurgeObjects(hCtx, userName, timestamp, &ulLroCount, &plroDescList);
if(sts)
{
    printf("Could not purge linked objects. \n");
}
if (plroDescList)
    EssFree(hInst, plroDescList);
return sts;

See Also

- “LRO Constant and Structure Definitions (C)” on page 107
- EssLROGetCatalog
- EssLRODeleteObject
- EssLRODeleteCellObjects

EssLROUpdateObject

Stores an updated version of an LRO on the server.

Syntax

`ESS_FUNC_M EssLROUpdateObject (hCtx, plinkId, usOption, pLRODesc);`

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>plinkId</td>
<td>&quot;ESS_LROHANDLE_API_T&quot; on page 109</td>
<td>Pointer to object identification structure.</td>
</tr>
<tr>
<td>usOption</td>
<td>ESS_USHORT_T</td>
<td>Option specifying whether to store the object, its catalog entry, or both. Use one of the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_LRO_OBJ_API stores only the object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_LRO_CATALOG_API stores only the catalog entry.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_LRO_BOTH_API stores the object and the catalog entry.</td>
</tr>
<tr>
<td>pLRODesc</td>
<td>&quot;ESS_LRODESC_API_T&quot; on page 108</td>
<td>Pointer to object’s description structure.</td>
</tr>
</tbody>
</table>

Notes

- The linked object can be any of the following types:
- A flat file, such as a Word document, Excel spreadsheet, or bitmap image.
- A cell note containing up to 599 characters of text.
- A link to another Essbase database (Linked Partitions feature).

Cell notes are part of the catalog entry for an object. To store a cell note, use
ESS_LRO_CATALOG_API for the usOption parameter. The linked note is contained in
structure “ESS_LRODESC_API_T” on page 108.

- The name of the last user to modify the object and the modification date are also updated.

**Return Value**

If successful, returns ESS_STS_NOERR. Otherwise, returns an error code.

**Access**

A call to this function requires write privileges (ESS_PRIV_WRITE) to the active database.

**Example**

```c
ESS_STS_T ESS_LRO UpdateObject (ESS_HCTX_T hCtx)
{
    ESS_STS_T               sts = ESS_STS_NOERR;
    ESS_LROHANDLE_API_T     linkId;
    ESS_LRODESC_API_T       lroDesc;
    ESS_USHORT_T            usOption = 2;   /* Default is catalog */

    memset (&linkId, 0, sizeof(ESS_LROHANDLE_API_T));
    memset (&lroDesc, 0, sizeof(ESS_LRODESC_API_T));

    linkId.hObject = 25;
    linkId.cellKey.cellOffset = 149;
    linkId.cellKey.blkOffset = 113.0;
    linkId.cellKey.segment = 0.0;

    /* Linked object is a LRO. (Windows Application) */
    lroDesc.usObjType = 1;
    usOption = ESS_LRO_BOTH_API;
    strcpy (lroDesc.lro.lroInfo.objName, "e:\lro\lroex.c");
    strcpy (lroDesc.lro.lroInfo.objDesc, "My C file");
    strcpy (lroDesc.userName, "user1");
    lroDesc.linkId.hObject = linkId.hObject;

    sts = EssLROUpdateObject(hCtx, &linkId, usOption, &lroDesc);
    if (sts)
    {
        printf("Could not update linked object.\n");
    }
    return sts;
}
```

**See Also**

- “LRO Constant and Structure Definitions (C)” on page 107
EssMdxTrig

Manipulates triggers based on the operations specified in an MDX statement. The MDX can create, replace, delete, enable, or disable a specific trigger.

Syntax

```c
ESS_FUNC_M EssMdxTrig (hCtx, AppName, DbName, mdxStatement);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle</td>
</tr>
<tr>
<td>AppName</td>
<td>ESS_STR_T</td>
<td>Application name</td>
</tr>
<tr>
<td>DbName</td>
<td>ESS_STR_T</td>
<td>Database name</td>
</tr>
<tr>
<td>mdxStatement</td>
<td>ESS_STR_T</td>
<td>An MDX statement that specifies whether to create, replace, delete, enable, or disable a specific trigger.</td>
</tr>
</tbody>
</table>

Access

This function requires the caller to have database Design privilege (ESS_PRIV_DBDESIGN) for the specified database.

See Also

- EssDisplayTriggers
- EssListSpoolFiles
- EssGetSpoolFile
- EssDeleteSpoolFile
- EssDeleteAllSpoolFiles

EssMergeDatabaseData

Merges two or more data slices into a single data slice. Optionally, the primary database slice can be excluded.

This function applies only to aggregate storage databases.

Syntax

```c
ESS_FUNC_M EssMergeDatabaseData (hCtx, AppName, DbName, ulOptions);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle</td>
</tr>
<tr>
<td>AppName</td>
<td>ESS_STR_T</td>
<td>Use NULL. Function always applies to the currently selected database.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DbName</td>
<td>ESS_STR_T</td>
<td>Use NULL. Function always applies to the currently selected database.</td>
</tr>
<tr>
<td>ulOptions</td>
<td>ESS_ULONG_T</td>
<td>One of the following constants:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- #define ESS_MERGE_DATABASE_DATA_ALL 1: Merges all data slices into one.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- #define ESS_MERGE_DATABASE_DATA_INCREMENTAL 2: Merges all incremental slices into one slice, but does not merge this slice with the primary slice. Afterwards, there will be two slices.</td>
</tr>
</tbody>
</table>

**Return Value**

Returns zero if successful; otherwise, returns an error code.

**Example**

```c
void TestMergeDatabaseData(ESS_HCTX_T hCtx, ESS_STR_T AppName, ESS_STR_T DbName)
{
    ESS_STS_T               sts = ESS_STS_NOERR;
    ESS_SHORT_T             isAbortOnError;
    ESS_OBJDEF_T    Rules;
    ESS_OBJDEF_T    Data;
    ESS_PMBRERR_T   pMbrErr = NULL;
    ESS_PMBRUSER_T  pMbrUser = NULL;
    ESS_ULONG_T             ulBufferId;
    ESS_ULONG_T             ulDuplicateAggregationMethod;
    ESS_ULONG_T             ulOptionsFlags;
    ESS_ULONG_T             ulSize;
    ESS_ULONG_T             ulBufferCnt;
    ESS_ULONG_T             ulCommitType;
    ESS_ULONG_T             ulActionType;
    ESS_ULONG_T             ulOptions;
    ESS_ULONG_T             ulBufferIdAry[1];
    ESS_ULONG_T             options;

    printf("Create the buffer:
    ulDuplicateAggregationMethod = ESS_ASO_DATA_LOAD_BUFFER_DUPLICATES_ADD;
    ulOptionsFlags = ESS_ASO_DATA_LOAD_BUFFER_IGNORE_MISSING_VALUES;
    ulSize = 100;
    ulBufferId = 1;
    sts = EssLoadBufferInit(hCtx, AppName, DbName, ulBufferId,
    ulDuplicateAggregationMethod,
    ulOptionsFlags, ulSize);
    printf("EssLoadBufferInit sts: %ld\n", sts);

    /* Server object */
    Rules.hCtx     = hCtx;
    Rules.AppName  = AppName;
    Rules.DbName   = DbName;
    Rules.ObjType  = ESS_OBJTYPE_RULES;
    Rules.FileName = "ddldinaq";
    Data.hCtx = hCtx;
    Data.AppName = AppName;
    Data.DbName = DbName;
    Data.ObjType = ESS_OBJTYPE_TEXT;
    Data.FileName = "ddldinaq_slicela";
```
isAbortOnError = ESS_TRUE;

printf("Load into buffer:\n");
sts = EssImportASO (hCtx, &Rules, &Data, &pMbrErr, pMbrUser, isAbortOnError, ulBufferId);
printf("EssImportASO sts: %ld\n",sts);
if(pMbrErr)
    EssFreeMbrErr(hCtx, pMbrErr);

ulBufferCnt = 1;
ulBufferIdAry[0] = ulBufferId;
ulCommitType = ESS_ASO_DATA_LOAD_BUFFER_STORE_DATA;
ulActionType = ESS_ASO_DATA_LOAD_BUFFER_COMMIT;
printf("Create a new slice:\n");
ulOptions = ESS_ASO_DATA_LOAD_INCR_TO_NEW_SLICE;
sts = EssLoadBufferTerm(hCtx, AppName, DbName, ulBufferCnt, ulBufferIdAry, ulCommitType, ulActionType, ulOptions);
printf("EssLoadBufferTerm sts: %ld\n",sts);

options = ESS_MERGE_DATABASE_DATA_ALL;
printf("Merge all data into one slice:\n");
sts = EssMergeDatabaseData(hCtx, AppName, DbName, options);
printf("EssMergeDatabaseData sts: %ld\n",sts);
}

See Also
- EssLoadBufferInit
- EssBeginDataloadASO
- EssSendString
- EssEndDataload
- EssLoadBufferTerm
- EssImportASO
- EssUpdateFileASO
- EssUpdateFileUTF8ASO
- EssListExistingLoadBuffers

EssPartialDataClear

Clears the data specified in a well-defined, symmetrical region in the active aggregate storage database. There are two methods for selectively clearing data from a region:

- Physical, in which the input cells in the specified region are physically removed from the aggregate storage database. The process for physically clearing data completes in a length of time that is proportional to the size of the input data, not the size of the data being cleared.

- Logical, in which the input cells in the specified region are written to a new data slice with negative, compensating values that result in a value of zero for the cells you want to clear. The process for logically clearing data completes in a length of time that is proportional to the size of the data being cleared.
Syntax

ESS_FUNC_M EssPartialDataClear (hCtx, RegionSpec, bPhysical);

Parameter | Data Type | Description
--- | --- | ---

hCtx | ESS_HCTX_T | API context handle (logged in)

RegionSpec | ESS_STR_T | Region specification (a valid MDX set expression)
The region must be symmetrical. Members in any dimension in the region must be stored members. When physically clearing data, members in the region can be upper-level members from primary and alternate hierarchies. (If the region contains upper-level members from alternate hierarchies, you may experience a decrease in performance.) When logically clearing data, members in the region can be upper level members from the primary hierarchy only. Members cannot be dynamic members (members with implicit or explicit MDX formulas), nor can they be from an attribute dimension.

bPhysical | ESS_BOOL_T | If TRUE, specifies clearing the data in the region using the physical clear region operation. If FALSE or not specified, data is cleared using the logical clear region operation.

Notes

- The caller must have Database Manager or Administrator permission to clear data.

Return Value

Return value for this function is zero upon successful completion; otherwise, an error code is returned.

Access

This function applies to aggregate storage databases only.

Example

ESS_FUNC_M
TestPartialDataClear(ESS_HCTX_T hCtx)
{
    ESS_STS_T       sts;
    ESS_STR_T       regionSpec="{Feb}";

    /* Perform a logical clear of February data */
    sts = EssPartialDataClear(hCtx, regionSpec, AD_FALSE);
    return(sts);
}

EssPartitionApplyOtlChangeFile

Replaced by EssPartitionApplyOtlChangeFileEx, but this format is maintained for backward compatibility. For complete information, see EssPartitionApplyOtlChangeFileEx.

Syntax

ESS_FUNC_M EssPartitionApplyOtlChangeFile (hCtx, usFileName, ppszFileName);
EssPartitionApplyOtlChangeFileEx

Applies outline change files (*.CHG) on the source to a target outline. This function is designed to be used in batch with EssPartitionGetOtlChanges() and can specify a list of change files. This function can be used with filters.

Use this function instead of EssPartitionApplyOtlChangeFile() whenever there exists more then one partition of the same type and the same metadata direction between the application/database pair.

Syntax

```c
ESS_FUNC_M EssPartitionApplyOtlChangeFileEx (hCtx, usFileName, ppszFileName, usDataDirectionType);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>Handle to Essbase API context</td>
</tr>
<tr>
<td>usFileName</td>
<td>ESS_USHORT_T</td>
<td>Number of outline change files</td>
</tr>
<tr>
<td>ppszFileName</td>
<td>ESS_PSTR_T</td>
<td>Array of file names; array size is defined by usFileName</td>
</tr>
<tr>
<td>usDataDirectionType</td>
<td>ESS_USHORT_T</td>
<td>One of the following Direction Type constants:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>#define ESS_PARTITION_DATA_SOURCE 0x0001</td>
</tr>
<tr>
<td></td>
<td></td>
<td>#define ESS_PARTITION_DATA_TARGET 0x0002</td>
</tr>
</tbody>
</table>

Notes

EssPartitionGetOltChanges() returns the name of the change file.

Return Value

Returns zero if successful; error code if unsuccessful.

Access

A call to this function requires database designer permission.

Example

```c
ESS_FUNC_M ESS_PartitionApplyOtlChangeFileEx (ESS_HCTX_T hCtx, ESS_HINST_T hInst) {
    ESS_FUNC_M          sts;
    ESS_STR_T          hostname, appname, dbname;
    ESS_USHORT_T       usType, uscnt, dataFlowDir, *dataFlowDirs = ESS_NULL;
    ESS_ULONG_T        uldimfilter=0, ulmbrfilter=0, ulmbrattrfilter=0;
    ESS_PARTOTL_QUERY_T MetaQuery;
```
ESS_PARTOTL_CHG_FILE_T MetaChangeFile;
ESS_PPART_INFO_T partitionp = NULL;

memset(&MetaQuery, 0, sizeof(ESS_PARTOTL_QUERY_T));

hostname = "local";
appname = "appl";
dbname = "src1";
usType = ESS_PARTITION_OP_LINKED;
dataFlowDir = ESS_PARTITION_DATA_SOURCE;
uldimfilter = ESS_DIMCHG_ALL;
ulmbrfilter = ESS_PARTITION_OTLMBR_ALL;
ulmbrattrfilter = ESS_PARTITION_OTLMBR_ATTR_ALL;

MetaQuery.HostDatabase.pszHostName = hostname;
MetaQuery.HostDatabase.pszAppName = appname;
MetaQuery.HostDatabase.pszDbName = dbname;
MetaQuery.usOperationType = usType;
MetaQuery.usDataDirectionType = dataFlowDir;
MetaQuery.MetaFilter.TimeStamp = 0;
MetaQuery.MetaFilter.ulDimFilter = uldimfilter;
MetaQuery.MetaFilter.ulMbrFilter = ulmbrfilter;
MetaQuery.MetaFilter.ulMbrAttrFilter = ulmbrattrfilter;

sts = EssPartitionGetOtlChanges(hCtx, &MetaQuery, &MetaChangeFile);

if (!sts)
    sts = EssAlloc(hInst, MetaChangeFile.usFileNum * sizeof(ESS_USHORT_T), &dataFlowDirs);

if (!sts)
    for (uscnt=0; uscnt < MetaChangeFile.usFileNum; uscnt++)
        dataFlowDirs[uscnt] = dataFlowDir;

if (!sts)
{
    sts = EssPartitionApplyOtlChangeFile
         (hCtx, MetaChangeFile.usFileNum, MetaChangeFile.ppszFileName);

    printf("EssPartitionApplyOtlChangeFile  sts: %ld\n", sts);
}

if(&MetaChangeFile) EssFree(hInst, &MetaChangeFile);
if(&dataFlowDirs) EssFree(hInst, &dataFlowDirs);

return(sts);

See Also

- “Constant and Structure Definitions for Partitions (C)” on page 109
- EssPartitionApplyOtlChangeRecs
- EssPartitionCloseDefFile
- EssPartitionFreeDefCtx
EssPartitionApplyOtlChangeRecs

Applies outline changes to a target outline. This function is designed to be used interactively with EssPartitionReadOtlChangeFile() after a call to EssPartitionGetOtlChanges(). The change file returned by EssPartitionReadOtlChangeFile() can be edited to set the reject flags. The reject flags are set in “ESS_PARTOTL_MBR_RSRVD_API_T” on page 164, which is referenced from ESS_PARTOTL_SELECT_APPLY_T.

Syntax

ESS_FUNC_M EssPartitionApplyOtlChangeRecs (hCtx, pApplyRecords);

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>Handle to API context.</td>
</tr>
<tr>
<td>pApplyRecords</td>
<td>“ESS_PARTOTL_SELECT_APPLY_T” on page 171</td>
<td>Records to apply.</td>
</tr>
</tbody>
</table>

Notes

- There may be dependencies among change records.
- Rejecting a record may cause a failure when applying another record. For example, you have two records "add A" and "add AA as a child of A". Rejecting the first record and accepting the second causes an apply failure.

Return Value

Returns zero if successful; error code if unsuccessful.

Access

A call to this function requires database designer access privileges.
Example

```c
ESS_FUNC_M Ess_ParticleApplyOtlChangeRecs (ESS_HCTX_T hCtx)
{
    ESS_FUNC_M                   sts = ESS_STS_NOERR;
    ESS_PARTOTL_SELECT_APPLY_T   ApplyRecords;
    ESS_STR_T                    chgfilename;
    ESS_TIME_T                   time = 0;
    ESS_PARTOTL_CHANGE_API_T     OtlChg;
    ESS_ULONG_T                  uldimfilter=0,ulmbrfilter=0,ulmbrattrfilter=0;
    ESS_PARTOTL_SELECT_CHG_T     SelectMetaRecords;
    ESS_PARTOTL_READ_T           MetaChangeRead;

    memset(&ApplyRecords,      0, sizeof(ESS_PARTOTL_SELECT_APPLY_T));
    memset(&SelectMetaRecords, 0, sizeof(ESS_PARTOTL_SELECT_CHG_T));
    memset(&MetaChangeRead,    0, sizeof(ESS_PARTOTL_READ_T));

    chgfilename = "C:\\Hyperion\\products\\Essbase\\EssbaseServer\\app\\app1\\trg1\\ess00001.chg";
    uldimfilter      = ESS_DIMCHG_ALL;
    ulmbrfilter      = ESS_PARTITION_OTLMBR_ALL;
    ulmbrattrfilter  = ESS_PARTITION_OTLMBRATTR_ALL;

    SelectMetaRecords.pszFileName                 = chgfilename;
    SelectMetaRecords.QueryFilter.TimeStamp       = time;
    SelectMetaRecords.QueryFilter.ulDimFilter     = uldimfilter;
    SelectMetaRecords.QueryFilter.ulMbrFilter     = ulmbrfilter;
    SelectMetaRecords.QueryFilter.ulMbrAttrFilter = ulmbrattrfilter;
    MetaChangeRead.pOtlChg = &OtlChg;
    sts = EssPartitionReadOtlChangeFile (hCtx, &SelectMetaRecords, &MetaChangeRead);
    printf("EssPartitionReadOtlChangeFile  sts: %ld\n",sts);
    if (!sts)
    {
        ApplyRecords.pszFileName = chgfilename;
        ApplyRecords.pOtlChg = MetaChangeRead.pOtlChg;
        ApplyRecords.SourceTime = MetaChangeRead.SourceTime;
        sts = EssPartitionApplyOtlChangeRecs(hCtx, &ApplyRecords);
        printf("EssPartitionApplyOtlChangeRecs(sts: %ld\n",sts);
    }
    sts = EssPartitionFreeOtlChanges(hCtx);
    return(sts);
}
```

See Also

- "Constant and Structure Definitions for Partitions (C)" on page 109
- EssPartitionApplyOtlChangeFile
- EssPartitionCloseDefFile
- EssPartitionFreeDefCtx
- EssPartitionFreeOtlChanges
- EssPartitionGetAreaCellCount
- EssPartitionGetList
- EssPartitionGetOtlChanges
EssPartitionCloseDefFile

Closes the shared partition definition file.

**Syntax**

```c
ESS_FUNC_M EssPartitionCloseDefFile (hCtx, iFileHandle);
```

**Parameter** | **Data Type** | **Description**
--- | --- | ---
`hCtx` | ESS_HCTX_T | Net context.
`iFileHandle` | ESS_INT_T | File handle to close.

**Notes**

- Use this function as part of a sequence of definition operations. Use `EssPartitionOpenDefFile()` to open existing definition files. Use `EssPartitionNewDefFile()` to create and open a new definition file. Use `EssPartitionReadDefFile()` or `EssPartitionWriteDefFile()` to read or write a definition file. Close with `EssPartitionCloseDefFile()`. Then free the memory with `EssPartitionFreeDefCtx()`.

**Return Value**

Returns zero if successful; error code if unsuccessful.

**Example**

For an example, see `EssPartitionNewDefFile`

**See Also**

- “Constant and Structure Definitions for Partitions (C)” on page 109
- `EssPartitionApplyOtlChangeFile`
- `EssPartitionApplyOtlChangeRecs`
- `EssPartitionFreeDefCtx`
- `EssPartitionFreeOtlChanges`
- `EssPartitionGetAreaCellCount`
- `EssPartitionGetList`
EssPartitionFreeDefCtx

Frees memory dynamically allocated under shared-partition context structures.

Syntax

```c
ESS_FUNC_M EssPartitionFreeDefCtx (hCtx, pDdbCtx);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>Api context.</td>
</tr>
<tr>
<td>pDdbCtx</td>
<td>“ESS_PART_T” on page 152</td>
<td>Pointer to shared-partition context.</td>
</tr>
</tbody>
</table>

Notes

- Use this function as part of a sequence of definition operations. Use `EssPartitionOpenDefFile()` to open existing definition files. Use `EssPartitionNewDefFile()` to create and open a new definition file. Use `EssPartitionReadDefFile()` or `EssPartitionWriteDefFile()` to read or write a definition file. Close with `EssPartitionCloseDefFile()`. Then free the memory with `EssPartitionFreeDefCtx()`.

Return Value

Returns zero if successful, error code if unsuccessful.

Example

For an example, see `EssPartitionNewDefFile`.

See Also

- “Constant and Structure Definitions for Partitions (C)” on page 109
- `EssPartitionAp plyOtlChangeFile`
- `EssPartitionApplyOtlChangeRecs`
- `EssPartitionFreeDefCtx`
- `EssPartitionFreeOtlChanges`
EssPartitionFreeOtlChanges

Frees memory allocated by the EssPartitionReadOtlChanges() routine. Call this routine after processing outline change records.

Syntax

ESS_FUNC_M EssPartitionFreeOtlChanges ( hCtx );

Parameter  Data Type  Description

hCtx  ESS_HCTX_T  Handle to API context.

Return Value

Returns zero if successful; error code if unsuccessful.

Example

For an example, see EssPartitionReadOtlChangeFile.

See Also

- “Constant and Structure Definitions for Partitions (C)” on page 109
- EssPartitionApplyOtlChangeFile
- EssPartitionApplyOtlChangeRecs
- EssPartitionCloseDefFile
- EssPartitionFreeDefCtx
- EssPartitionGetAreaCellCount
- EssPartitionGetList
- EssPartitionGetOtlChanges
- EssPartitionGetReplCells
- EssPartitionNewDefFile
- EssPartitionOpenDefFile
- EssPartitionPurgeOtlChangeFile
- EssPartitionPutReplCells
- EssPartitionReadDefFile
- EssPartitionReadOtlChangeFile
- EssPartitionReplaceDefFile
- EssPartitionResetOtlChangeTime
- EssPartitionValidateDefinition
- EssPartitionValidateLocal
- EssPartitionWriteDefFile
EssPartitionPutReplCells
EssPartitionReadDefFile
EssPartitionReadOtlChangeFile
EssPartitionReplaceDefFile
EssPartitionResetOtlChangeTime
EssPartitionValidateDefinition
EssPartitionWriteDefFile

EssPartitionGetAreaCellCount

Returns the number of cells in the specified slice string.

Syntax

ESS_FUNC_M EssPartitionGetAreaCellCount(hCtx, pszSlice, pdCount);

Parameter | Data Type | Description
---|---|---
| hCtx | ESS_HCTX_T | API context handle.
| pszSlice | ESS_STR_T | Input slice definition to be checked.
| pdCount | ESS_PDOUBLE_T | Returns number of cells here.

Return Value

Returns zero if successful; error code if unsuccessful.

Example

ESS_FUNC_M ESS_PartitionGetAreaCellCount(ESS_HCTX_T hCtx)
{
    ESS_FUNC_M sts = ESS_STS_NOERR;
    ESS_DOUBLE_T pdCount;
    ESS_STR_T pszSlice;

    pszSlice = "@IDESC(East)";

    sts = EssPartitionGetAreaCellCount(hCtx, pszSlice, &pdCount);
    if (!sts)
    {
        printf("EssPartitionGetAreaCellCount  sts: %ld\n",sts);
        printf("\tArea cell count = %.g \n",pdCount);
    }
    return(sts);
}

See Also

- “Constant and Structure Definitions for Partitions (C)” on page 109
- EssPartitionApplyOtlChangeFile
- EssPartitionApplyOtlChangeRecs
- EssPartitionCloseDefFile
- EssPartitionFreeDefCtx
- EssPartitionFreeOtlChanges

542  C Main API Functions
EssPartitionGetAreaLev0CellCount

Returns the number of cells which are level0 combinations of dimensions in a specified slice string. This is useful if the target of replicated partition is an ASO cube.

Syntax

ESS_FUNC_M EssPartitionGetAreaLev0CellCount (hCtx, pszSlice, pdCount);

Parameter | Data Type | Description
--- | --- | ---
hCtx | ESS_HCTX_T | API context handle.
pszSlice | ESS_STR_T | Input slice definition to be checked.
pdCount | ESS_PDOUBLE_T | Returns number of cells here.

Return Value

Returns zero if successful; error code if unsuccessful.

Example

ESS_FUNC_M ESS_PartitionGetAreaLev0CellCount (ESS_HCTX_T hCtx)
{
    ESS_FUNC_M   sts = ESS_STS_NOERR;
    ESS_DOUBLE_T  pdCount;
    ESS_STR_T    pszSlice;

    pszSlice = "@IDESC(East)";

    sts = EssPartitionGetAreaLev0CellCount(hCtx, pszSlice, &pdCount);
    if (!sts)
    {
        printf("EssPartitionGetAreaLev0CellCount  sts: %ld\n",sts);
        printf("\tArea cell count = %g \n",pdCount);
    }

    return(sts);
}
See Also

- “Constant and Structure Definitions for Partitions (C)” on page 109
- EssPartitionApplyOtlChangeFile
- EssPartitionApplyOtlChangeRecs
- EssPartitionCloseDefFile
- EssPartitionFreeDefCtx
- EssPartitionFreeOtlChanges
- EssPartitionGetList
- EssPartitionGetOtlChanges
- EssPartitionGetReplCells
- EssPartitionNewDefFile
- EssPartitionOpenDefFile
- EssPartitionPurgeOtlChangeFile
- EssPartitionPutReplCells
- EssPartitionReadDefFile
- EssPartitionReadOtlChangeFile
- EssPartitionReplaceDefFile
- EssPartitionResetOtlChangeTime
- EssPartitionValidateDefinition
- EssPartitionWriteDefFile

EssPartitionGetList

Returns a list of the partition definitions in which the currently selected database participates.

Syntax

```
ESS_FUNC_M EssPartitionGetList (hCtx, pSelectPartition, pusCount, ppPartition);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>pSelectPartition</td>
<td>“ESS_PARTSLCT_T” on page 171</td>
<td>Criteria to select partitions.</td>
</tr>
<tr>
<td>pusCount</td>
<td>ESS_PUSHORT_T</td>
<td>Count of partitions returned.</td>
</tr>
<tr>
<td>ppPartition</td>
<td>“ESS_PART_INFO_T” on page 154</td>
<td>Pointer to allocated array of partition information structures.</td>
</tr>
</tbody>
</table>

Return Value

Returns zero if successful; error code if unsuccessful.

Example

```
ESS_FUNC_M ESS_PartitionGetList(ESS_HCTX_T hCtx, ESS_HINST_T hInst)
{
    ESS_FUNC_M           sts       = ESS_STS_NOERR;
    ESS_USHORT_T         op_types  = 0;
    ESS_USHORT_T         dir_types = 0;
    ESS_USHORT_T         meta_dir_types = 0;
    ESS_USHORT_T         count, i;
```
ESS_PPART_INFO_T partitionp = NULL;
ESS_PARTSLCT_T SelectPartition;
memeze(&Selectpartition, 0, sizeof(ESS_PARTSLCT_T));

op_types = ESS_PARTITION_OP_REPLICATED |
ESS_PARTITION_OP_LINKED |
ESS_PARTITION_OP_TRANSPARENT;

dir_types = ESS_PARTITION_DATA_SOURCE | ESS_PARTITION_DATA_TARGET;

meta_dir_types = ESS_PARTITION_OTL_SOURCE | ESS_PARTITION_OTL_TARGET;

SelectPartition.usOperationTypes = op_types;
SelectPartition.usDirectionTypes = dir_types;
SelectPartition.usMetaDirectionTypes = meta_dir_types;

sts = EssPartitionGetList(hCtx, &SelectPartition, &count, &Partitionp);
printf("EssPartitionGetList  sts:  %ld\n",sts);
if (!sts)
{
  printf("\n# Partitions matching input criteria: %d\n
", (int)count);
  for (i = 0; i < count; i++)
  {
    ESS_PART_INFO_T *info = &partitionp[i];

    printf("%2d: %s %s %s: Host=%s App=%s Db=%s\n", i+1,
           info->OperationType==ESS_PARTITION_OP_REPLICATED ? "Replication":
           info->OperationType==ESS_PARTITION_OP_LINKED ? "Link":
           info->OperationType==ESS_PARTITION_OP_TRANSPARENT ? "Transparent":
           "Unknown",
           info->DataDirection==ESS_PARTITION_DATA_SOURCE ? "Source":
           info->DataDirection==ESS_PARTITION_DATA_TARGET ? "Target":
           "(Unknown Outline Change Target)" :
           "(Unknown Outline Change Type)",
           info->SvrName, info->AppName, info->DbName);
    printf("    Outline last changed: %s\n",
           info->LastMetaUpdateTime==0 ? "Never":
           ctime(&info->LastMetaUpdateTime));

    if (info->OperationType==ESS_PARTITION_OP_REPLICATED &&
        info->DataDirection==ESS_PARTITION_DATA_TARGET)
    {
      printf("      Last replicated: %s %s\n",
             info->LastRefreshTime==0 ? "Never":
             ctime(&info->LastRefreshTime),
             info->PartitionUpdatable ? "Locally updatable":
             "Not locally updatable");
    }
    else if (info->OperationType==ESS_PARTITION_OP_REPLICATED &&
             info->DataDirection==ESS_PARTITION_DATA_SOURCE)
    {
      printf("      Last updated: %s %s\n",
             info->LastUpdateTime==0 ? "Never":
             ctime(&info->LastUpdateTime),
             info->IncrRefreshAllowed ? "Incrementally replicatable":
             "Not incrementally replicatable");
    }
EssPartitionGetOtlChanges

Reads outline changes from a .CHG file on a source server and writes them to a .CHG file on the target server. This function is designed to be used in a batch with EssPartitionApplyOtlChangeFile(), or interactively with a combination of EssPartitionReadOtlChangeFile() and EssPartitionApplyOtlChangeRecs().

Syntax

ESS_FUNC_M EssPartitionGetOtlChanges (hCtx, pQuery, pChangeFile);

Parameter | Data Type | Description
---|---|---
hCtx | ESS_HCTX_T | API context handle.
pQuery | “ESS_PARTOTL_QUERY_T” on page 169 | Change query criteria.
pChangeFile | “ESS_PARTOTL_CHG_FILE_T” on page 160 | Caller allocated change files information structure.

Notes

- Call EssPartitionFreeOtlChanges() to free change file name strings in pChangeFile.
**Return Value**

Returns zero if successful; error code if unsuccessful.

**Access**

A call to this function requires database designer access privileges.

**Example**

```c
ESS_FUNC_M Ess_PartitionGetOtlChanges(ESS_HCTX_T hCtx, ESS_HINST_T hInst)
{
    ESS_FUNC_M             sts;
    ESS_STR_T              hostname, appname, dbname;
    ESS_USHORT_T           usType, dataFlowDir;
    ESS_ULONG_T            uldimfilter,ulmbrfilter,ulmbrattrfilter;
    ESS_PARTOTL_QUERY_T    MetaQuery;
    ESS_PARTOTL_CHG_FILE_T MetaChangeFile;
    ESS_PPART_INFO_T       partitionp = NULL;
    memset(&MetaQuery, 0, sizeof(ESS_PARTOTL_QUERY_T));

    hostname = "local";
    appname = "app1";
    dbname = "src1";
    usType = ESS_PARTITION_OP_LINKED;
    dataFlowDir = ESS_PARTITION_DATA_SOURCE;
    uldimfilter = ESS_PARTITION_OTLDIM_ALL;
    ulmbrfilter = ESS_PARTITION_OTLMBR_ALL;
    ulmbrattrfilter = ESS_PARTITION_OTLMBRATTR_ALL;
    MetaQuery.HostDatabase.pszHostName   = hostname;
    MetaQuery.HostDatabase.pszAppName    = appname;
    MetaQuery.HostDatabase.pszDbName     = dbname;
    MetaQuery.usOperationType            = usType;
    MetaQuery. usDataDirectionType       = dataFlowDir;
    MetaQuery.MetaFilter.TimeStamp       = 0;
    MetaQuery.MetaFilter.ulDimFilter     = uldimfilter;
    MetaQuery.MetaFilter.ulMbrFilter     = ulmbrfilter;
    MetaQuery.MetaFilter.ulMbrAttrFilter = ulmbrattrfilter;

    sts = EssPartitionGetOtlChanges(hCtx, &MetaQuery, &MetaChangeFile);
    printf("EssPartitionGetOtlChanges   sts: %ld\n",sts);
    if (!sts) {
        printf("\tNumber of meta change file found: %d\n",MetaChangeFile.usFileNum);
        printf("\tName of meta change file found: %s\n",MetaChangeFile.ppszFileName[0]);
    }
    if(&MetaChangeFile) EssFree(hInst,&MetaChangeFile);

    return(sts);
}
```

**See Also**

- “Constant and Structure Definitions for Partitions (C)” on page 109
- EssPartitionApplyOtlChangeFile
- EssPartitionApplyOtlChangeRecs
- EssPartitionCloseDefFile
- EssPartitionFreeDefCtx
EssPartitionGetReplCells

Replicates all data cells that are identified in the replication partition from the source database to the selected target database.

Syntax

ESS_FUNC_M EssPartitionGetReplCells (hCtx, pReplicatePartition);

Parameter | Data Type | Description
--- | --- | ---
hCtx | ESS_HCTX_T | API context handle.
pReplicatePartition | “ESS_PART_REPL_T” on page 155 | Partition information.

Return Value

Returns zero if successful; error code if unsuccessful.

Access

A call to this function requires database designer access privileges.

Example

ESS_FUNC_M Ess_PartitionGetReplCells(ESS_HCTX_T hCtx) {
    ESS_FUNC_M sts;
    ESS_PART_REPL_T ReplicatePartition;
    ESS_PART_CONNECT_INFO_T HostDatabase;

    memset(&ReplicatePartition, 0, sizeof(ESS_PART_REPL_T));
    memset(&HostDatabase, 0, sizeof(ESS_PART_CONNECT_INFO_T));

    ReplicatePartition.pHostDatabase = &HostDatabase;

    ReplicatePartition.lPartitionCount = -1;
    ReplicatePartition.bUpdatedOnly = ESS_FALSE;
}
sts = EssPartitionGetReplCells(hCtx, &ReplicatePartition);
printf("EssPartitionGetReplCells  sts: \%ld\n",sts);
return(sts);
}

See Also

- "Constant and Structure Definitions for Partitions (C)" on page 109
- EssPartitionApplyOtlChangeFile
- EssPartitionApplyOtlChangeRecs
- EssPartitionCloseDefFile
- EssPartitionFreeDefCtx
- EssPartitionGetAreaCellCount
- EssPartitionGetList
- EssPartitionGetOtlChanges
- EssPartitionOpenDefFile
- EssPartitionPurgeOtlChangeFile
- EssPartitionPutReplCells
- EssPartitionReadDefFile
- EssPartitionReadOtlChangeFile
- EssPartitionReplaceDefFile
- EssPartitionResetOtlChangeTime
- EssPartitionValidateDefinition
- EssPartitionWriteDefFile

**EssPartitionNewDefFile**

Creates and opens a new shared-partition definition file based upon input parameters supplied.

**Syntax**

```c
ESS_FUNC_M EssPartitionNewDefFile (hCtx, pszFileName, pHostDatabase, piFileHandle, ppDdbCtx);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API network context.</td>
</tr>
<tr>
<td>pszFileName</td>
<td>ESS_STR_T</td>
<td>Name of file to be created (full path).</td>
</tr>
<tr>
<td>pHostDatabase</td>
<td>&quot;ESS_PART_CONNECT_INFO_T&quot; on page 153</td>
<td>Identifies the host database.</td>
</tr>
<tr>
<td>piFileHandle</td>
<td>ESS_PINT_T</td>
<td>Handle to created file.</td>
</tr>
<tr>
<td>ppDdbCtx</td>
<td>&quot;ESS_PART_T&quot; on page 152</td>
<td>An initialized distributed context.</td>
</tr>
</tbody>
</table>

**Return Value**

Returns zero if successful; error code if unsuccessful.
Example

```c
ESS_FUNC_M ESS_PartitionNewDefFile(ESS_HCTX_T hCtx)
{
    ESS_FUNC_M     sts = 0;
    ESS_INT_T     iFileHandle;
    ESS_STR_T     pszFileName;
    ESS_PART_T    *pDdbCtx;
    ESS_STR_T     hostname, appname, dbname;
    ESS_PART_CONNECT_INFO_T   HostDatabase;
    pszFileName = "C:\\Hyperion\\products\\Essbase\\EssbaseServer\\app\\app1\\trg1\\trg1.ddb";
    hostname = "local";
    appname = "app1";
    dbname = "dbname";
    HostDatabase.pszHostName   = hostname;
    HostDatabase.pszAppName    = appname;
    HostDatabase.pszDbName     = dbname;
    sts = ESS_PartitionNewDefFile(hCtx,pszFileName,&HostDatabase,&iFileHandle,&pDdbCtx);
    printf("EssPartitionNewDefFile  sts: %ld\n",sts);
    if (!sts)
    {
        /* ...
           ...  process definition file information
           ... */
        sts = ESS_PartitionWriteDefFile(hCtx,iFileHandle,pDdbCtx);
        printf("\tESS_PartitionWriteDefFile  sts: %ld\n",sts);
        sts = ESS_PartitionCloseDefFile(hCtx,iFileHandle);
        printf("\tESS_PartitionCloseDefFile  sts: %ld\n",sts);
        sts = ESS_PartitionFreeDefCtx(hCtx,pDdbCtx);
        printf("\tESS_PartitionFreeDefCtx  sts: %ld\n",sts);
    }
    return (sts);
}
```

See Also

- “Constant and Structure Definitions for Partitions (C)” on page 109
- `EssPartitionApplyOtlChangeFile`
- `EssPartitionApplyOtlChangeRecs`
- `EssPartitionCloseDefFile`
- `EssPartitionFreeDefCtx`
- `EssPartitionFreeOtlChanges`
- `EssPartitionGetAreaCellCount`
- `EssPartitionGetList`
- `EssPartitionGetOtlChanges`
 EssPartitionOpenDefFile

Opens an existing shared-partition definition file.

Syntax

ESS_FUNC_M EssPartitionOpenDefFile (hCtx, pszFileName, piFileHandle, ppDdbCtx);

Parameter  Data Type  Description
hCtx    ESS_HCTX_T  API context handle.
pszFileName  ESS_STR_T  Name of file to be opened (complete path).
piFileHandle  ESS_PINT_T  Handle to created file.
ppDdbCtx  "ESS_PART_T" on page 152  An initialized distributed context.

Return Value

Returns zero if successful; error code if unsuccessful.

Example

ESS_FUNC_M ESS_PartitionOpenDefFile(ESS_HCTX_T hCtx)
{
    ESS_FUNC_M    sts = 0;
    ESS_INT_T    iFileHandle;
    ESS_STR_T    pszFileName;
    ESS_PART_T    DdbCtx, *pDdbCtx;
    pszFileName = "C:\\Hyperion\\products\\Essbase\\EssbaseServer\\app\\appl\\trg1\\trg1.ddb";
    sts = EssPartitionOpenDefFile(hCtx,pszFileName,&iFileHandle,&pDdbCtx);
    printf("EssPartitionOpenDefFile  sts: %ld\n",sts);
    if (!sts)
    {
        sts = EssPartitionReadDefFile(hCtx,iFileHandle,&DdbCtx);
        printf("\tEssPartitionReadDefFile  sts: %ld\n",sts);
        /* ... 
        ...  process definition file information
    }
}
EssPartitionPurgeOtlChangeFile

Purges changes made previous to the time specified with the TimeStamp parameter.

Syntax

ESS_FUNC_M EssPartitionPurgeOtlChangeFile (hCtx, pPartition, TimeStamp);

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>pPartition</td>
<td>“ESS_PART_DEFINED_T” on page 153</td>
<td>Partition specification</td>
</tr>
<tr>
<td>TimeStamp</td>
<td>ESS_TIME_T</td>
<td>Purge all change records before this time.</td>
</tr>
</tbody>
</table>
Return Value

Returns zero if successful, error code if unsuccessful.

Example

```c
ESS_FUNC_M ESS_PartitionPurgeOtlChangeFile(ESS_HCTX_T hCtx)
{
    ESS_FUNC_M sts;
    ESS_STR_T hostname, appname, dbname;
    ESS_USHORT_T usType, usdir;
    ESS_PART_DEFINED_T Partition;
    memset(&Partition, 0, sizeof(ESS_PART_DEFINED_T));

    hostname = "local";
    appname = "App1";
    dbname = "Src1";
    usType = ESS_PARTITION_OP_LINKED;
    usdir = ESS_PARTITION_DATA_TARGET;
    Partition.usType = usType;
    Partition.usDirection = usdir;
    Partition.HostDatabase.pszHostName = hostname;
    Partition.HostDatabase.pszAppName = appname;
    Partition.HostDatabase.pszDbName = dbname;
    sts = EssPartitionPurgeOtlChangeFile (hCtx, &Partition, 0);
    printf("EssPartitionPurgeOtlChangeFile   sts:  %ld\n",sts);
    return(sts);
}
```

See Also

- “Constant and Structure Definitions for Partitions (C)” on page 109
- EssPartitionApplyOtlChangeFile
- EssPartitionApplyOtlChangeRecs
- EssPartitionCloseDefFile
- EssPartitionFreeDefCtx
- EssPartitionFreeOtlChanges
- EssPartitionGetAreaCellCount
- EssPartitionGetList
- EssPartitionGetOtlChanges
- EssPartitionGetReplCells
- EssPartitionNewDefFile
- EssPartitionOpenDefFile
- EssPartitionPutReplCells
- EssPartitionReadDefFile
- EssPartitionReadOtlChangeFile
- EssPartitionReplaceDefFile
- EssPartitionResetOtlChangeTime
- EssPartitionValidateDefinition
- EssPartitionWriteDefFile
EssPartitionPutReplCells

Replicates all data cells that are identified in the replication partition from the selected source database to the target database.

Syntax

ESS_FUNC_M EssPartitionPutReplCells (hCtx, pReplicatePartition);

Parameter | Data Type          | Description
----------|--------------------|----------------
hCtx       | ESS_HCTX_T         | API context handle.
pReplicatePartition | ESS_PPART_REPL_T  | Partition information.

Notes

This routine removes the file if it's empty after purging.

Return Value

Returns zero if successful; error code if unsuccessful.

Access

A call to this function requires database designer access privileges.

Example

ESS_FUNC_M  Ess_PartitionPutReplCells(ESS_HCTX_T hCtx)
{
    ESS_FUNC_M  sts;
    ESS_PART_REPL_T ReplicatePartition;
    ESS_PART_CONNECT_INFO_T  HostDatabase;

    memset(&ReplicatePartition, 0, sizeof(ESS_PART_REPL_T));
    memset(&HostDatabase, 0, sizeof(ESS_PART_CONNECT_INFO_T));

    ReplicatePartition.pHostDatabase = &HostDatabase;
    ReplicatePartition.lPartitionCount = -1;
    ReplicatePartition.bUpdatedOnly = ESS_FALSE;

    sts = EssPartitionPutReplCells(hCtx, &ReplicatePartition);
    printf("EssPartitionPutReplCells  sts: %ld\n",sts);

    return(sts);
}

See Also

- “Constant and Structure Definitions for Partitions (C)” on page 109
- EssPartitionApplyOtlChangeFile
- EssPartitionApplyOtlChangeRecs
- EssPartitionCloseDefFile
- EssPartitionFreeDefCtx
- EssPartitionFreeOtlChanges
**EssPartitionReadDefFile**

Reads a partition definition file into memory.

**Syntax**

```c
ESS_FUNC_M EssPartitionReadDefFile (hCtx, iFileHandle, pDdbCtx);
```

**Parameter** | **Data Type** | **Description**
--- | --- | ---
`hCtx` | ESS_HCTX_T | API context handle.
`iFileHandle` | ESS_INT_T | Handle to partition definitions file.
`pDdbCtx` | “ESS_PART_T” on page 152 | Distributed database context to be filled.

**Notes**

- Use this function as part of a sequence of definition operations. Use `EssPartitionOpenDefFile()` to open existing definition files. Use `EssPartitionNewDefFile()` to create and open a new definition file. Use `EssPartitionReadDefFile()` or `EssPartitionWriteDefFile()` to read or write a definition file. Close with `EssPartitionCloseDefFile()`. Then free the memory with `EssPartitionFreeDefCtx()`.

**Return Value**

Returns zero if successful; error code if unsuccessful.

**Example**

For an example, see `EssPartitionOpenDefFile`.

**See Also**

- “Constant and Structure Definitions for Partitions (C)” on page 109
- `EssPartitionApplyOtlChangeFile`
- `EssPartitionApplyOtlChangeRecs`
EssPartitionReadOtlChangeFile

Reads changes from a change file (*.CHG) on the target database into memory. This function is designed to be used interactively with EssPartitionApplyOtlChangeRecs() after a call to EssPartitionGetOtlChanges(). This function can be used with filters.

Syntax

ESS_FUNC_M EssPartitionReadOtlChangeFile (hCtx, pSelectMetaRecords, pMetaChangeRead);

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>pSelectMetaRecords</td>
<td>“ESS_PARTOTL_SELECT_CHG_T” on page 171</td>
<td>Criteria to select records to read.</td>
</tr>
<tr>
<td>pMetaChangeRead</td>
<td>ESS_PREAD_T</td>
<td>Pointer to meta change records read from the file.</td>
</tr>
</tbody>
</table>

Notes

This routine returns a time in pMetaChangeRead. It’s the same time stamp you should pass to EssXXApplyRecords() to update the timestamp at target database. It’s also the same time stamp you should use for EssStampPurge() to purge applied records.

Return Value

Returns zero if successful; error code if unsuccessful.

Access

A call to this function requires database designer access privileges.
Example

```c
ESS_FUNC_M ESS_PartitionReadOtlChangeFile(ESS_HCTX_T hCtx, ESS_HINST_T hInst)
{
    ESS_FUNC_M stst;
    ESS_STR_T chgfilename;
    ESS_TIME_T time;
    ESS_PARTOTL_CHANGE_API_T OtlChg;
    ESSULONG_T uldimfilter, ulmbrfilter, ulmbrattrfilter = 0;
    ESS_PARTOTL_SELECT_CHG_T SelectMetaRecords;
    ESS_PARTOTL_READ_T MetaChangeRead;

    memset(&OtlChg, 0, sizeof(ESS_PARTOTL_CHANGE_API_T));
    memset(&SelectMetaRecords, 0, sizeof(ESS_PARTOTL_SELECT_CHG_T));
    memset(&MetaChangeRead, 0, sizeof(ESS_PARTOTL_READ_T));

    chgfilename = "d:\essbase5\app\app1\trg1\ess00001.chg";
    time = 0;

    uldimfilter = ESS_DIMCHG_ALL;
    ulmbrfilter = ESS_PARTITION_OTLMBR_ALL;
    ulmbrattrfilter = ESS_PARTITION_OTLMBRATTR_ALL;
    SelectMetaRecords.pszFileName = chgfilename;
    SelectMetaRecords.QueryFilter.TimeStamp = time;
    SelectMetaRecords.QueryFilter.ulDimFilter = uldimfilter;
    SelectMetaRecords.QueryFilter.ulMbrFilter = ulmbrfilter;
    SelectMetaRecords.QueryFilter.ulMbrAttrFilter = ulmbrattrfilter;

    MetaChangeRead.pOtlChg = &OtlChg;
    stst = ESS_PartitionReadOtlChangeFile (hCtx, &SelectMetaRecords, &MetaChangeRead);
    printf("\t\n\tEssPartitionReadOtlChangeFile stst: %ld\n", stst);
    stst = ESS_PartitionFreeOtlChanges(hCtx);
    printf("\t\n\tEssPartitionFreeOtlChanges stst: %ld\n", stst);

    return(stst);
}
```

See Also

- “Constant and Structure Definitions for Partitions (C)” on page 109
- ESS_PartitionApplyOtlChangeFile
- ESS_PartitionApplyOtlChangeRecs
- ESS_PartitionCloseDefFile
- ESS_PartitionFreeDefCtx
- ESS_PartitionFreeOtlChanges
- ESS_PartitionGetAreaCellCount
- ESS_PartitionGetList
- ESS_PartitionGetOtlChanges
- ESS_PartitionGetRep1Cells
- ESS_PartitionNewDefFile
- ESS_PartitionOpenDefFile
- ESS_PartitionPurgeOtlChangeFile
- ESS_PartitionPutRep1Cells
- ESS_PartitionReadDefFile
- ESS_PartitionReplaceDefFile
EssPartitionReplaceDefFile

Tells the server that a new shared-partition file has been sent, which replaces any existing file for this database.

Syntax

```
ESS_FUNC_M EssPartitionReplaceDefFile (hCtx);
```

Parameter | Data Type | Description
--- | --- | ---
hCtx | ESS_HCTX_T | API context handle.

Return Value

Returns zero if successful; error code if unsuccessful

Access

A call to this function requires database designer access privileges.

Example

```
ESS_FUNC_M Ess_PartitionReplaceDefFile(ESS_HCTX_T hCtx)
{
    ESS_FUNC_M sts = ESS_STS_NOERR;

    sts = EssPartitionReplaceDefFile(hCtx);
    printf("EssPartitionReplaceDefFile   sts: %ld\n",sts);
    return(sts);
}
```

See Also

- “Constant and Structure Definitions for Partitions (C)” on page 109
- EssPartitionApplyOtlChangeFile
- EssPartitionApplyOtlChangeRecs
- EssPartitionCloseDefFile
- EssPartitionFreeDefCtx
- EssPartitionFreeOtlChanges
- EssPartitionGetAreaCellCount
- EssPartitionGetList
- EssPartitionGetOtlChanges
- EssPartitionGetReplCells
- EssPartitionNewDefFile
- EssPartitionOpenDefFile
- EssPartitionPurgeOtlChangeFile
- EssPartitionPutReplCells
- EssPartitionReadDefFile
EssPartitionResetOTLChangeTime

Takes the "last change" time from the source partition and assigns it as a "last meta change" time of a destination partition.

Syntax

```
ESS_FUNC_M EssPartitionResetOTLChangeTime
    (hCtx, pSourcePartition, pDestinationPartition);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>pSourcePartition</td>
<td>“ESS_PART_DEFINED_T” on page 153</td>
<td>Partition for the new time.</td>
</tr>
<tr>
<td>pDestinationPartition</td>
<td>“ESS_PART_DEFINED_T” on page 153</td>
<td>Partition where the time is reset.</td>
</tr>
</tbody>
</table>

Notes

- The source partition refers to a partition that provides a time stamps and target partition refers to a partition which receives the time stamp.
- A source partition does not have to be either a data source partition or an outline source partition.

Return Value

Returns zero if successful; error code if unsuccessful

Access

A call to this function requires Database Manager permission.

Example

```
ESS_FUNC_M ESS_PartitionResetOTLChangeTime(ESS_HCTX_T  hCtx) {
    ESS_FUNC_M         sts;
    ESS_PART_DEFINED_T SourcePartition, TargetPartition;
    memset(&SourcePartition, 0, sizeof(ESS_PART_DEFINED_T));
    memset(&TargetPartition, 0, sizeof(ESS_PART_DEFINED_T));

    SourcePartition.HostDatabase.pszHostName = "local";
    SourcePartition.HostDatabase.pszAppName = "App1";
    SourcePartition.HostDatabase.pszDbName = "Src1";
    SourcePartition.usType = ESS_PARTITION_OP_LINKED;
    SourcePartition.usDirection = ESS_PARTITION_DATA_SOURCE;

    TargetPartition.HostDatabase.pszHostName = "local";
}
```
TargetPartition.HostDatabase pszAppName = "App1";
TargetPartition.HostDatabase pszDbName = "Trg1";
TargetPartition.usType = ESS_PARTITION_OP_LINKED;
TargetPartition.usDirection = ESS_PARTITION_DATA_TARGET;

sts = EssPartitionResetOtlChangeTime (hCtx, &SourcePartition, &TargetPartition);
printf("EssPartitionResetOtlChangeTime   sts: %ld\n", sts);
return(sts);
}

See Also

- “Constant and Structure Definitions for Partitions (C)” on page 109
- EssPartitionApplyOtlChangeFile
- EssPartitionApplyOtlChangeRecs
- EssPartitionCloseDefFile
- EssPartitionFreeDefCtx
- EssPartitionFreeOtlChanges
- EssPartitionGetAreaCellCount
- EssPartitionGetList
- EssPartitionGetOtlChanges
- EssPartitionGetReplCells
- EssPartitionNewDefFile
- EssPartitionOpenDefFile
- EssPartitionPurgeOtlChangeFile
- EssPartitionPutReplCells
- EssPartitionReadDefFile
- EssPartitionReplaceDefFile
- EssPartitionResetOtlChangeTime
- EssPartitionValidateDefinition
- EssPartitionWriteDefFile

**EssPartitionValidateDefinition**

Verifies the local partition definition (specified by ESS_PPARTSLCT_VALIDATE_T) against the corresponding partition definition in pRemoteDDBfilename on the remote server.

**Syntax**

```
ESS_FUNC_M EssPartitionValidateDefinition (hCtx, pSelectVerify,
                                          pulInvalidComponent, ppInvalidComponent, pRemoteDDBFileName);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>pSelectVerify</td>
<td>“ESS_PPARTSLCT_VALIDATE_T” on page 172</td>
<td>Description of the partition to verify.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>pulInvalidComponent</td>
<td>ESS_PULONG_T</td>
<td>Number of errors and warnings resulting from validation.</td>
</tr>
<tr>
<td>ppInvalidComponent</td>
<td>“ESS_PARTDEF_INVALID_T” on page 155</td>
<td>List of errors and warnings resulting from validation.</td>
</tr>
<tr>
<td>pRemoteDDBFileName</td>
<td>ESS_STR_T</td>
<td>Remote server partition definition file name.</td>
</tr>
</tbody>
</table>

**Notes**

- Call the function `EssFree()` to free the invalid component when `pulInvalidComponent` is not 0.
- The remote partition definition file can reside locally or on the remote host. If the partition definition file is local, `pRemoteDDBFileName` must specify the full path, including the file name with extension. If the partition definition file is remote, `pRemoteDDBFileName` must specify the file name without extension (the extension is assumed to be .DDB).
- The server uses the following rule to find the partition definition file on the system:
  - If `pSelectVerify->pszFileName = DbName`, the server looks for `DbName.DDN`.
  - If `pSelectVerify->pszFileName != DbName`, the server looks for `pSelectVerify->pszFileName.DDB`.

**Return Value**

Returns zero if successful; error code if unsuccessful.

**Access**

A call to this function requires database designer access privileges.

**Example**

```c
ESS_STS_T  ESS_PartitionValidateDefinition(ESS_HCTX_T hCtx, ESS_HINST_T hInst) {
    ESS_STS_T  sts = 0;
    ESS_PARTSLCT_VALIDATE_T  SelectVerify;
    ESS_PARTDEF_INVALID_T  *pInvalidComponent;
    ESS_ULONG_T  ulInvalidComponentCount = 0;
    ESS_STR_T  pRemoteDDBFileName = "src";

    /* assume, logged into target database */

    memset(&SelectVerify, 0, sizeof(ESS_PARTSLCT_VALIDATE_T));
    SelectVerify.usLoc             = ESS_FILE_SERVER;
    SelectVerify.pszFileName       = "trg";
    SelectVerify.Part.usType       = ESS_PARTITION_OP_REPLICATED;
    SelectVerify.Part.usDirection  = ESS_PARTITION_DATA_TARGET;
    SelectVerify.Part.HostDatabase.pszHostName = "Local"
    SelectVerify.Part.HostDatabase.pszAppName = "PartSrc";
    SelectVerify.Part.HostDatabase.pszDbName = "Src";

    sts = EssPartitionValidateDefinition (hCtx, &SelectVerify,
                                         &ulInvalidComponentCount, &pInvalidComponent, pRemoteDDBFileName);
```

C Main API Function Reference  561
if (ulInvalidComponentCount > 0)
    printf("Validation resulted in warnings and errors.\n");
else
    printf ("Partition is valid.\n");

if (pInvalidComponent)
    EssFree(hInst, pInvalidComponent);

return(sts);
}

See Also

- “Constant and Structure Definitions for Partitions (C)” on page 109
- EssPartitionApplyOtlChangeFile
- EssPartitionApplyOtlChangeRecs
- EssPartitionCloseDefFile
- EssPartitionFreeDefCtx
- EssPartitionFreeOtlChanges
- EssPartitionGetAreaCellCount
- EssPartitionGetList
- EssPartitionGetOtlChanges
- EssPartitionGetReplCells
- EssPartitionNewDefFile
- EssPartitionOpenDefFile
- EssPartitionPurgeOtlChangeFile
- EssPartitionPutReplCells
- EssPartitionReadDefFile
- EssPartitionReadOtlChangeFile
- EssPartitionReplaceDefFile
- EssPartitionResetOtlChangeTime
- EssPartitionValidateLocal
- EssPartitionWriteDefFile

**EssPartitionValidateLocal**

Verifies all partition definitions associated with the database specified by ESS_HCTX_T.

**Syntax**

```c
ESS_FUNC_M EssPartitionValidateLocal (hCtx, pusValidateResult);
```

**Parameter**

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
</tr>
<tr>
<td>pusValidateResult</td>
<td>ESS_PUSHORT_T</td>
</tr>
</tbody>
</table>

Result of partition validation.

**Notes**

pusValidateResult can be one of these values:

- ESS_DDB_VERIFY_ERROR (validation resulted in errors)
- ESS_DDB_VERIFY_FAIL (validation failed)
- ESS_DDB_VERIFY_NOERR (all partitions are valid)
- ESS_DDB_VERIFY_WARNING (validation resulted in warnings)

**Return Value**

Returns zero if the function completes successfully; error code if the function completes unsuccessfully. Returns zero if the function operates on a database with no partition definition.

**Access**

A call to this function requires database designer access privileges.

**Example**

```c
ESS_FUNC_RESULT ESS_PartitionValidateLocal(ESS_HCTX_T hCtx) {
    ESS_FUNC_RESULT sts = ESS_STS_NOERR;
    ESS_USHORT_T usValidateRes = (ESS_USHORT_T)ESS_DDB_VERIFY_NOERR;

    sts = EssPartitionValidateLocal(hCtx, &usValidateRes);

    if (!sts) {
        switch (usValidateRes) {
            case ESS_DDB_VERIFY_WARNING:
                printf("Validation resulted in warning(s) - see server log for details\n");
                break;
            case ESS_DDB_VERIFY_ERROR:
                printf("Validation resulted in error(s) - see server log for details\n");
                break;
            default:
                printf("\nPartition(s) validated\n");
                break;
        }
    } else {
        printf("Call to EssPartitionValidateLocal() failed.\n");
    }
    return (sts);
}
```

**See Also**

- “Constant and Structure Definitions for Partitions (C)” on page 109
- EssPartitionApplyOtlChangeFile
- EssPartitionApplyOtlChangeRecs
- EssPartitionCloseDefFile
- EssPartitionFreeDefCtx
- EssPartitionFreeOtlChanges
- EssPartitionGetAreaCellCount
- EssPartitionGetList
- EssPartitionGetOtlChanges
EssPartitionWriteDefFile

Writes the current memory version of the shared-partition definition file to disk.

Syntax

\texttt{ESS\_FUNC\_M \textbf{EssPartitionWriteDefFile} (hCtx, iFileHandle, TpDdbCtx);}

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>Essbase API context handle.</td>
</tr>
<tr>
<td>iFileHandle</td>
<td>ESS_INT_T</td>
<td>Handle to shared partition definition file.</td>
</tr>
<tr>
<td>pDdbCtx</td>
<td>“ESS_PART_T” on page 152 Values to be written out.</td>
<td></td>
</tr>
</tbody>
</table>

Notes

- Use this function as part of a sequence of definition operations. Use \texttt{EssPartitionOpenDefFile()} to open existing definition files. Use \texttt{EssPartitionNewDefFile()} to create and open a new definition file. Use \texttt{EssPartitionReadDefFile()} or \texttt{EssPartitionWriteDefFile()} to read or write a definition file. Close with \texttt{EssPartitionCloseDefFile()}. Then free the memory with \texttt{EssPartitionFreeDefCtx()}.

Return Value

Returns zero if successful, error code if unsuccessful.

Example

For an example, see \texttt{EssPartitionNewDefFile}.

See Also

- “Constant and Structure Definitions for Partitions (C)” on page 109
- \texttt{EssPartitionApplyOtlChangeFile}
- \texttt{EssPartitionApplyOtlChangeRecs}
- \texttt{EssPartitionCloseDefFile}
- \texttt{EssPartitionFreeDefCtx}
EssPerformAllocationASO

Performs or verifies an allocation on an aggregate storage database.

**Syntax**

```
ESS_FUNC_M EssPerformAllocationASO (hCtx, verifyOnly, errorList, allocStruct);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>verifyOnly</td>
<td>ESS_BOOL_T</td>
<td>Flag to indicate validation of allocation parameters without performing allocation. If is set to ESS_TRUE, the allocation parameters are validated only. If it is ESS_FALSE, the allocation is verified and executed.</td>
</tr>
<tr>
<td>errorList</td>
<td>&quot;ESS_PERF_ALLOC_ERROR_T&quot; on page 173**</td>
<td>A pointer to the linked list of error structures that will be allocated and returned by the API function. This is so the client has more information about warning and error messages. This argument cannot be 0. The linked list must be freed by the client.</td>
</tr>
<tr>
<td>allocStruct</td>
<td>&quot;ESS_PERF_ALLOC_T&quot; on page 174*</td>
<td>Structure specifying the allocation parameters.</td>
</tr>
</tbody>
</table>

**Return Value**

Returns 0 if successful; otherwise, returns an error.

**Example**

```c
void HandleErrors(ESS_HINST_T hInst, ESS_PERF_ALLOC_ERROR_T **pErrorList)
{
    if (pErrorList)
    {
        ESS_PERF_ALLOC_ERROR_T *errorList = *pErrorList;
        ESS_PERF_ALLOC_ERROR_T *nextError;

        while (errorList)
```

---

*Note: The ** syntax indicates linked list pointers.*

**EssPartitionNewDefFile**

Creates a new definition file.

**Syntax**

```
void EssPartitionNewDefFile(ESS_HINST_T hInst, ESS_HCTX_T hCtx, const char* filename);
```

**Parameter**

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInst</td>
<td>API instance handle.</td>
</tr>
<tr>
<td>hCtx</td>
<td>API context handle.</td>
</tr>
<tr>
<td>filename</td>
<td>Path to the new definition file.</td>
</tr>
</tbody>
</table>

**EssPartitionResetOtlChangeTime**

Resets the Otl Change Time.

**Syntax**

```
void EssPartitionResetOtlChangeTime(ESS_HINST_T hInst, ESS_HCTX_T hCtx);
```

**EssPartitionValidateDefinition**

Validates the definition.

**Syntax**

```
void EssPartitionValidateDefinition(ESS_HINST_T hInst, ESS_HCTX_T hCtx, const char* filename);
```

**Parameter**

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInst</td>
<td>API instance handle.</td>
</tr>
<tr>
<td>hCtx</td>
<td>API context handle.</td>
</tr>
<tr>
<td>filename</td>
<td>Path to the definition file.</td>
</tr>
</tbody>
</table>

---

*Note: The ** syntax indicates linked list pointers.*
void ESS_GLAllocation()
{
    ESS_STS_T sts = ESS_STS_NOERR;
    ESS_BOOL_T verifyOnly;
    ESS_PERF_ALLOC_ERROR_T *errorList = ESS_NULL;
    ESS_PERF_ALLOC_T *allocStruct;
    sts = EssAlloc (hInst, sizeof(ESS_PERF_ALLOC_T), (ESS_PPVOID_T)&allocStruct);
    printf("EssAlloc sts for allocStruct: %ld\n", sts);
    verifyOnly = ESS_FALSE;
    errorList = ESS_NULL;
    allocStruct->pov = "[[Account]]@[1100]].Children";
    allocStruct->amount = "100";
    allocStruct->amountContext = "";
    allocStruct->amountTimeSpan = "";
    allocStruct->target = "((Allocated), [041509GR PL2], [11], [[All Department Values]].
    [000]), [0000], [Base], [USD], [Total])";
    allocStruct->targetTimeSpan = "[[Feb-08]]";
    allocStruct->targetTimeSpanOption = ESS_ASO_ALLOCATION_TIMESPAN_DIVIDEDMT;
    allocStruct->offset = "((Mar-08), [041509GR PL2], [11], [[All Department Values]].
    [000]), [0000], [Base], [USD], [Total], [291], [Allocated])";
    allocStruct->debitMember = "[Beginning Balance Dr]";
    allocStruct->creditMember = "[Beginning Balance Cr]";
    allocStruct->range = "DESCENDANTS([Accessories], [Product].Levels(0))";
    allocStruct->excludedRange = "";
    allocStruct->basis = "([041509GR PL2], [11], [[All Department Values]].[000]),
    [0000], [Base], [USD], [Total], [Beginning Balance Cr], [4140], [Actual])";
    allocStruct->basisTimeSpan = "[[Feb-08]]";
    allocStruct->basisTimeSpanOption = ESS_ASO_ALLOCATION_TIMESPAN_COMBINEBASIS;
    allocStruct->allocationMethod = ESS_ASO_ALLOCATION_METHOD_SHARE;
    allocStruct->spreadSkipOption = 0;
    allocStruct->zeroAmountOption = ESS_ASO_ALLOCATION_ZEROAMT_DEFAULT;
    allocStruct->zeroBasisOption = ESS_ASO_ALLOCATION_ZEROBASIS_NEXTAMT;
    allocStruct->negativeBasisOption = ESS_ASO_ALLOCATION_NEGBASIS_DEFAULT;
    allocStruct->roundMethod = ESS_ASO_ALLOCATION_ROUND_NONE;
    allocStruct->roundDigits = "";
    allocStruct->roundToLocation = "";
}
allocStruct->groupID = 0;
allocStruct->ruleID = 0;

sts = EssPerformAllocationAso(hCtx, verifyOnly, &errorList, allocStruct);
printf("EssPerformAllocationAso sts: %ld\n",sts);

HandleErrors(hInst, &errorList);
if(allocStruct)
{
    sts = EssFree (hInst, allocStruct);
    printf("EssFree sts for allocStruct %ld\n",sts);
}

---

**EssPerformCustomCalcASO**

Performs or verifies a custom calculation on an aggregate storage database.

**Syntax**

ESS_FUNC_M EssPerformCustomCalcASO (hCtx, verifyOnly, errorList, calcStruct);

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>verifyOnly</td>
<td>ESS_BOOL_T</td>
<td>Flag to indicate whether the calculation will be validated without executing it. If it is set to ESS_TRUE, the calculation is validated only. If it is ESS_FALSE, the calculation is validated and executed.</td>
</tr>
<tr>
<td>errorList</td>
<td>“ESS_PERF_ALLOC_ERROR_T” on page 173**</td>
<td>A pointer to the linked list of error structures that will be populated and returned by the API containing error information about the custom calculation. This argument cannot be 0. The linked list must be freed by the client.</td>
</tr>
<tr>
<td>calcStruct</td>
<td>“ESS_PERF_CUSTCALC_T” on page 177*</td>
<td>Pointer to a client-allocated custom calculation structure and parameters.</td>
</tr>
</tbody>
</table>

**Return Value**

Returns 0 if successful; otherwise, returns an error.

**Example**

```c
void HandleErrors(ESS_HINST_T hInst, ESS_PERF_ALLOC_ERROR_T **pErrorList)
{
    if (pErrorList)
    {
        ESS_PERF_ALLOC_ERROR_T *errorList = *pErrorList;
        ESS_PERF_ALLOC_ERROR_T *nextError;

        while (errorList)
        {
            printf("Error number %ld occurred\n", errorList->messageNumber);
            if (errorList->argument != ESS_PERF_ALLOC_ARG_NA)
                printf(" in argument %d\n", errorList->argument);
            if (errorList->lineNumber)
```
printf(" on line %ld\n", errorList->lineNumber);
if (errorList->token[0] != '\0')
    printf(" on token %s\n", errorList->token);

nextError = errorList->nextError;
ESS_STS_T sts = EssFree (hInst, errorList);
printf("\nEssFree sts for errorList %ld\n",sts);
errorList = nextError;
}

*pErrorList = NULL;
}
}

void ESS_GLCustomCalc()
{
    ESS_STS_T sts = ESS_STS_NOERR;
    ESS_BOOL_T verifyOnly;
    ESS_PERF_ALLOC_ERROR_T *errorList = ESS_NULL;
    ESS_PERF_CUSTCALC_T *calcStruct;

    sts = EssAlloc (hInst, sizeof(ESS_PERF_CUSTCALC_T), (ESS_PPVOID_T)&calcStruct);
    printf("EssAlloc sts for calcStruct: %ld\n", sts);

    sts = EssAlloc (hInst, sizeof(ESS_PERF_CUSTCALC_T), (ESS_PPVOID_T)&calcStruct);
    printf("EssAlloc sts: %ld\n", sts);

    verifyOnly = ESS_FALSE;
    errorList = ESS_NULL;
    calcStruct->pov = "{[1120], [1130]}";
    calcStruct->script = "[Jan-96] := ([Feb-08], [041509GR PL2], [00], [[All Department Values]].[000]], [0000], [[All Product Values]].[000]], [Actual], [Beginning Balance Dr], [BASE], [USD], [Total]);";
    calcStruct->target = "([041509GR PL2], [00], [[All Department Values]].[000]], [0000], [[All Product Values]].[000]], [Actual], [Beginning Balance Dr], [BASE], [USD], [Total]);";
    calcStruct->debitMember = "[Beginning Balance Dr]";
    calcStruct->creditMember = "[Beginning Balance Cr]";
    calcStruct->offset = "";
    calcStruct->sourceRegion = "{([Feb-08], [041509GR PL2], [00], [[All Department Values]].[000]], [0000], [[All Product Values]].[000]], [Actual], [Beginning Balance Dr], [BASE], [USD], [Total])";
    calcStruct->groupId = 0;
    calcStruct->ruleID = 0;

    sts = EssPerformCustomCalcAso(hCtx, verifyOnly, &errorList, calcStruct);
    printf("EssPerformCustomCalcAso sts: %ld\n",sts);

    HandleErrors(hInst, &errorList);

    if(calcStruct)
    {
        sts = EssFree (hInst, calcStruct);
        printf("EssFree sts for allocStruct %ld\n",sts);
    }
}
EssPutObject

Copies an object from a local file to the server or client object system, and optionally unlocks it.

**Syntax**

```c
ESS_FUNC_M EssPutObject (hCtx, ObjType, AppName, DbName, ObjName, LocalName, Unlock);
```

**Parameter** | **Data Type** | **Description**
--- | --- | ---
 hCtx | ESS_HCTX_T | API context handle. Can be local context handle returned by EssCreateLocalContext().
 ObjType | ESS_OBJTYPE_T | Object type (must be single type). Refer to “Bitmask Data Types (C)” on page 96 for a list of possible values.
 AppName | ESS_STR_T | Application name.
 DbName | ESS_STR_T | Database name. If NULL, uses the application subdirectory.
 ObjName | ESS_STR_T | Name of object to put.
 LocalName | ESS_STR_T | Full path name of local source file on client.
 Unlock | ESS_BOOL_T | Flag to control object unlocking. If TRUE, the server object is unlocked to allow updates by other users.

**Notes**

- In order to put an object which already exists on the server, it must have previously been locked by the caller. If the object does not already exist on the server, it will be created.

**Return Value**

If successful, the object is copied to the server from the local file specified by *LocalName*.

**Access**

This function requires the caller to have the appropriate level of access to the specified application and/or database to contain the object (depending on the object type). To unlock the object (unlock flag is TRUE), the caller must have Application or Database Design privilege (ESS_PRIV_APPDESIGN or ESS_PRIV_DBDESIGN) for the specified application or database containing the object.

**Example**

```c
ESS_FUNC_M
EssPutObject (ESS_HCTX_T hCtx)
{
    ESS_FUNC_M
    sts = ESS_STS_NOERR;
    ESS_STR_T
    AppName;
    ESS_STR_T
    DbName;
    ESS_STR_T
    ObjName;
    ESS_OBJTYPE_T
    ObjType;
    ESS_STR_T
    LocalName;
    ESS_BOOL_T
    UnLock;

    AppName    = "Sample";
}
```
DbName    = "Basic";
ObjName   = "Basic1";
ObjType   = ESS_OBJTYPE_OUTLINE;
LocalName = "C:\Hyperion\products\Essbase\EssbaseClient\Test.otl";
UnLock    = ESS_TRUE;

sts = EssPutObject (hCtx, ObjType, AppName,
                      DbName, ObjName, LocalName, UnLock);
return (sts);
}

See Also

● EssGetObject
● EssLockObject
● EssUnlockObject

EssQueryDatabaseMembers

Performs a report-style query to list a selection of database member information.

Syntax

ESS_FUNC_M EssQueryDatabaseMembers (hCtx, mbrQuery);

Parameter | Data Type | Description
-----------|-----------|------------------
hCtx       | ESS_HCTX_T | API context handle.
mbrQuery   | ESS_STR_T  | Member query string. A query string is a command similar to a report specification. For valid query strings see the Notes topic.

Notes

● The member information returned by this query must be read by calling EssGetString() until a NULL string is returned.
● This function supports an attribute member long name.
● See the Oracle Essbase Technical Reference for descriptions of report specifications.
● This function can return information on member stored as a relational partition if the Boolean bSpanRelPart has been set by EssSetSpanRelationalPartition. This function supports sorting of members based on member names, aliases (which are the same as member names for relational members) and dimension/generation numbers. For other options, the relational members are treated identically and displayed at the end of the list of members.

Not all member selection strings are supported in the relational store. This function can return relational information on the following member selection strings:

○ ALLINSAMEDIM
○ DIMTOP
○ CHILDRENOF
● The Member query string consists of a selection string and an optional sorting command followed by an optional output command. The form is:

\[ \text{mbQuery} ==: \text{<selectionstring>} \ [\text{<sortcommand> \ [\text{<outputcommand>}]}] \]

● The valid values for member <selectionstring> are:

- <CHILDRENOF -- returns ICHILDRENOF
- <ALLINSAMEDIM
- <DIMTOP
- <OFFSAMEGENERATION
- <ONSAMELEVELAS
- <ANCESTORSOFT -- returns IANCESTORSOFT
- <PARENTOF
- <DESCENDANTSOFT -- returns IDESCENDANTSOFT
- <ALLSIBLINGSOF
- <LSIBLINGOF

● Valid values for <sortcommand> are:

- <SORTASCENDING
- <SORTDESCENDING
- <SORTNONE
- <SORTMBRNUMBERS
- <SORTALTNAMES
- <SORTLEVELNUMBERS
- <SORTDIMNUMBERS
- <SORTGENERATION

● The form for <outputcommand> is:

\[ \text{<outputcommand>} ==: \text{Item \ [separator]} | \text{FORMAT \ {<item> \ <separator>}} \]

○ To obtain a one-item list of information on a member, use the following output commands:

\[ \text{<outputcommand>} ==: \text{<MBRNUMBERS} | \text{<ALTNAMES} | \text{<MBRNUMBERS} | \text{<DIMNUMBERS} | \text{<LEVELNUMBERS} | \text{<GENERATIONS} | \text{<CALCSTRINGS} | \text{<UCALCS} | \text{<TABSEPARATED} | \text{<SPACESEPARATED} | \text{<COMMASEPARATED} | \text{<NEWLINESEPARATED} | \text{<ATTRIBUTES}} \]

○ To obtain a list of two or more items of information on a member, use a format specification clause. Specify the items you want listed, their order, and what character to use to separate them. The syntax for a format specification clause is:
The valid values for `<item>` are:

- MBRNAMES
- ALTNAMES
- MBRNUMBERS
- DIMNUMBERS
- LEVELNUMBERS
- GENERATIONS
- CALCSTRINGS
- UCALCS
- ATTRIBUTES

ATTRIBUTES are listed as the number of attributes followed by a tab-separated list of attribute names.

The valid values for `<separator>` are:

- TABSEPARATED
- SPACESEPARATED
- COMMASEPARATED
- NEWLINESEPARATED

If you do not specify a separator, the default is TABSEPARATED.

Here is a sample script:

```c
login "local" "user1" "password" " " "
select "attr" "attr"
GetMembers "<NEWLINESEPARATED
<FORMAT {
  MBRNAMES SPACESEPARATED ALTNAMES TABSEPARATED
  MBRNUMBERS SPACESEPARATED DIMNUMBERS TABSEPARATED
  LEVELNUMBERS SPACESEPARATED GENERATIONS TABSEPARATED
  CALCSTRINGS SPACESEPARATED UCALCS TABSEPARATED
  DIMTYPES SPACESEPARATED STATUSES TABSEPARATED
  ATTRIBUTES
}
<DESCENDANTS Product 
```

Return Value

None.

Access

This function requires the caller to have at least read access (ESS_PRIV_READ) to the database, and to have selected it as their active database using EssSetActive().

Example

```c
ESS_STS_T
ESS_GetMembers (ESS_HCTX_T hCtx,
                 ESS_HINST_T hInst
              )
{
  ESS_STS_T     sts = ESS_STS_NOERR;
  ESS_STR_T     mString = NULL;
```
sts = EssQueryDatabaseMembers (hCtx, "<ALLINSAMEDIM Year");

if (!sts)
    sts = EssGetString (hCtx, &mString);

while (!sts && (mString != NULL))
{
    printf ("%s\r\n", mString);
    EssFree (hInst, mString);

    sts = EssGetString (hCtx, &mString);
}

return(sts);
}

See Also
- EssCheckMemberName
- EssGetMemberInfo
- EssSetActive

EssRealloc

Reallocates a previously-allocated block of memory to a different size, using the defined memory allocation scheme.

**Syntax**

ESS_FUNC_M EssRealloc (hInstance, Size, ppBlock);

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>ESS_HINST_T</td>
<td>API instance handle.</td>
</tr>
<tr>
<td>Size</td>
<td>ESS_SIZE_T</td>
<td>New size of memory block to reallocate.</td>
</tr>
<tr>
<td>ppBlock</td>
<td>ESS_PPVOID_T</td>
<td>Address of pointer to previously allocated memory block, to be updated to point to reallocated memory block.</td>
</tr>
</tbody>
</table>

**Notes**

- This function reallocates previously-allocated memory using the user-supplied memory management function passed to the EssInit() function. If no such functions are supplied, the default memory reallocation function (dependent on the platform) will be used.
- Only memory allocated with the EssAlloc() function should be reallocated using this call. Also, memory reallocated using this function should always be freed by using the EssFree() function.
- It is generally not advisable to reallocate a block of zero size, as the effects of such a reallocation are platform- and compiler-dependent.
Return Value
If successful, returns a pointer to the reallocated memory block in `ppBlock`.

Access
This function requires no special privileges.

Example
```c
ESS_VOID_T
ESS_Realloc (ESS_HINST_T hInst)
{
    ESS_FUNC_M     sts = ESS_STS_NOERR;
    ESS_SIZE_T    Size;
    ESS_PVOID_T   pBlock = NULL;

    /* Allocate memory */
    Size = 10;
    sts = EssAlloc(hInst, Size, &pBlock);
    if(sts)
        printf("Cannot allocate memory\r\n");

    /* Reallocate memory */
    Size = 20;
    if(!sts)
    {
        sts = EssRealloc(hInst, Size, &pBlock);
        if(sts)
            printf("Cannot reallocate memory\r\n");
    }

    if(pBlock)
        EssFree(hInst, pBlock);
}
```

See Also
- EssAlloc
- EssFree
- EssInit

**EssRemoveAlias**
Permanently removes an alias table from the active database.

Syntax
```c
ESS_FUNC_M    EssRemoveAlias (hCtx, AliasName);
```

Parameter  | Data Type      | Description
---------- | -------------- | ------------
hCtx       | ESS_HCTX_T     | API context handle.
AliasName   | ESS_STR_T      | Name of alias table to remove.
Notes

- This function can not remove the active alias table or the default alias table.
- Make sure that no one else is using the same database as the one you try to remove an alias table from by calling `EssListConnections()`.

Return Value

None.

Access

This function requires the caller to have at least read access (ESS_PRIV_READ) to the database, and to have selected it as their active database using `EssSetActive()`.

Example

```c
ESS_FUNC_M
ESS_RemoveAlias (ESS_HCTX_T hCtx)
{
    ESS_FUNC_M     sts = ESS_STS_NOERR;
    ESS_STR_T     AliasName;
    AliasName = "NewAlias";
    sts = EssRemoveAlias(hCtx, AliasName);
    if(!sts)
        printf("The %s is removed.\r\n",AliasName);

    return (sts);
}
```

See Also

- `EssClearAliases`
- `EssListAliases`
- `EssSetActive`
- `EssRemoveLocks`

**EssRemoveLocks**

Removes all data block locks on a database which are currently held by a user.

Syntax

```c
ESS_FUNC_M EssRemoveLocks (hCtx, AppName, DbName, LoginId);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>AppName</td>
<td>ESS_STR_T</td>
<td>Application name.</td>
</tr>
<tr>
<td>DbName</td>
<td>ESS_STR_T</td>
<td>Database name.</td>
</tr>
<tr>
<td>LoginId</td>
<td>ESS_LOGINID_T</td>
<td>Id of user login whose locks are to be removed.</td>
</tr>
</tbody>
</table>
Notes

- The required *LoginId* can be obtained from the user lock info structure returned by the `EssListLocks` function.

- `EssRemoveLocks()` terminates the connection of the user specified by *LoginId* if that user is currently logged in.

Return Value

None.

Access

This function requires the caller to have database Design privilege (ESS_PRIV_DBDESIGN) for the specified database.

Example

```c
ESS_FUNC_M
ESS_RemoveLocks (ESS_HCTX_T hCtx,
                 ESS_HINST_T hInst)
{
    ESS_FUNC_M sts = ESS_STS_NOERR;
    ESS_USHORT_T Count;
    ESS_PLOCKINFO_T plockinfo = NULL;
    ESS_PLOCKINFO_T plinfo;
    ESS_USHORT_T ind;
    ESS_SHORT_T Item;
    ESS_STR_TAppName;
    ESS_STR_TDbName;
    AppName = "Sample";
    DbName = "Basic";
    for (ind = 0; ind < Count; ind++)
    {
        plinfo = plockinfo + ind;
        printf ("%-2d %-15s %-12ld %-5d %ld\r\n",
            ind, plinfo->UserName, plinfo->LoginId,
            plinfo->nLocks, plinfo->Time);
    }
    printf ("\r\n");
    /***********************************
    * Chooser Lock List Item to Remove *
    ***********************************/
    Item = 1;
}
else
{
    printf ("\r\nExclusive Lock List on %s:%s is empty\r\n\r\n",
        AppName, DbName);
    goto exit;
}
if (!sts)
{
    if ((Item >= 0) && (Item < Count))
    {
```
plinfo = plockinfo + Item;
sts = EssRemoveLocks (hCtx, AppName,
    DbName, plinfo->LoginId);
}
exit:
if (plockinfo)
    EssFree (hInst, plockinfo);
return (sts);
}

See Also
● EssListLocks

**EssReplayTransactions**

Executes (replays) the specified transactions.

- By default, **EssReplayTransactions** replays everything since the last restored backup time or last replayed request time—whichever is the latest.
- It will not replay requests made after the restore, because the recommended way to use restore command is to replay transactions and then open up for new transactions.
- You can use the `pSeqIds` option to force replays.

**Syntax**

```c
ESS_FUNC_M EssReplayTransactions(hCtx, AppName,
                                    DbName, ReplayDat, pSeqIds);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>Login context</td>
</tr>
<tr>
<td>AppName</td>
<td>ESS_STR_T</td>
<td>Application Name</td>
</tr>
<tr>
<td>DbName</td>
<td>ESS_STR_T</td>
<td>Database Name</td>
</tr>
<tr>
<td>ReplayDat</td>
<td>ESS_TRANSACTION_REPLAY_INP_T</td>
<td>Replay input parameters</td>
</tr>
<tr>
<td>pSeqIds</td>
<td>ESS_PSEQID_T</td>
<td>Array of sequence ID ranges if input type is sequence ID</td>
</tr>
</tbody>
</table>

**Return Value**

- 0—If successful
  - `pSeqIds` contains a range of sequence IDs
- Error number—If unsuccessful

**Access**

The caller must have Essbase Administrator access to the database.
Example

void ListAndReplayTransactions()
{
    ESS_FUNC_M                sts = ESS_STS_NOERR;
    ESS_USHORT_T                TimeSrc;
    ESS_TIME32_T                timestamp = 0;
    ESS_USHORT_T                listOption;
    ESS_STR_T                    FileName = ESS_NULL;
    ESS_ULONG_T                Count = 0;
    ESS_PTRANSACTION_ENTRY_T    pResults;
    ESS_CHAR_T                listTime[ESS_TIMESIZE];
    ESS_TRANSACTION_REPLAY_INP_T    ReplayDat;
    ESS_PSEQID_T                pSeqIds = ESS_NULL;
    ESS_OBJDEF_T                Data;
    ESS_STR_T                    Script;
    ESS_SHORT_T                isAbortOnError;
    ESS_PMBRERR_T            pMbrErr = NULL;
    ESS_PROCSTATE_T        pState;

    /* Load data from server */
    Data.hCtx      = hCtx;
    Data.AppName   = AppName;
    Data.DbName    = DbName;
    Data.ObjType   = ESS_OBJTYPE_TEXT;
    Data.FileName  = "Calcdat";
    isAbortOnError = ESS_TRUE;
    sts = EssImport (hCtx, ESS_NULL, &Data,
    &pMbrErr, NULL, isAbortOnError);
    printf("EssImport sts: %ld\r\n",sts);

    /* List and replay with a specified time */
    TimeSrc = 1;
    strcpy(listTime, "09/18/2007:00:00:00");
    /* mm/dd/yyyy:hh:mm:ss */
    timestamp = adtGenericGetTime(listTime);
    listOption = ESS_LIST_TRANSACTIONS_TOCLIENT;
    sts = EssListTransactions(hCtx, TimeSrc,
    timestamp, listOption,
    FileName, &Count, &pResults);
    printf("EssListTransactions sts: %ld\r\n",sts);

    if (Count && pResults)
        PrintTransactionLog(Count, pResults);

    memset(&ReplayDat, 0, sizeof
    (ESS_TRANSACTION_REPLAY_INP_T));
    ReplayDat.InpType = ESS_REPLAY_BASED_GIVENTIME;
    ReplayDat.value.InpTime = timestamp;
    sts = EssReplayTransactions (hCtx, AppName, DbName,
    &ReplayDat, pSeqIds);
    printf("EssReplayTransactions sts: %ld\r\n",sts);

    /* Run a calc*/
    Script = "CALC ALL;";
    sts = EssCalc(hCtx, ESS_TRUE, Script);
printf("EssCalc sts: %ld\r\n",sts);
if (!sts)
{
    sts = EssGetProcessState (hCtx, &pState);
    while (!sts && (pState.State != ESS_STATE_DONE))
        sts = EssGetProcessState (hCtx, &pState);
}

/* List and replay with last replay time */
TimeSrc = 2;
timestamp = 0;
sts = EssListTransactions(hCtx, TimeSrc,
    /* This function converts listTime to the number of
    seconds since January 1, 1970. */
    timestamp, listOption,
    FileName, &Count, &pResults);

/* This function converts listTime to the number of
seconds since January 1, 1970. */
printf("EssListTransactions sts: %ld\r\n",sts);
if (Count && pResults)
    PrintTransactionLog(Count, pResults);
memset(&ReplayDat, 0, sizeof
    (ESS_TRANSACTION_REPLAY_INP_T));
ReplayDat.InpType = ESS_REPLAY_BASED_LASTREPLAYTIME;
sts = EssReplayTransactions (hCtx, AppName,
    DbName, ReplayDat, pSeqIds);

printf("EssReplayTransactions sts: %ld\r\n",sts);
if(pSeqIds)
    EssFree(hInst, pSeqIds);
if(pResults)
    EssFree(hInst, pResults);
if(pMbrErr)
    EssFree(hInst, pMbrErr);
}

Using SeqIds

When you replay using the sequence id array, specify a range of sequence ids.

- Enter the range count in num_seq_id_range.
- Follow num_seq_id_range with an array of ESS_SEQID_T, and the type data structure
  and the number of elements in the array should be num_seq_id_range.
- The seq_id_upper_start and seq_id_upper_end fields are reserved, and should be
  filled with zeros.
- The seq_id_start and seq_id_end fields should be filled in with start and end values of
  the range.
- If you only have one sequence id, specify that id as the start and end value.

Example 1: To replay the ranges 1-5, 8-10 and 12-16 while skipping 6,7, and 11:

num_seq_id_range = 3
seqid_array[0].seq_id_start = 1
seqid_array[0].seq_id_end = 5
seqid_array[0].seq_id_start_upper = 0
Example 2: To replay one range 3-7, num_seq_id_range = 1:

seqid_array[0].seq_id_start = 3
seqid_array[0].seq_id_end = 7
seqid_array[0].seq_id_start_upper = 0
seqid_array[0].seq_id_end_upper = 0

Example 3: To replay only transaction id 5:

num_seq_id_range = 1
seqid_array[0].seq_id_start = 5
seqid_array[0].seq_id_end = 5
seqid_array[0].seq_id_start_upper = 0
seqid_array[0].seq_id_end_upper = 0

See Also
- “ESS_SEQID_T” on page 186
- “ESS_DISKVOLUME_REPLACE_T” on page 137
- “ESS_TRANSACTION_ENTRY_T” on page 187
- “ESS_TRANSACTION_REPLAY_INP_T” on page 188
- “ESS_TRANSACTION_REQSPECIFIC_T” on page 188
- EssListTransactions

### EssRenameApplication

 Renames an existing application, either on the client or the server. If the application is running on the server, it is first stopped.

**Syntax**

```c
ESS_FUNC_M EssRenameApplication (hCtx, OldName, NewName);
```

**Parameter** | **Data Type** | **Description**
---|---|---
hCtx | ESS_HCTX_T | API context handle.
OldName | ESS_STR_T | Name of existing application to rename.
NewName | ESS_STR_T | New name of application. See “Application Name Limits” on page 1727.

**Notes**

Renaming a client application renames the local application directory.
**Return Value**

None.

**Access**

For a server application, the caller must have Application Create/Delete/Edit privilege (ESS_PRIV_APPCREATE).

**Example**

```c
ESS_FUNC_M
ESS_RenameApp (ESS_HCTX_T hCtx)
{
    ESS_FUNC_M  sts = ESS_STS_NOERR;
    ESS_STR_T  OldName;
    ESS_STR_T  NewName;

    OldName = "Sample";
    NewName = "Sample2";
    sts = EssRenameApplication(hCtx, OldName, NewName);

    return (sts);
}
```

**See Also**

- EssRenameDatabase
- EssRenameObject

## EssRenameDatabase

Renames an existing database within an application, either on the client or the server. If the database is running on the server, it is first stopped.

**Syntax**

```c
ESS_FUNC_M EssRenameDatabase (hCtx, AppName, OldName, NewName);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>AppName</td>
<td>ESS_STR_T</td>
<td>Application name.</td>
</tr>
<tr>
<td>OldName</td>
<td>ESS_STR_T</td>
<td>Name of existing database to rename.</td>
</tr>
<tr>
<td>NewName</td>
<td>ESS_STR_T</td>
<td>New name of database. See &quot;Database Name Limits&quot; on page 1728.</td>
</tr>
</tbody>
</table>

**Notes**

Renaming a client database renames the local database directory.

**Return Value**

None.
## Access

For a server database, the caller must have Database Create/Delete/Edit privilege (ESS_PRIV_DBCREATE).

## Example

```c
ESS_FUNC_M
ESS_RenameDatabase (ESS_HCTX_T hCtx)
{
    ESS_FUNC_M  sts;
    ESS_STR_T  AppName;
    ESS_STR_T   OldName;
    ESS_STR_T   NewName;

    AppName = "Sample";
    OldName = "Basic";
    NewName = "Basic2";

    sts = EssRenameDatabase(hCtx, AppName, OldName, NewName);

    return(sts);
}
```

## See Also

- EssRenameApplication
- EssRenameObject

### EssRenameFilter

Renames an existing filter.

## Syntax

```
ESS_FUNC_M EssRenameFilter (hCtx, AppName, DbName, OldName, NewName);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle</td>
</tr>
<tr>
<td>AppName</td>
<td>ESS_STR_T</td>
<td>Application name</td>
</tr>
<tr>
<td>DbName</td>
<td>ESS_STR_T</td>
<td>Database name</td>
</tr>
<tr>
<td>OldName</td>
<td>ESS_STR_T</td>
<td>Old name of existing filter to be renamed</td>
</tr>
<tr>
<td>NewName</td>
<td>ESS_STR_T</td>
<td>New name of renamed filter. See “Filter Name Limits” on page 1728.</td>
</tr>
</tbody>
</table>

## Notes

The old filter name must already exist, and the destination filter name must not exist.

## Return Value

None.
Access

This function requires the caller to have database Design privilege (ESS_PRIV_DBDESIGN) for the specified database.

Example

ESS_FUNC_M
ESS_RenameFilter (ESS_HCTX_T  hCtx)
{
    ESS_FUNC_M        sts = ESS_STS_NOERR;
    ESS_STR_T        AppName;
    ESS_STR_T        DbName;
    ESS_STR_T        OldName;
    ESS_STR_T        NewName;

    AppName = "Sample";
    DbName  = "Basic";
    OldName = "Test";
    NewName = "NewTest";

    sts = EssRenameFilter(hCtx, AppName, DbName,
                          OldName, NewName);
    return (sts);
}

See Also

- EssCopyFilter
- EssDeleteFilter
- EssListFilters

EssRenameGroup

Renames an existing group.

Syntax

ESS_FUNC_M  EssRenameGroup (hCtx, OldName, NewName);

Parameter  Data Type  Description
hCtx        ESS_HCTX_T  API context handle.
OldName     ESS_STR_T    Old name of existing group to rename.
NewName     ESS_STR_T    New name for renamed group. See “Group Name Limits” on page 1728.

Notes

- The specified new group name must not already exist.

Return Value

None.
**Access**

This function requires the caller to have Create/Delete User privilege (ESS_PRIV_USERCREATE) for the logged in server.

**Example**

```c
ESS_FUNC_M
ESST_RenameGroup (ESS_HCTX_T hCtx)
{
    ESS_FUNC_M sts = ESS_STS_NOERR;
    ESS_STR_T OldName;
    ESS_STR_T NewName;
    OldName = "PowerUsers";
    NewName = "PowerGroup";

    sts = EssRenameGroup (hCtx, OldName, NewName);
    return (sts);
}
```

**See Also**

- [EssCreateGroup](#)
- [EssDeleteGroup](#)
- [EssListGroups](#)

## EssRenameObject

Renames an existing object on the server or client object system.

**Syntax**

```c
ESS_FUNC_M EssRenameObject (hCtx, ObjType, AppName, DbName,
OldName, NewName);
```

**Parameter** | **Data Type** | **Description**
---|---|---
**hCtx** | ESS_HCTX_T | API context handle. Can be local context handle returned by EssCreateLocalContext().
**ObjType** | ESS_OBJTYPE_T | object type (must be single type). Refer to "Bitmask Data Types (C)" on page 96 for a list of possible values
**AppName** | ESS_STR_T | Application name
**DbName** | ESS_STR_T | database name. If NULL, uses the Application subdirectory
**OldName** | ESS_STR_T | Old name of object to rename
**NewName** | ESS_STR_T | New name of renamed object. See "Object Name Limits" on page 1728.

**Notes**

- To rename an object, the object must not be locked, and the new object must not already exist.
- Outline objects and LRO objects cannot be renamed.
Use the EssRenameDatabase() function to rename a database, including its associated outline.

Objects cannot be renamed across different applications or databases. Use the EssCopyObject() function to copy an object to another application or database.

**Return Value**

None.

**Access**

This function requires the caller to have Application or Database Design privilege (ESS_PRIV_APPDESIGN or ESS_PRIV_DBDESIGN) for the specified application or database containing the object.

**Example**

```c
ESS_STS_T Ess_renameObject (ESS_HCTX_T hCtx)
{
    ESS_STS_T         sts = ESS_STS_NOERR;
    ESS_STR_T         AppName;
    ESS_STR_T         DbName;
    ESS_STR_T         OldName;
    ESS_STR_T         NewName;
    ESS_OBJTYPE_T     ObjType;

    AppName    = "Sample";
    DbName     = "Basic";
    OldName    = "Test";
    NewName    = "NewTest";
    ObjType    = ESS_OBJTYPE_TEXT;

    sts = Ess_renameObject (hCtx, ObjType, AppName, DbName, OldName, NewName);

    if(!sts)
        printf("The Object is renamed.\r\n");

    return (sts);
}
```

**See Also**

- EssCopyObject
- EssCreateObject
- EssDeleteObject
- EssListObjects

**EssRenameUser**

Renames an existing user.
Syntax
ESS_FUNC_M Essrenameuser (hCtx, OldName, NewName);

Parameter | Data Type | Description
---|---|---
hCtx | ESS_HCTX_T | API context handle.
OldName | ESS_STR_T | Old name of existing user to rename.
NewName | ESS_STR_T | New name for renamed user. See “User Name Limits” on page 1728.

Notes
The specified new user name must not already exist.

Return Value
None.

Access
This function requires the caller to have Create/Delete User privilege (ESS_PRIV_USERCREATE) for the logged in server.

Example
ESS_FUNC_M
ESS_RenameUser (ESS_HCTX_T hCtx)
{
    ESS_FUNC_M  sts = ESS_STS_NOERR;
    ESS_STR_T   OldName;
    ESS_STR_T   NewName;

    OldName = "Jim Smith";
    NewName = "Tom Smith";

    sts = EssRenameUser (hCtx, OldName, NewName);

    return (sts);
}

See Also
- EssCreateUser
- EssDeleteUser
- EssListUsers

EssReport
Sends a report specification to the active database as a single string. This function is equivalent to making a call to EssBeginReport(), followed by calls to EssSendString() and finally EssEndReport(). The report data can either be output, or the report specification can just be verified and any errors returned. Also, the corresponding data blocks in the database can optionally be locked by this call (lock for update).
Syntax

```c
ESS_FUNC_M EssReport (hCtx, Output, Lock, RptSpec);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>Output</td>
<td>ESS_BOOL_T</td>
<td>Controls output of data. If TRUE, data is output from the server, according to the specified report. If FALSE, no data is output.</td>
</tr>
<tr>
<td>Lock</td>
<td>ESS_BOOL_T</td>
<td>Controls block locking. If TRUE, all blocks which are accessed by the report specification are locked for update. If FALSE, no blocks are locked.</td>
</tr>
<tr>
<td>RptSpec</td>
<td>ESS_STR_T</td>
<td>The report specification, as a single string (must be less than 64 KB).</td>
</tr>
</tbody>
</table>

**Notes**
- The report specification string must be less than 64 KB long.
- If this function causes data to be output (Output flag is TRUE), the returned data must be read by calling `EssGetString()` until a NULL is returned.
- If this function causes blocks to be locked (Lock flag is TRUE), the caller is responsible for unlocking the locked blocks (e.g. by calling `EssUpdate()` with the Unlock flag set to TRUE).
- If both the Output and Lock flags are set to FALSE, the database merely performs a syntax check of the report specification.

**Return Value**
None.

**Access**
This function requires the caller to have read privilege (ESS_PRIV_READ) to one or more members in the active database. Any members that the caller does not have access to will be returned as missing.

**Example**

```c
ESS_FUNC_M
ESS_ReportLine (ESS_HCTX_T       hCtx,
                ESS_HINST_T  hInst
        )
{
    ESS_FUNC_M       sts = ESS_STS_NOERR;
    ESS_STR_T     rString;

    sts = EssReport (hCtx, ESS_TRUE, ESS_FALSE,
                     "<Desc Year !");
    /***************
    * Get the report *
    *************** /

    if (!sts)
        sts = EssGetString (hCtx, &rString);
    while (((!sts) && (rString != NULL))
    {
```
printf ("%s", rString);
EssFree (hInst, rString);
sts = EssGetString (hCtx, &rString);
}
printf ("\r\n ");

return (sts);
}

See Also

- EssBeginReport
- EssEndReport
- EssGetString
- EssReportFile
- EssUpdate

EssReportFile

Sends a report specification to the active database from a file. The report data can either be output, or the report specification can just be verified and any errors returned. Also, the corresponding data blocks in the database can optionally be locked by this call (lock for update).

Syntax

```
ESS_FUNC_M EssReportFile (hDestCtx, hSrcCtx, AppName, DbName, FileName, Output, Lock);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hDestCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle of target database on the server.</td>
</tr>
<tr>
<td>hSrcCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle for report file location. The report file can reside on the client or on the same server as the target database. If the report file is on the client (local), the local context must be created with <code>EssCreateLocalContext</code>.</td>
</tr>
<tr>
<td>AppName</td>
<td>ESS_STR_T</td>
<td>Application name for report file location.</td>
</tr>
<tr>
<td>DbName</td>
<td>ESS_STR_T</td>
<td>Database name for report file location.</td>
</tr>
<tr>
<td>FileName</td>
<td>ESS_STR_T</td>
<td>Name of report specification file. It is not necessary to specify the file extension; the extension is understood to be .rep.</td>
</tr>
<tr>
<td>Output</td>
<td>ESS_BOOL_T</td>
<td>Controls output of data. If TRUE, data is output from the server, according to the specified report. If FALSE, no data is output.</td>
</tr>
<tr>
<td>Lock</td>
<td>ESS_BOOL_T</td>
<td>Controls block locking. If TRUE, all blocks which are accessed by the report specification are locked for update. If FALSE, no blocks are locked.</td>
</tr>
</tbody>
</table>

Notes

- If this function causes data to be output (`Output` flag is TRUE), the returned data can be read by calling `EssGetString()`.
- If this function causes blocks to be locked (`Lock` flag is TRUE), the caller is responsible for unlocking the locked blocks (e.g. by calling `EssUpdate()` with the `Unlock` flag set to TRUE).
If both the Output and Lock flags are set to FALSE, the database merely performs a syntax check of the report specification.

Return Value
None.

Access
This function requires the caller to have read privilege (ESS_PRIV_READ) to one or more members in the active database.

Example

```
ESS_FUNC_M
ESS_ReportFile (ESS_HCTX_T  hCtx,
                ESS_HINST_T hInst
        )
{
    ESS_FUNC_M    sts = ESS_STS_NOERR;
    ESS_HCTX_T   hSrcCtx;
    ESS_STR_T    rString;
    ESS_STR_T    AppName;
    ESS_STR_T    DbName;
    ESS_STR_T    FileName;

    hSrcCtx = hCtx;
   AppName = "Sample";
    DbName  = "Basic";
    FileName = "Test";

    sts = EssReportFile (hCtx, hSrcCtx, AppName,
                         DbName, FileName, ESS_TRUE, ESS_FALSE);
    /* Get the report */
    if (!sts)
        sts = EssGetString (hCtx, &rString);
    while ((!sts) && (rString != NULL))
    {
        printf ("%s", rString);
        EssFree (hInst, rString);
        sts = EssGetString (hCtx,&rString);
    }
    return(sts);
}
```

See Also
- EssBeginReport
- EssGetString
- EssReport
- EssUpdateFile

**EssReRegisterApplication**

Re-establishes one or all Essbase applications as Shared Services applications.
Syntax

ESS_FUNC_M EssReRegisterApplication (hCtx,AppName,AllApps);

Parameter | Data Type | Description
--- | --- | ---
hCtx | ESS_HCTX_T | API context handle.
AppName | ESS_STR_T | Name of application to reregister.
AllApps | ESS_BOOL_T | If ESS_TRUE, all applications are reregistered; otherwise, only the named application is reregistered.

Return Value

Returns 0 if successful; otherwise, returns an error.

Access

This function requires the caller to be an Administrator, Application Manager, or Database Manager. For any applications for which the caller does not have sufficient permissions, the reregistration will be skipped with a warning.

Example

```c
ESS_FUNC_M ESS_SS_ReRegisterApplication(ESS_HCTX_T hCtx, ESS_HINST_T hInst)
{
    ESS_STS_T         sts = ESS_STS_NOERR;
    ESS_BOOL_T        allApps;
    ESS_STR_T         appName = ESS_NULL;
    sts = EssAlloc(hInst, sizeof(ESS_APPNAME_T), &appName);
    if(sts)
        return (sts);
    memset(appName, 0, sizeof(ESS_APPNAME_T));
    strcpy( appName, "Sample");
    /* Do you want All applications re-registered?
    * Enter ESS_TRUE for Yes
    * ESS_FALSE for No
    **/
    allApps = ESS_FALSE; /* Re-registering only 1 application */
    sts = EssReRegisterApplication(hCtx, appName, allApps);
    if (sts)
        printf("Failed to Re-register Application %s\n", appName);
    if (appName)
        EssFree(hInst, appName);

    return (sts);
}
```

See also an extended Appendix B
**EssResetDatabase**

Clears all loaded data and resets the outline to be empty in the active database.

**Syntax**

```
ESS_FUNC_M EssResetDatabase (hCtx);
```

**Parameter**  
**Data Type**  
**Description**

- `hCtx`  
  ESS_HCTX_T  
  API context handle.

**Notes**

- Data deleted and outlines reset using this function cannot be restored. Use it with care!
- This function call is asynchronous. Call `EssGetProcessState()` after making this call until `EssGetProcessState()` returns a status indicating that the reset database operation is complete.

**Return Value**

None.

**Access**

This function requires that the caller have Write privilege (ESS_PRIV_WRITE) for the database, and to select it as the active database using `EssSetActive()`.

**Example**

```c
ESS_FUNC_M
ESS_ResetDb (ESS_HCTX_T     hCtx)
{
    ESS_FUNC_M        sts = ESS_STS_NOERR;
    ESS_PROCSTATE_T   pState;
    sts = EssResetDatabase(hCtx);

    if (!sts)
    {
        sts = EssGetProcessState (hCtx, &pState);
        while(!sts && (pState.State != ESS_STATE_DONE))
            sts = EssGetProcessState (hCtx, &pState);
    }
    return (sts);
}
```

**See Also**

- `EssResetPerfStats`
- `EssDumpPerfStats`

**EssResetPerfStats**

Resets values in the performance statistics tables to zero.
Syntax

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx;</td>
<td>ESS_HCTX_T</td>
<td>API context handle</td>
</tr>
<tr>
<td>persistence;</td>
<td>ESS_ULONG_T</td>
<td>One of the following values indicating the persistence of the set of tables to be reset:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0: Reset short term tables only</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1: Reset short and medium term tables</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2: Reset short, medium and long term tables</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3: Disable performance statistics gathering</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4: Enable performance statistics gathering</td>
</tr>
<tr>
<td>scope;</td>
<td>ESS_ULONG_T</td>
<td>One of the following values indicating the scope of the set of tables to be reset:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1: Reset thread-based tables only</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2: Reset database-based tables only</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4: Reset server-based tables only</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7: Reset all tables</td>
</tr>
</tbody>
</table>

Notes

- Enabling statistics gathering (persistence 4) or disabling statistics gathering (persistence 3) does not reset any statistics.
- For more information on performance statistics tables, see the ESSCMD commands GETPERFSTATS and RESETPERFSTATS in the Oracle Essbase Technical Reference.

Return Value

If successful, EssResetPerfStats returns 0.

Access

The caller of this function must have supervisor access.

Example

/* This function resets all short term tables */

ESS_STS_T ESSResetPerfStats(ESS_HCTX_T *context) {
    ESS_STS_T sts = ESS_STS_NOERR;
    ESS_ULONG_T persistence = 0;
    ESS_ULONG_T scope = 7;

    sts = EssResetPerfStats(context, persistence, scope);

    return sts;
}
EssResetUser

Resets the user's security structure to its initial state.

Syntax

```c
ESS_FUNC_M EssResetUser (hCtx, UserName);
```

Parameter | Data Type | Description
---|---|---
hCtx; | ESS_HCTX_T | API context handle
UserName; | ESS_STR_T | User name

Notes

The following user security parameters are reset to their initial state:

- `LockedOut`
- `PwdChgNow`
- `Failcount`
- `LastLogin`
- `LastPwdChg`
- `Expiration`

Return Value

Returns zero (0) if successful.

Access

This function requires the caller to have Create/Delete User privilege (ESS_PRIV_USERCREATE) for the logged in server.

Example

```c
ESS_FUNC_M
Ess_ResetUser (ESS_HCTX_T hCtx)
{
    ESS_FUNC_M      sts = ESS_STS_NOERR;
    ESS_STR_T      UserName = "William";

    sts = EssResetUser (hCtx, UserName);
    return (sts);
}
```

See Also

- `EssDumpPerfStats`
- `EssGetStatBufSize`
EssRestore

No longer in use.

This function is retained for compatibility with earlier releases of Essbase only. For current Essbase archiving, see EssArchiveBegin() and EssArchiveEnd(). This function now returns the error message ESS_STS_OBSOLETE.

See Also
EssArchiveBegin
EssArchiveEnd
EssArchive
EssSetActive

EssRestoreDatabase

Restores a database from a backup, archive file that you specify.

Syntax

ESS_FUNC_M EssRestoreDatabase (hCtx, AppName, DbName, BackupFileName, bForceDiffName, Count, ReplaceVol);

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>Login context</td>
</tr>
<tr>
<td>AppName</td>
<td>ESS_STR_T</td>
<td>Application Name</td>
</tr>
<tr>
<td>DbName</td>
<td>ESS_STR_T</td>
<td>Database Name</td>
</tr>
<tr>
<td>BackupFileName</td>
<td>ESS_STR_T</td>
<td>Full path to the backup file from which to read archive data. Specify the full path, for example: c:\hyperion\Test.arc</td>
</tr>
</tbody>
</table>

Note: Works only at the database level. The AppName parameter specifies an Application in order to access the database residing within.
### Parameter Table

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bForceDiffName</td>
<td>ESS_BOOL_T</td>
<td>Use a different application and database names for the restore.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_TRUE—force a different application and/or database name for the restore.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If you use ESS_TRUE and the application and database name is same as the backup, the result is the same as ESS_FALSE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_FALSE—use application and database names stored in the backup file. Verifies that the names in backup file are same the ones to which you are restoring to.</td>
</tr>
<tr>
<td>Count</td>
<td>ESS_USHORT_T</td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Number of disk volume replacement structures that are being restored.</td>
</tr>
<tr>
<td>ReplaceVol</td>
<td>ESS_PDISKVOLUME_REPLACE_T</td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Disk volume replacement input structures.</td>
</tr>
</tbody>
</table>

### Return Value

Returns:

- 0—If successful
- Error number—If unsuccessful

### Access

The caller must have Essbase Administrator access to the database.

### Example

```c
void RestoreDB()
{
    ESS_FUNC_M        sts = ESS_STS_NOERR;
    ESS_STR_T            AppName = "Backup";
    ESS_STR_T            DbName = "Basic";
    ESS_STR_T            BackupFileName = "F:\\testArea\\ArchiveAndRestore\\TempBackup.arc";
    ESS_STR_T            optionsFileName = "";
    ESS_BOOL_T        bOverWrite;
    ESS_BOOL_T        bForceDiffName;
    ESS_USHORT_T        count;
    ESS_PDISKVOLUME_REPLACE_T    replaceVol;

    printf("\nArchive DB:\n");
    bOverWrite = ESS_TRUE;
    sts = EssArchiveDatabase(hCtx, AppName, DbName, BackupFileName, optionsFileName, bOverWrite);
    printf("EssArchiveDatabase sts: %ld\r\n",sts);
    sts = EssUnloadApplication(hCtx, AppName);
    printf("\nEssUnloadApplication sts: %ld\r\n",sts);
```
printf("\nCase with no volume replacement:\n");
bForceDiffName = ESS_FALSE;
count = 0;
replaceVol = ESS_NULL;
sts = EssRestoreDatabase (hCtx, AppName, DbName, BackupFileName, bForceDiffName, count, replaceVol);
printf("EssRestoreDatabase sts: %ld\n",sts);

printf("\nCase with a replacement volume (index and page files to a different volume):\n");
bForceDiffName = ESS_FALSE;
count = 1;
if (count)
{
    sts = EssAlloc(hInst, count * sizeof(ESS_DISKVOLUME_REPLACE_T), (ESS_PPVOID_T)&replaceVol);
    memset(replaceVol, 0, count * sizeof(ESS_DISKVOLUME_REPLACE_T));
}
strcpy(replaceVol->szPartition_Src, "C");
strcpy(replaceVol->szPartition_Dest, "F");
sts = EssUnloadApplication(hCtx, AppName);
printf("\nEssUnloadApplication sts: %ld\n",sts);

sts = EssRestoreDatabase (hCtx, AppName, DbName, BackupFileName, bForceDiffName, count, replaceVol);
printf("EssRestoreDatabase sts: %ld\n",sts);
if (replaceVol)
    EssFree(hInst, replaceVol);
}

See Also

EssArchiveDatabase

EssSendString

Sends a string of data to the active database. The string must be less than 32 KB long. This function should be called after EssBeginReport(), EssBeginUpdate(), or EssBeginCalc().

Syntax

ESS_FUNC_M EssSendString (hCtx, String);

Parameter  Data Type  Description

hCtx        ESS_HCTX_T  API context handle.
String      ESS_STR_T    Data string (must be less than 32 KB in length.)
Notes

- Calling this function other than after successfully executing a begin report, update or calculate function will generate an error.
- The string to be sent must be less than 32 KB long.
- When you are using this function with `EssBeginUpdate()`, you must end the update string with a carriage return or line feed character.

Return Value

None.

Access

This function requires no special privileges.

Example

```c
ESS_FUNC_M
ESS_Report (ESS_HCTX_T    hCtx,
             ESS_HINST_T  hInst
             )
{
  ESS_FUNC_M     sts     = ESS_STS_NOERR;
  ESS_STR_T     rString = NULL;
  sts = EssBeginReport (hCtx, ESS_TRUE, ESS_FALSE);
  if (!sts)
    sts = EssSendString (hCtx, "<Desc Year !");
  if (!sts)
    sts = EssEndReport (hCtx);

  /***************
  * Get report *
  ***************/
  if (!sts)
    sts = EssGetString (hCtx, &rString);
  while (((!sts) && (rString != NULL))
  {
    printf ("%s", rString);
    EssFree (hInst, rString);
    sts = EssGetString (hCtx, &rString);
  }
  printf ("\r\n");
  return(sts);
}
```

Unicode clients using the C Main API to communicate with Unicode-enabled Essbase applications must use this function to send the UTF-8 encoded Unicode byte order mark (BOM) in the text stream. For an example, see “Specifying the Byte Order Encoding” on page 75.

See Also

- `EssBeginCalc`
- `EssBeginReport`
**EssSetActive**

Sets the caller's active application and database.

**Syntax**

```c
ESS_FUNC_M EssSetActive (hCtx,AppName,DbName,pAccess);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>AppName</td>
<td>ESS_STR_T</td>
<td>Application name.</td>
</tr>
<tr>
<td>DbName</td>
<td>ESS_STR_T</td>
<td>Database name.</td>
</tr>
<tr>
<td>pAccess</td>
<td>ESS_PACCESS_T</td>
<td>Address of variable to receive the user's access level to the selected database. See &quot;Bitmask Data Types (C)&quot; on page 96 for a list of possible values for this field.</td>
</tr>
</tbody>
</table>

**Notes**

- If the application and database have not been loaded, this function will load them.
- The `EssAutoLogin()` function can also be used to allow a user to login and set the active application and database.

**Return Value**

If successful, returns the user's access level to the selected application and database in `pAccess`.

**Access**

This function requires no special privileges.

**Example**

```c
ESS_FUNC_M
ESS_SetActive (ESS_HCTX_T hCtx)
{
    ESS_FUNC_M sts = ESS_STS_NOERR;
    ESS_ACCESS_T Access;
    ESS_STR_T AppName;
    ESS_STR_T DbName;

    AppName = "Sample";
    DbName = "Basic";

    sts = EssSetActive (hCtx,AppName,DbName,&Access);
    return (sts);  
}
```
See Also
- EssClearActive
- EssGetActive
- EssListApplications
- EssListDatabases
- EssLogin

**EssSetAlias**

Sets the active alias table in the active database for a user.

**Syntax**

```c
ESS_FUNC_M EssSetAlias (hCtx, AliasName);
```

**Parameter** | **Data Type** | **Description**
--- | --- | ---
`hCtx` | ESS_HCTX_T | API context handle.
`AliasName` | ESS_STR_T | Name of alias table to set active.

**Return Value**

None.

**Example**

```c
ESS_FUNC_M
Ess_SetAlias (ESS_HCTX_T  hCtx)
{
    ESS_FUNC_M     sts = ESS_STS_NOERR;
    ESS_STR_T     AliasName;
    AliasName = "TestAlias";
    sts = EssSetAlias (hCtx, AliasName);

    return (sts);
}
```

See Also
- EssGetAlias
- EssListAliases

**EssSetApplicationAccess**

Sets a list of user application access structures, which contain information about user access to applications.

**Syntax**

```c
ESS_FUNC_M EssSetApplicationAccess (hCtx, Count, pUserApp);
```
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>Count</td>
<td>ESS_USHORT_T</td>
<td>Count of user application structures.</td>
</tr>
<tr>
<td>pUserApp</td>
<td>“ESS_USERAPP_T, ESS_GROUPAPP_T” on page 189</td>
<td>Pointer to an array of user application structures.</td>
</tr>
</tbody>
</table>

**Notes**

- The *Access* field of the user application structure is used to set the user’s granted access to the application. For this call the *MaxAccess* field is ignored.

**Return Value**

None.

**Access**

This function requires the caller to have application Design privilege (ESS_PRIV_APPDESIGN) for the specified application.

**Example**

```c
ESS_FUNC_M
ESS_SetApplicationAccess (ESS_HCTX_T hCtx)
{
    ESS_FUNC_M      sts = ESS_STS_NOERR;
    ESS_USHORT_T  Count;
    ESS_USERAPP_T UserApp;
    strcpy(UserApp.UserName,"Jim Smith");
    strcpy(UserApp.AppName,"Sample");
    UserApp.Access     = ESS_PRIV_APPDESIGN;
    UserApp.MaxAccess  = ESS_PRIV_APPDESIGN;
    sts = EssSetApplicationAccess(hCtx, Count,
                                 &UserApp);
    return (sts);
}
```

**See Also**

- EssSetApplicationAccessEx
- EssGetApplicationAccess
- EssListUsers
- EssSetDatabaseAccess
- EssSetUser

**EssSetApplicationAccessEx**

Sets a list of user application access structures, which contain information about user access to applications. Similar to EssSetApplicationAccess, but the input structure can include user directories and unique identity attributes.

**Syntax**

```c
ESS_FUNC_M EssSetApplicationAccessEx (hCtx, Count, pUserApp);
```
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle (input).</td>
</tr>
<tr>
<td>Count</td>
<td>ESS_USHORT_T</td>
<td>Count of user application structures (input).</td>
</tr>
<tr>
<td>pUserApp</td>
<td>ESS_PUSERAPPEX_T</td>
<td>Pointer to an array of user application structures (input).</td>
</tr>
</tbody>
</table>

**Notes**

The *Access* field of the user application structure is used to set the user’s granted access to the application. For this call the *MaxAccess* field is ignored.

**Return Value**

None.

**Access**

This function requires the caller to have application Design privilege (ESS_PRIV_APPDESIGN) for the specified application.

**Example**

```c
void DisplayUserAppInfo(ESS_PUSERAPPEX_T userApp, ESS_USHORT_T count)
{
    ESS_STS_T sts = ESS_STS_NOERR;
    ESS_USHORT_T ind;

    printf ("\n------Application Access List-----\n\n");
    for (ind = 0; ind < count; ind++)
    {
        printf("\tUser: %s\n", userApp[ind].UserName);
        printf("\tProvider Name: %s\n", userApp[ind].ProviderName);
        printf("\tConnection Param: %s\n", userApp[ind].connparam);
        printf("\tAppName: %s\n", userApp[ind].AppName);
        printf("\tAccess: %d - ESS_PRIV_\n", userApp[ind].Access);

        switch(userApp[ind].Access)
        {
            case ESS_PRIV_NONE:
                printf("\tAccess: %d - ESS_PRIV_NONE\n", userApp[ind].Access);
                break;
            case ESS_PRIV_READ:
                printf("\tAccess: %d - ESS_PRIV_READ\n", userApp[ind].Access);
                break;
            case ESS_PRIV_WRITE:
                printf("\tAccess: %d - ESS_PRIV_WRITE\n", userApp[ind].Access);
                break;
            case ESS_PRIV_CALC:
                printf("\tAccess: %d - ESS_PRIV_CALC\n", userApp[ind].Access);
                break;
            case ESS_PRIV_METAREAD:
                printf("\tAccess: %d - ESS_PRIV_M\n", userApp[ind].Access);
                break;
            case ESS_PRIV_DBLOAD:
                printf("\tAccess: %d - ESS_PRIV_DBLOAD\n", userApp[ind].Access);
                break;
        }
    }
}
```
case ESS_PRIV_DBMANAGE:
    printf("\tAccess: %d - ESS_PRIV_DBMANAGE\n", userApp[ind].Access);
    break;

case ESS_PRIV_DBCREATE:
    printf("\tAccess: %d - ESS_PRIV_DBCREATE\n", userApp[ind].Access);
    break;

case ESS_PRIV_APPLOAD:
    printf("\tAccess: %d - ESS_PRIV_APPLOAD\n", userApp[ind].Access);
    break;

case ESS_PRIV_APPMANAGE:
    printf("\tAccess: %d - ESS_PRIV_APPMANAGE\n", userApp[ind].Access);
    break;

case ESS_PRIV_APPCREATE:
    printf("\tAccess: %d - ESS_PRIV_APPCREATE\n", userApp[ind].Access);
    break;

case ESS_PRIV_USERCREATE:
    printf("\tAccess: %d - ESS_PRIV_USERCREATE\n", userApp[ind].Access);
    break;

case ESS_ACCESS_READ:
    printf("\tAccess: %d - ESS_ACCESS_READ\n", userApp[ind].Access);
    break;

case ESS_ACCESS_WRITE:
    printf("\tAccess: %d - ESS_ACCESS_WRITE\n", userApp[ind].Access);
    break;

case ESS_ACCESS_CALC:
    printf("\tAccess: %d - ESS_ACCESS_CALC\n", userApp[ind].Access);
    break;

case ESS_ACCESS_METAREAD:
    printf("\tAccess: %d - ESS_ACCESS_METAREAD\n", userApp[ind].Access);
    break;

case ESS_ACCESS_DBMANAGE:
    printf("\tAccess: %d - ESS_ACCESS_DBMANAGE\n", userApp[ind].Access);
    break;

case ESS_ACCESS_DBCREATE:
    printf("\tAccess: %d - ESS_ACCESS_DBCREATE\n", userApp[ind].Access);
    break;

case ESS_ACCESS_APPMANAGE:
    printf("\tAccess: %d - ESS_ACCESS_APPMANAGE\n", userApp[ind].Access);
    break;

case ESS_ACCESS_APPCREATE:
    printf("\tAccess: %d - ESS_ACCESS_APPCREATE\n", userApp[ind].Access);
    break;

case ESS_ACCESS_FILTER:
    printf("\tAccess: %d - ESS_ACCESS_FILTER\n", userApp[ind].Access);
    break;

case ESS_ACCESS_DBALL:
    printf("\tAccess: %d - ESS_ACCESS_DBALL\n", userApp[ind].Access);
    break;

case ESS_ACCESS_APPALL:
    printf("\tAccess: %d - ESS_ACCESS_APPALL\n", userApp[ind].Access);
    break;

case ESS_ACCESS_ADMIN:
    printf("\tAccess: %d - ESS_ACCESS_ADMIN\n", userApp[ind].Access);
    break;

default:
    printf("\tAccess: Unknown\n");
switch(userApp[ind].MaxAccess)
{
    case ESS_PRIV_NONE:
        printf("\tMax Access: %d - ESS_PRIV_NONE\n", userApp[ind].MaxAccess);
        break;
    case ESS_PRIV_READ:
        printf("\tMax Access: %d - ESS_PRIV_READ\n", userApp[ind].MaxAccess);
        break;
    case ESS_PRIV_WRITE:
        printf("\tMax Access: %d - ESS_PRIV_WRITE\n", userApp[ind].MaxAccess);
        break;
    case ESS_PRIV_CALC:
        printf("\tMax Access: %d - ESS_PRIV_CALC\n", userApp[ind].MaxAccess);
        break;
    case ESS_PRIV_METAREAD:
        printf("\tMax Access: %d - ESS_PRIV_METAREAD\n", userApp[ind].MaxAccess);
        break;
    case ESS_PRIV_DBLOAD:
        printf("\tMax Access: %d - ESS_PRIV_DBLOAD\n", userApp[ind].MaxAccess);
        break;
    case ESS_PRIV_DBMANAGE:
        printf("\tMax Access: %d - ESS_PRIV_DBMANAGE\n", userApp[ind].MaxAccess);
        break;
    case ESS_PRIV_DBCREATE:
        printf("\tMax Access: %d - ESS_PRIV_DBCREATE\n", userApp[ind].MaxAccess);
        break;
    case ESS_PRIV_APPLOAD:
        printf("\tMax Access: %d - ESS_PRIV_APPLOAD\n", userApp[ind].MaxAccess);
        break;
    case ESS_PRIV_APPMANAGE:
        printf("\tMax Access: %d - ESS_PRIV_APPMANAGE\n", userApp[ind].MaxAccess);
        break;
    case ESS_PRIV_APPCREATE:
        printf("\tMax Access: %d - ESS_PRIV_APPCREATE\n", userApp[ind].MaxAccess);
        break;
    case ESS_PRIV_USERCREATE:
        printf("\tMax Access: %d - ESS_PRIV_USERCREATE\n", userApp[ind].MaxAccess);
        break;
    case ESS_ACCESS_READ:
        printf("\tMax Access: %d - ESS_ACCESS_READ\n", userApp[ind].MaxAccess);
        break;
    case ESS_ACCESS_WRITE:
        printf("\tMax Access: %d - ESS_ACCESS_WRITE\n", userApp[ind].MaxAccess);
        break;
    case ESS_ACCESS_CALC:
        printf("\tMax Access: %d - ESS_ACCESS_CALC\n", userApp[ind].MaxAccess);
        break;
    case ESS_ACCESS_METAREAD:
        printf("\tMax Access: %d - ESS_ACCESS_METAREAD\n", userApp[ind].MaxAccess);
        break;
    case ESS_ACCESS_DBMANAGE:
        printf("\tMax Access: %d - ESS_ACCESS_DBMANAGE\n", userApp[ind].MaxAccess);
        break;
    case ESS_ACCESS_DBCREATE:
        printf("\tMax Access: %d - ESS_ACCESS_DBCREATE\n", userApp[ind].MaxAccess);
        break;
}
C Main API Functions

```c

ESS_FUNC_M ESS_SetApplicationAccessEx (ESS_HCTX_T hCtx, ESS_HINST_T hInst)
{
    ESS_STS_T sts = ESS_STS_NOERR;
    ESS_STR_T userId;
    ESS_BOOL_T bIsIdentity;
    ESS_USHORT_T type;
    ESS_STR_T AppName;
    ESS_USHORT_T count;
    ESS_USERAPPEX_T userApp[2];
    ESS_PUSERAPPEX_T pUserApp = ESS_NULL;
    memset(&userApp, '0', sizeof(userApp));
    userId = "IDUser1";
    AppName = "";
    type = ESS_TYPE_USER;
    bIsIdentity = ESS_FALSE;
    count = 1;
    strcpy(userApp[0].UserName, "IDUser1");
    strcpy(userApp[0].ProviderName, "LDAP");
    strcpy(userApp[0].connparam, "");
    userApp[0].type = ESS_TYPE_USER;
    strcpy(userApp[0].AppName, AppName);
    userApp[0].Access = ESS_PRIV_APPMANAGE;
    userApp[0].MaxAccess = ESS_PRIV_APPMANAGE;
    sts = EssSetApplicationAccessEx(hCtx, count, &userApp);
    printf("EssSetApplicationAccessEx sts: %ld\n", sts);
}
```
if(!sts)
{
    userId = userApp[0].UserName;
    type = userApp[0].type;
    sts = EssGetApplicationAccessEx(hCtx, userId, bIsIdentity, type, AppName, &count, &pUserApp);
    printf("EssGetApplicationAccessEx sts: %ld\n", sts);
    if(!sts)
    {
        if(count && pUserApp)
        {
            DisplayUserAppInfo(pUserApp, count);
            sts = EssFree (hInst, pUserApp);
        }
        else
        {
            printf ("\rUser Application list is empty\n\n");
        }
    }
}

return (sts);
}

See Also

- EssGetApplicationAccessEx
- EssListUsersInfoEx
- EssSetDatabaseAccessEx

EssSetApplicationState

Sets user-configurable parameters for the application using the application’s state structure.

Syntax

ESS_FUNC_M EssSetApplicationState (hCtx, AppName, pAppState);

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>AppName</td>
<td>ESS_STR_T</td>
<td>Application name.</td>
</tr>
<tr>
<td>pAppState</td>
<td>“ESS_APPSTATE_T” on page 117</td>
<td>Pointer to application state structure.</td>
</tr>
</tbody>
</table>

Notes

- When changing parameter values, it is advisable to call EssGetApplicationState() first to get the correct values of any parameters you do not wish to change.
- The following parameters do not apply to aggregate storage databases: LockTimeout and IroSizeLimit.

Return Value

None.
Access

This function requires the caller to have application designer privilege (ESS_PRIV_APPDESIGN) for the specified application.

Example

```c
ESS_FUNC_M
ESS_SetAppState (ESS_HCTX_T hCtx,
                ESS_HINST_T hInst
             )
{
    ESS_FUNC_M         sts = ESS_STS_NOERR;
    ESS_PAPPSTATE_T   AppState;
    ESS_STR_T         AppName;
    AppName = "Sample";

    sts = EssGetApplicationState (hCtx, AppName,
                                &AppState);
    if (!sts)
    {
        if (AppState)
        {
             /*****************************/
            * Update AppState structure *
             *****************************/
            sts = EssSetApplicationState (hCtx,
                              AppName, AppState);
            EssFree (hInst, AppState);
        }
        return (sts);
    }
}
```

See Also

- EssGetApplicationState
- EssSetDatabaseState

**EssSetCalcList**

Sets the list of calc scripts objects which are accessible to a user.

Syntax

```c
ESS_FUNC_M   EssSetCalcList (hCtx, UserName, AppName, DbName, AllCalcs, Count,
                              pCalcList);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle</td>
</tr>
<tr>
<td>UserName</td>
<td>ESS_STR_T</td>
<td>User name</td>
</tr>
<tr>
<td>AppName</td>
<td>ESS_STR_T</td>
<td>Application name</td>
</tr>
<tr>
<td>DbName</td>
<td>ESS_STR_T</td>
<td>Database name. If NULL, uses Application subdirectory</td>
</tr>
<tr>
<td>Parameter</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>----------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>AllCalcs</td>
<td>ESS_BOOL_T</td>
<td>Allow all calcs flag. If TRUE, the user can access all calc scripts, otherwise, they can only access those specified in the CalcList argument.</td>
</tr>
<tr>
<td>Count</td>
<td>ESS_USHORT_T</td>
<td>Count of the number of accessible calc script objects</td>
</tr>
<tr>
<td>pCalcList</td>
<td>ESS_POBJNAME_T</td>
<td>Pointer to an array of calc script object names</td>
</tr>
</tbody>
</table>

**Notes**

- If the `AllCalcs` flag is set to TRUE, the `Count` and `pCalcList` arguments will be ignored.
- In order to access any calc script objects, the user must have at least calculate access to the appropriate database.

**Return Value**

None.

**Access**

This function requires the caller to have database Design privilege (ESS_PRIV_DBDESIGN) for the specified database.

**Example**

```c
ESS_FUNC_M
ESS_SetCalcList (ESS_HCTX_T hCtx)
{
    ESS_FUNC_M       sts = ESS_STS_NOERR;
    ESS_STR_T       UserName;
    ESS_STR_T       AppName;
    ESS_STR_T       DbName;
    ESS_BOOL_T      AllCalcs;
    ESS_USHORT_T    Count;
    ESS_OBJSNAME_T  pCalcList[3];

    UserName = "Newuser";
    AppName  = "Sample";
    DbName   = "Basic";
    AllCalcs = ESS_FALSE ;
    Count = 3;
    strcpy(pCalcList[0],"test1");
    strcpy(pCalcList[1],"test2");
    strcpy(pCalcList[2],"test3");

    sts = EssSetCalcList(hCtx, UserName, AppName,
                         DbName, AllCalcs, Count, pCalcList);

    return (sts);
}
```

**See Also**

- `EssSetCalcListEx`
- `EssGetCalcList`
- `EssListObjects`
EssListUsers

EssSetCalcListEx

Sets the calculation list accessible to the specified user or group. Similar to EssSetCalcList, but includes users and groups hosted in a user directory.

Syntax

ESS_FUNC_M EssSetCalcListEx (hCtx, UserId, bIsIdentity, entityType,AppName, DbName, AllCalc, count, pCalcList);

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle (input).</td>
</tr>
<tr>
<td>UserID</td>
<td>ESS_STR_T</td>
<td>Input. User or group for which to set the calculation list. Can be specified as name@provider or as a unique identity attribute.</td>
</tr>
<tr>
<td>bIsIdentity</td>
<td>ESS_BOOL_T</td>
<td>Input. Indicates whether UserID is a name or an identity. If TRUE, UserID is an identity.</td>
</tr>
<tr>
<td>entityType</td>
<td>ESS_USHORT_T</td>
<td>Input. Type of entity contained in UserID. Can be one of the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_TYPE_USER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_TYPE_GROUP</td>
</tr>
<tr>
<td>AppName</td>
<td>ESS_STR_T</td>
<td>Application name (input).</td>
</tr>
<tr>
<td>DbName</td>
<td>ESS_STR_T</td>
<td>Database name (input).</td>
</tr>
<tr>
<td>AllCalc</td>
<td>ESS_BOOL_T</td>
<td>Allow all calcs flag (input). If TRUE, the user or group can access all calculation scripts; otherwise, only those specified in CalcList are accessible.</td>
</tr>
<tr>
<td>Count</td>
<td>ESS_USHORT_T</td>
<td>Count of the number of accessible calculation script objects (input).</td>
</tr>
<tr>
<td>pCalcList</td>
<td>ESS_POBJNAME_T</td>
<td>Pointer to an array of accessible calculation script object names (input).</td>
</tr>
</tbody>
</table>

Notes

- If the AllCalcs flag is set to TRUE, the Count and pCalcList arguments will be ignored.
- In order to access any calculation script objects, the user must have at least calculate access to the appropriate database.

Return Value

None.

Access

This function requires the caller to have database Design privilege (ESS_PRIV_DBDESIGN) for the specified database.

Example

```c
void GetCalcList(ESS_STR_T userName)
```


{  
  ESS_STS_T sts = ESS_STS_NOERR;  
  ESS_BOOL_T AllCalcs = ESS_FALSE;  
  ESS_USHORT_T Count, ind;  
  ESS_POBJNAME_T pCalcList = NULL;  

  sts = EssGetCalcList(hCtx, userName, AppName, DbName, &AllCalcs, &Count, &pCalcList);  
  printf("EssGetCalcList sts: %ld\n", sts);  
  //sts = EssGetCalcListEx(hCtx, userName, AppName, DbName, &AllCalcs, &Count, 
  //&pCalcList);  
  //printf("EssGetCalcListEx sts: %ld\n", sts);  

  if(AllCalcs)  
    printf("This user has access to all script on %s %s\n", AppName, DbName);  
  else  
    {  
      if(!sts && pCalcList)  
      {  
        printf("-------- Get Calc List -----------\rn");  
        for (ind = 0; ind < Count; ind ++)  
          printf(" %s\n",pCalcList[ind]);  

          EssFree(hInst, pCalcList);  
      }  
    }  
  
}

ESS_FUNC_M ESS_SetCalcListEx (ESS_HCTX_T hCtx)  

{  
  ESS_STS_T sts = ESS_STS_NOERR;  
  ESS_STR_T calcUser;  
  ESS_BOOL_T AllCalcs;  
  ESS_USHORT_T count;  
  ESS_OBJNAME_T CalcList[2];  
  ESS_USERINFO_T pUserInfo;  
  ESS_BOOL_T bIsIdentity;  
  ESS_USHORT_T type;  
  ESS_USERDBEX_T userDb[1];  
  ESS_PUSERDBEX_T pUserDb = ESS_NULL;  

  bIsIdentity = ESS_FALSE;  
  type = ESS_TYPE_USER;  
  AllCalcs = ESS_FALSE;  
  count = 2;  
  strcpy(CalcList[0],"calc1");  
  strcpy(CalcList[1],"calc2");  

  sts = EssSetCalcListEx(hCtx, calcUser, bIsIdentity, type, AppName, DbName, AllCalcs, 
  count, CalcList);  
  printf("EssSetCalcListEx sts: %ld\n", sts);  
  if(!sts)  
    GetCalcList(calcUser);  

  return (sts);  
}
EssSetDatabaseAccess

Sets a list of user database access structures, which contain information about user access to databases.

Syntax

```c
ESS_FUNC_M EssSetDatabaseAccess (hCtx, Count, pUserDb);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>Count</td>
<td>ESS_USHORT_T</td>
<td>Count of user database structures.</td>
</tr>
<tr>
<td>pUserDb</td>
<td>“ESS_USERDB_T, ESS_GROUPDB_T” on page 190</td>
<td>Pointer to an array of user database structures.</td>
</tr>
</tbody>
</table>

Notes

The Access field of the user database structure is used to set the user’s granted access to the database. For this call the MaxAccess and FilterName fields are ignored.

Return Value

None.

Access

This function requires the caller to have database Design privilege (ESS_PRIV_DBDESIGN) for the specified database.

Example

```c
ESS_FUNC_M
ESS_SetDatabaseAccess (ESS_HCTX_T hCtx) {
    ESS_FUNC_M st = ESS_STS_NOERR;
    ESS_USHORT_T Count;
    ESS_USERDB_T UserDb[2];

    Count = 2;
    /* Initialize user database structure for user1 */
    strcpy(UserDb[0].UserName,"Newuser");
    strcpy(UserDb[0].AppName,"Sample");
    strcpy(UserDb[0].DbName,"Basic");
    UserDb[0].Access = ESS_PRIV_WRITE;

    /* Initialize user database structure for user2 */
    strcpy(UserDb[1].UserName,"Newuser2");
    strcpy(UserDb[1].AppName,"Sample");
}
```
strcpy(UserDb[1].DbName,"Basic");
UserDb[1].Access = ESS_PRIV_READ;
sts = EssSetDatabaseAccess(hCtx, Count, UserDb);
return (sts);
}

See Also
- EssSetDatabaseAccessEx
- EssGetDatabaseAccess
- EssListUsers
- EssSetApplicationAccess
- EssSetUser

EssSetDatabaseAccessEx

Sets a list of user database access structures, which contain information about user access to databases. Similar to EssSetDatabaseAccess, but the input structure can include user directories and unique identity attributes.

Syntax

```c
ESS_FUNC_M EssSetDatabaseAccessEx (hCtx, Count, pUserDb);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle (input).</td>
</tr>
<tr>
<td>Count</td>
<td>ESS_USHORT_T</td>
<td>Count of user database structures (input).</td>
</tr>
<tr>
<td>pUserDb</td>
<td>ESS_PUSERDBEX_T</td>
<td>Pointer to an array of user database structures (input).</td>
</tr>
</tbody>
</table>

Notes

The Access field of the user database structure is used to set the user’s granted access to the database. For this call the MaxAccess and FilterName fields are ignored.

Return Value

None.

Access

This function requires the caller to have database Design privilege (ESS_PRIV_DBDESIGN) for the specified database.

Example

```c
void DisplayUserDbInfo(ESS_PUSERDBEX_T userDb, ESS_USHORT_T count) {
    ESS_STS_T sts = ESS_STS_NOERR;
    ESS_USHORT_T ind;

    printf ("\n-------Database Access List----\n\n");
```
for (ind = 0; ind < count; ind++)
{
    printf("\tUser: %s\n", userDb[ind].UserName);
    printf("\tProvider Name: %s\n", userDb[ind].ProviderName);
    printf("\tConnection Param: %s\n", userDb[ind].connparam);
    printf("\tApp Name: %s\n", userDb[ind].AppName);
    printf("\tDb Name: %s\n", userDb[ind].DbName);
    switch(userDb[ind].Access)
    {
        case ESS_PRIV_NONE:
            printf("\tAccess: %d - ESS_PRIV_NONE\n", userDb[ind].Access);
            break;
        case ESS_PRIV_READ:
            printf("\tAccess: %d - ESS_PRIV_READ\n", userDb[ind].Access);
            break;
        case ESS_PRIV_WRITE:
            printf("\tAccess: %d - ESS_PRIV_WRITE\n", userDb[ind].Access);
            break;
        case ESS_PRIV_CALC:
            printf("\tAccess: %d - ESS_PRIV_CALC\n", userDb[ind].Access);
            break;
        case ESS_PRIV_METAREAD:
            printf("\tAccess: %d - ESS_PRIV_METAREAD\n", userDb[ind].Access);
            break;
        case ESS_PRIV_DBCREATE:
            printf("\tAccess: %d - ESS_PRIV_DBCREATE\n", userDb[ind].Access);
            break;
        case ESS_PRIV_APPLOAD:
            printf("\tAccess: %d - ESS_PRIV_APPLOAD\n", userDb[ind].Access);
            break;
        case ESS_PRIV_APPMANAGE:
            printf("\tAccess: %d - ESS_PRIV_APPMANAGE\n", userDb[ind].Access);
            break;
        case ESS_PRIV_APPCREATE:
            printf("\tAccess: %d - ESS_PRIV_APPCREATE\n", userDb[ind].Access);
            break;
        case ESS_PRIV_USERCREATE:
            printf("\tAccess: %d - ESS_PRIV_USERCREATE\n", userDb[ind].Access);
            break;
        case ESS_ACCESS_READ:
            printf("\tAccess: %d - ESS_ACCESS_READ\n", userDb[ind].Access);
            break;
        case ESS_ACCESS_WRITE:
            printf("\tAccess: %d - ESS_ACCESS_WRITE\n", userDb[ind].Access);
            break;
        case ESS_ACCESS_CALC:
            printf("\tAccess: %d - ESS_ACCESS_CALC\n", userDb[ind].Access);
            break;
        case ESS_ACCESS_METAREAD:
            printf("\tAccess: %d - ESS_ACCESS_METAREAD\n", userDb[ind].Access);
            break;
    }
}
break;
case ESS_ACCESS_DBMANAGE:
    printf("\tAccess: %d - ESS_ACCESS_DBMANAGE\n", userDb[ind].Access);
    break;
case ESS_ACCESS_DBCREATE:
    printf("\tAccess: %d - ESS_ACCESS_DBCREATE\n", userDb[ind].Access);
    break;
case ESS_ACCESS_APPMANAGE:
    printf("\tAccess: %d - ESS_ACCESS_APPMANAGE\n", userDb[ind].Access);
    break;
case ESS_ACCESS_APPCREATE:
    printf("\tAccess: %d - ESS_ACCESS_APPCREATE\n", userDb[ind].Access);
    break;
case ESS_ACCESS_FILTER:
    printf("\tAccess: %d - ESS_ACCESS_FILTER\n", userDb[ind].Access);
    break;
case ESS_ACCESS_DBALL:
    printf("\tAccess: %d - ESS_ACCESS_DBALL\n", userDb[ind].Access);
    break;
case ESS_ACCESS_APPALL:
    printf("\tAccess: %d - ESS_ACCESS_APPALL\n", userDb[ind].Access);
    break;
case ESS_ACCESS_ADMIN:
    printf("\tAccess: %d - ESS_ACCESS_ADMIN\n", userDb[ind].Access);
    break;
default:
    printf("\tAccess: Unknown\n");
}

switch(userDb[ind].MaxAccess)
{
    case ESS_PRIV_NONE:
        printf("\tMax Access: %d - ESS_PRIV_NONE\n", userDb[ind].MaxAccess);
        break;
case ESS_PRIV_READ:
        printf("\tMax Access: %d - ESS_PRIV_READ\n", userDb[ind].MaxAccess);
        break;
case ESS_PRIV_WRITE:
        printf("\tMax Access: %d - ESS_PRIV_WRITE\n", userDb[ind].MaxAccess);
        break;
case ESS_PRIV_CALC:
        printf("\tMax Access: %d - ESS_PRIV_CALC\n", userDb[ind].MaxAccess);
        break;
case ESS_PRIV_METAREAD:
        printf("\tMax Access: %d - ESS_PRIV_METAREAD\n", userDb[ind].MaxAccess);
        break;
case ESS_PRIV_DBLOAD:
        printf("\tMax Access: %d - ESS_PRIV_DBLOAD\n", userDb[ind].MaxAccess);
        break;
case ESS_PRIV_DBMANAGE:
        printf("\tMax Access: %d - ESS_PRIV_DBMANAGE\n", userDb[ind].MaxAccess);
        break;
case ESS_PRIV_DBCREATE:
        printf("\tMax Access: %d - ESS_PRIV_DBCREATE\n", userDb[ind].MaxAccess);
        break;
case ESS_PRIV_APPLOAD:
        printf("\tMax Access: %d - ESS_PRIV_APPLOAD\n", userDb[ind].MaxAccess);
        break;
break;
case ESS_PRIV_APPMANAGE:
    printf("\tMax Access: %d - ESS_PRIV_APPMANAGE\n", userDb[ind].MaxAccess);
    break;
case ESS_PRIV_APPCREATE:
    printf("\tMax Access: %d - ESS_PRIV_APPCREATE\n", userDb[ind].MaxAccess);
    break;
case ESS_PRIV_USERCREATE:
    printf("\tMax Access: %d - ESS_PRIV_USERCREATE\n", userDb[ind].MaxAccess);
    break;
case ESS_ACCESS_READ:
    printf("\tMax Access: %d - ESS_ACCESS_READ\n", userDb[ind].MaxAccess);
    break;
case ESS_ACCESS_WRITE:
    printf("\tMax Access: %d - ESS_ACCESS_WRITE\n", userDb[ind].MaxAccess);
    break;
case ESS_ACCESS_CALC:
    printf("\tMax Access: %d - ESS_ACCESS_CALC\n", userDb[ind].MaxAccess);
    break;
case ESS_ACCESS_METAREAD:
    printf("\tMax Access: %d - ESS_ACCESS_METAREAD\n", userDb[ind].MaxAccess);
    break;
case ESS_ACCESS_DBMANAGE:
    printf("\tMax Access: %d - ESS_ACCESS_DBMANAGE\n", userDb[ind].MaxAccess);
    break;
case ESS_ACCESS_DBCREATE:
    printf("\tMax Access: %d - ESS_ACCESS_DBCREATE\n", userDb[ind].MaxAccess);
    break;
case ESS_ACCESS_APPMANAGE:
    printf("\tMax Access: %d - ESS_ACCESS_APPMANAGE\n", userDb[ind].MaxAccess);
    break;
case ESS_ACCESS_APPCREATE:
    printf("\tMax Access: %d - ESS_ACCESS_APPCREATE\n", userDb[ind].MaxAccess);
    break;
case ESS_ACCESS_FILTER:
    printf("\tMax Access: %d - ESS_ACCESS_FILTER\n", userDb[ind].MaxAccess);
    break;
case ESS_ACCESS_DBALL:
    printf("\tMax Access: %d - ESS_ACCESS_DBALL\n", userDb[ind].MaxAccess);
    break;
case ESS_ACCESS_APPALL:
    printf("\tMax Access: %d - ESS_ACCESS_APPALL\n", userDb[ind].MaxAccess);
    break;
case ESS_ACCESS_ADMIN:
    printf("\tMax Access: %d - ESS_ACCESS_ADMIN\n", userDb[ind].MaxAccess);
    break;
default:
    printf("\tMax Access: Unknown\n");
}

printf("\tFilter Name: %s\n", userDb[ind].FilterName);
printf("\n");
}

ESS_FUNC_M ESS_SetDatabaseAccessEx (ESS_HCTX_T hCtx, ESS_HINST_T hInst)
{  
    ESS_STS_T sts = ESS_STS_NOERR;
    ESS_STR_T userId;
    ESS_BOOL_T bIsIdentity;
    ESS_USHORT_T type;
    ESS_USHORT_T count;
    ESS_USERDBEX_T userDb[2];
    ESS_PUSERDBEX_T pUserDb = ESS_NULL;

    memset(&userDb, '\0', sizeof(userDb));

    count = 1;
    strcpy(userDb[0].UserName, "IDUser1");
    strcpy(userDb[0].ProviderName, "");
    strcpy(userDb[0].connparam, "");
    userDb[0].type = ESS_TYPE_USER;
    strcpy(userDb[0].AppName, AppName);
    strcpy(userDb[0].DbName, DbName);
    userDb[0].Access = ESS_PRIV_READ;
    userDb[0].MaxAccess = ESS_PRIV_READ;
    sts = EssSetDatabaseAccessEx(hCtx, count, &userDb);
    printf("EssSetDatabaseAccessEx sts: %ld\n\n", sts);
    if(!sts)
    {
        sts = EssGetDatabaseAccessEx(hCtx, userId, bIsIdentity, type, AppName, DbName,
                                &count, &pUserDb);
        printf("EssGetDatabaseAccessEx sts: %ld\n\n", sts);
        if(!sts)
        {
            if(count && pUserDb)
            {
                DisplayUserDbInfo(pUserDb, count);
                sts = EssFree (hInst, pUserDb);
            }
            else
                printf ("\rUser Application list is empty\n\n");
        }
    }

    return (sts);
}

See Also

- EssGetDatabaseAccessEx
- EssListUsersInfoEx
- EssSetApplicationAccessEx
EssSetDatabaseNote

Sets a database's note-of-the-day message. This message may be used to display useful information about the database (whether data has been loaded, when it was last calculated, etc.) to users before they connect to the database.

Syntax

```c
ESS_FUNC_M EssSetDatabaseNote (hCtx, AppName, DbName, DbNote);
```

Parameter  Data Type  Description

- **hCtx**: ESS_HCTX_T  API context handle
- **AppName**: ESS_STR_T  Application name
- **DbName**: ESS_STR_T  Database name
- **DbNote**: ESS_STR_T  Pointer to database note string

Notes

The database note string must be less than 64 KB in length.

Return Value

None.

Access

This function requires the caller to have database Design privilege (ESS_PRIV_DBDESIGN) for the specified database.

Example

```c
ESS_FUNC_M

ESS_SetDatabaseNote (ESS_HCTX_T hCtx)
{
    ESS_FUNC_M st = ESS_STS_NOERR;
    ESS_STR_T AppName;
    ESS_STR_T DbName;
    ESS_STR_T DbNote;

    AppName = "Sample";
    DbName  = "Basic";
    DbNote  = "This is a test";

    st = EssSetDatabaseNote(hCtx, AppName, DbName,
                            DbNote);

    return (st);
}
```

See Also

- EssGetDatabaseNote
**EssSetDatabaseState**

Sets user-configurable parameters for the database using the database’s state structure.

**Syntax**

```
ESS_FUNC_M EssSetDatabaseState (hCtx, AppName, DbName, pDbState);
```

**Parameter** | **Data Type** | **Description**
--- | --- | ---
 hCtx | ESS_HCTX_T | API context handle
AppName | ESS_STR_T | Application name
DbName | ESS_STR_T | Database name
pDbState | “ESS_DBSTATE_T” on page 127 | Pointer to database state structure

**Notes**

- EssGetDatabaseState() should be called to initialize the ESS_DBSTATE_T structure before EssSetDatabaseState() is called.
- This function can only set user-configurable parameters for server databases.

**Return Value**

None.

**Access**

This function requires the caller to have database designer privilege (ESS_PRIV_DBDESIGN) for the specified database.

**Example**

```
ESS_FUNC_M
ESS_SetDbState (ESS_HCTX_T hCtx,
               ESS_HINST_T hInst
        )
{
  ESS_FUNC_M     sts = ESS_STS_NOERR;
  ESS_PDBSTATE_T DbState;
  ESS_STR_T      AppName;
  ESS_STR_T      DbName;
  AppName = "Sample";
  DbName = "Basic";

  sts = EssGetDatabaseState (hCtx, AppName,
                             DbName, &DbState);
  if (!sts)
  { 
    if (DbState)
    {
      /**********************************************
      * Update DbState structure *
      **********************************************/
      sts = EssSetDatabaseState (hCtx, AppName,
```

``````
DbName, DbState);
        EssFree (hInst, DbState);
    }
}
return (sts);
}

See Also

- EssGetDatabaseState
- EssSetApplicationState

EssSetDefaultCalc

Sets the default calc script for the active database.

Syntax

ESS_FUNC_M EssSetDefaultCalc (hCtx, CalcScript);

Parameter | Data Type | Description
---|---|---
hCtx | ESS_HCTX_T | API context handle.
CalcScript | ESS_STR_T | Default calc script string.

Notes

The calc script string must not be greater than 64 KB long.

Return Value

None.

Access

This function requires the caller to have calc privilege (ESS_PRIV_CALC) to the active database.

Example

ESS_FUNC_M
Ess_SetDefaultCalc (ESS_HCTX_T   hCtx)
{
    ESS_FUNC_M           sts = ESS_STS_NOERR;
    sts = EssSetDefaultCalc (hCtx, "CALC ALL;");

    return (sts);
}

See Also

- EssDefaultCalc
- EssGetDefaultCalc
- EssSetActive
- EssSetDefaultCalcFile
**EssSetDefaultCalcFile**

Sets the default calc script for the active database from a calc script file.

**Syntax**

```c
ESS_FUNC_M EssSetDefaultCalcFile (hDestCtx, hSrcCtx, AppName, DbName, FileName);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hDestCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle of target database on the server.</td>
</tr>
<tr>
<td>hSrcCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle for calc script file location. The calc script file can reside on the client or on the same server as the target database.</td>
</tr>
<tr>
<td>AppName</td>
<td>ESS_STR_T</td>
<td>Application name for calc script file location</td>
</tr>
<tr>
<td>DbName</td>
<td>ESS_STR_T</td>
<td>Database name for calc script file location</td>
</tr>
<tr>
<td>FileName</td>
<td>ESS_STR_T</td>
<td>Name of default calc script file</td>
</tr>
</tbody>
</table>

**Notes**

- The default calc script must not be greater than 64 KB long.
- The server makes a copy of the text in the calc script file when this function is called. Subsequent changes to the calc script file have no effect on the default calc unless this function is called again to update it.

**Return Value**

None.

**Access**

This function requires the caller to have calc privilege (ESS_PRIV_CALC) to the active database.

**Example**

```c
ESS_FUNC_M
ESS_SetDefaultCalcFile (ESS_HCTX_T hCtx)
{
    ESS_FUNC_M    sts = ESS_STS_NOERR;
    ESS_HCTX_T    hSrcCtx;
    ESS_STR_T     AppName;
    ESS_STR_T     DbName;
    ESS_STR_T     FileName;
    AppName  = "Sample";
    DbName   = "Basic";
    FileName = "DefTest";
    hSrcCtx = hCtx;
    sts = EssSetDefaultCalcFile (hCtx, hSrcCtx,
                              AppName, DbName, FileName);
    return(sts);
}
```

**See Also**

- **EssDefaultCalc**
EssSetEasLocation

Set or change the Essbase Administration Server location that will be registered with Shared Services upon application creation or migration.

Syntax

ESS_FUNC_M EssSetEasLocation (hCtx, EasLocation);

Parameter | Data Type | Description
--- | --- | ---
hCtx | ESS_HCTX_T | API context handle.
EasLocation | ESS_STR_T | The name (or IP address) and port number of the computer on which Essbase Administration Server runs. Examples:

Aspen:10080
127.0.0.1:10080

Notes

After changing the Essbase Administration Server location, you must use EssReRegisterApplication to re-register any existing applications with Shared Services.

Return Value

Returns 0 if successful; otherwise, returns an error.

Access

This function requires the caller to be an Administrator.

Example

ESS_FUNC_M ESS_SS_SetEasLocation(ESS_HCTX_T hCtx, ESS_HINST_T hInst)
{
    ESS_STS_T     sts = ESS_STS_NOERR;
    ESS_STR_T     easLoc = ESS_NULL;

    /* Eas Location */
    sts = EssAlloc(hInst, sizeof(ESS_PATHLEN), &easLoc);
    if(sts)
        return (sts);
    memset(easLoc, 0, sizeof(ESS_PATHLEN));
    strcpy( easLoc, "localhost:10080");
    strcpy( easLoc, "localhost:10080");

    if (EasLocation)
        printf("Failed to set EAS Location.\n");
    if (easLoc)
        EssFree(hInst, easLoc);

}
return (sts);
}

See also an extended Appendix B

See Also

- EssReRegisterApplication

**EssSetExtUser**

Sets user information for externally authenticated users.

**Syntax**

```c
ESS_FUNC_M EssSetExtUser (hCtx, type, UserName, Password, Protocol, ConnParam);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>type</td>
<td>ESS_USHORT_T</td>
<td>Type of user (external).</td>
</tr>
<tr>
<td>UserName</td>
<td>ESS_STR_T</td>
<td>Name of user.</td>
</tr>
<tr>
<td>Password</td>
<td>ESS_STR_T</td>
<td>User’s password.</td>
</tr>
<tr>
<td>Protocol</td>
<td>ESS_STR_T</td>
<td>External authentication protocol: CSS, for Shared Services mode.</td>
</tr>
<tr>
<td>ConnParam</td>
<td>ESS_STR_T</td>
<td>External authentication connection parameters. Null, for CSS protocol.</td>
</tr>
</tbody>
</table>

**Access**

This function requires the caller to have Create/Delete User privilege (ESS_PRIV_USERCREATE) for the logged in server, unless they are setting their own user information.

**EssSetFilter**

Creates or replaces a filter, and starts setting the contents of the filter.

**Syntax**

```c
ESS_FUNC_M EssSetFilter (hCtx, AppName, DbName, FilterName, Active, Access);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle</td>
</tr>
<tr>
<td>AppName</td>
<td>ESS_STR_T</td>
<td>Application name</td>
</tr>
<tr>
<td>DbName</td>
<td>ESS_STR_T</td>
<td>Database name</td>
</tr>
<tr>
<td>FilterName</td>
<td>ESS_STR_T</td>
<td>Filter name. See “Filter Name Limits” on page 1728.</td>
</tr>
</tbody>
</table>

See Also

- EssReRegisterApplication
### Parameter Data Type Description

- **Active**: ESS_BOOL_T  Filter active flag. If TRUE, the filter is set active, otherwise it is set inactive.
- **Access**: ESS_ACCESS_T  The default filter access level

### Notes
- If the filter does not already exist, it will first be created by this call.
- This call must be followed by successive calls to `EssSetFilterRow()` to set all the rows for the filter.
- To avoid overwriting a filter that already exists, use `EssCreateFilter`. `EssCreateFilter` creates only a uniquely named filter for a particular database, but will not overwrite an existing filter of the same name on the same database.

### Return Value
None.

### Access
This function requires the caller to have database Design privilege (ESS_PRIV_DBDESIGN) for the specified database.

### Example

```c
ESS_FUNC_M
ESS_SetFilter (ESS_HCTX_T  hCtx)
{
    ESS_FUNC_M       sts = ESS_STS_NOERR;
    ESS_STR_T       AppName;
    ESS_STR_T       DbName;
    ESS_STR_T       FilterName;
    ESS_BOOL_T      Active;
    ESS_ACCESS_T    Access, AccessAry[3];
    ESS_STR_T       RowString[3];
    ESS_USHORT_T    ind;

    AppName    = "Sample";
    DbName     = "Basic";
    FilterName = "NewFilter";
    Active     = ESS_TRUE;

    /**** Set Filter *****/
    sts = EssSetFilter(hCtx, AppName, DbName, FilterName, Active, Access);
    if(!sts)
    {
        RowString[0] = "@IDESCENDANTS(Scenario)";
        RowString[1] = "@IDESCENDANTS(Product)";
        RowString[2] = "Qtr1, @IDESCENDANTS("Colas")";

        AccessAry[0] = ESS_ACCESS_READ;
        AccessAry[1] = ESS_ACCESS_NONE;
        /**** Set Filter Rows *****/
```
for(ind = 0; ind < 3; ind++)
{
    sts = EssSetFilterRow(hCtx, RowString[ind],
                        AccessAry[ind]);
    if(sts)
        printf("Cannot set Filter row %s\r\n",
               RowString[ind]);
    sts = EssSetFilterRow(hCtx,
                        ",", ESS_ACCESS_NONE);
    return (sts);
}

See Also
● EssCreateFilter
● EssGetFilter
● EssListFilters
● EssSetFilterRow

EssSetFilterList

Sets the list of groups or users that are assigned to a filter. The count parameter controls the number of groups or users assigned to the filter. A count of zero will remove all the groups or users from the list.

Syntax

ESS_FUNC_M EssSetFilterList (hCtx, AppName, DbName, FilterName, Count, pUserList);

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle</td>
</tr>
<tr>
<td>AppName</td>
<td>ESS_STR_T</td>
<td>Application name</td>
</tr>
<tr>
<td>DbName</td>
<td>ESS_STR_T</td>
<td>Database name</td>
</tr>
<tr>
<td>FilterName</td>
<td>ESS_STR_T</td>
<td>Filter name</td>
</tr>
<tr>
<td>Count</td>
<td>ESS_USHORT_T</td>
<td>Count of groups or users assigned this filter</td>
</tr>
<tr>
<td>pUserList</td>
<td>ESS_PUSERNAME_T</td>
<td>Pointer to array of user names</td>
</tr>
</tbody>
</table>

Return Value

None.

Access

This function requires the caller to have database Design privilege (ESS_PRIV_DBDESIGN) for the specified database.
Example

```c
ESS_FUNC_M
ESS_SetFilterList (ESS_HCTX_T hCtx)
{
    ESS_FUNC_M sts = ESS_STS_NOERR;
    ESS_STR_T AppName;
    ESS_STR_T DbName;
    ESS_STR_T FilterName;
    ESS_USHORT_T Count = 0;
    ESS_USERNAME_T UserList[2];

    AppName    = "Sample";
    DbName     = "Basic";
    FilterName = "Test";
    strcpy(UserList[0], "Jim Smith");
    strcpy(UserList[1], "Newuser");
    Count = 2;

    sts = EssSetFilterList(hCtx, AppName, DbName,
                            FilterName, Count, UserList);
    return (sts);
}
```

See Also

- `EssSetFilterListEx`
- `EssGetFilterList`
- `EssListFilters`
- `EssSetFilter`

### EssSetFilterListEx

Sets the list of groups or users that are assigned to a filter. The count parameter controls the number of groups or users assigned to the filter. A count of zero will remove all the groups or users from the list.

Similar to `EssSetFilterList`, but includes users and groups hosted in a user directory.

#### Syntax

```c
ESS_FUNC_M EssSetFilterListEx (hCtx, AppName, DbName, FilterName, bIsIdentity,
                                entityType, Count, UserList);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle (input).</td>
</tr>
<tr>
<td>AppName</td>
<td>ESS_STR_T</td>
<td>Application name (input).</td>
</tr>
<tr>
<td>DbName</td>
<td>ESS_STR_T</td>
<td>Database name (input).</td>
</tr>
<tr>
<td>FilterName</td>
<td>ESS_STR_T</td>
<td>Filter name (input).</td>
</tr>
</tbody>
</table>
Parameter | Data Type | Description
---|---|---
'bIsIdentity' | ESS_BOOL_T | Input. Indicates whether userList contains names or identities. If TRUE, userList contains identities.

The list can contain names or identities, but not both.

'entityType' | ESS_USHORT_T | Type of entity contained in userList (input). Can be one of the following:

* ESS_TYPE_USER
* ESS_TYPE_GROUP

The list can contain users or groups, but not both.

'Count' | ESS_USHORT_T | Count of entities contained in userList (input).

'UserList' | ESS_PSTR_T | Array of user or group names or identities (input).

**Return Value**

None.

**Access**

This function requires the caller to have database Design privilege (ESS_PRIV_DBDESIGN) for the specified database.

**Example**

```c
void GetFilterList()
{
    ESS_STS_T sts = ESS_STS_NOERR;
    ESS_STR_T FilterName;
    ESS_USHORT_T Count = 0;
    ESS_USHORT_T ind;
    ESS_PUSERNAME_T UserList = NULL;

    FilterName = "Filter1";

    sts = EssGetFilterList(hCtx, AppName, DbName, FilterName, &Count, &UserList);
    printf("EssGetFilterList sts: %ld\n", sts);
    if(!sts)
    {
        printf("--------%s User List--------\n", FilterName);
        if(Count & UserList)
        {
            for (ind = 0; ind < Count; ind++)
            {
                printf("%s\n",UserList[ind]);
                EssFree(hInst, UserList[ind]);
            }
        }
        else
        {
            printf("none.\n");
        }
    }
}
```
ESS_FUNC_M ESS_SetFilterListEx (ESS_HCTX_T hCtx)
{
    ESS_STS_T sts = ESS_STS_NOERR;
    ESS_STR_T FilterName;
    ESS_USHORT_T Count = 0;
    //ESS_USERNAME_T UserList[2];
    ESS_STR_T UserList[2];
    ESS_BOOL_T bIsIdentity;
    ESS_USHORT_T type;

    FilterName = "Filter1";
    UserList[0] = "IDUser9@ldap";
    UserList[1] = "IDUser10@ldap";
    Count = 2;
    bIsIdentity = ESS_TRUE;
    type = ESS_TYPE_USER;

    sts = EssSetFilterListEx(hCtx, AppName, DbName, FilterName, bIsIdentity, type, Count,
UserList);
    printf("EssSetFilterListEx sts: %ld\n", sts);
    if(!sts)
        GetFilterList();

    return (sts);
}

See Also

● ESSGetFilterList
● EssListFilters
● EssSetFilter

EssSetFilterRow

Sets the next row of a filter.

Syntax

ESS_FUNC_M EssSetFilterRow (hCtx, RowString, Access);

Parameter Data Type Description

hCtx ESS_HCTX_T API context handle.
RowString ESS_STR_T Pointer to the next row of the filter.
Access ESS_ACCESS_T Access level for the filter row.

Notes

This function should be called repeatedly after calling EssSetFilter(), once for each row of the
filter, terminating the row list with a NULL row string pointer.
**Return Value**

None.

**Access**

This function requires the caller to have database designer privilege (ESS_PRIV_DBDESIGN) for the specified database.

**Example**

See the example of `EssSetFilter`.

**See Also**

- `EssListFilters`
- `EssSetFilter`

---

### EssSetGlobalState

Sets the server global state structure which contains parameters for system administration.

**Syntax**

```
ESS_FUNC_M EssSetGlobalState (hCtx, pGlobal);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>pGlobal</td>
<td>“ESS_GLOBAL_T” on page 140</td>
<td>Pointer to global state structure.</td>
</tr>
</tbody>
</table>

**Notes**

When changing parameter values, it is advisable to call `EssGetGlobalState()` first to get the correct values of any parameters you do not wish to change.

**Return Value**

None.

**Access**

This function requires the caller to be an administrator.

**Example**

```c
ESS_FUNC_M
ESS_SetGlobalState (ESS_HCTX_T hCtx) 
{
    ESS_FUNC_M       sts = ESS_STS_NOERR;
    ESS_GLOBAL_T     Global;

    /* Initialize Global State */
    Global.Security = 1;
    Global.Logins = 1;
    Global.Access = ESS_ACCESS_NONE;
```
Global.Validity = 200;
Global.Currency = 1;
Global.PwMin = 8;
Global.InactivityTime = 3600;
Global.InactivityCheck = 300;

sts = EssSetGlobalState(hCtx, &Global);
return (sts);
}

See Also
● EssGetGlobalState

**EssSetGroup**

Sets a group information structure, which contains security information for the group.

**Syntax**

```c
ESS_FUNC_M EssSetGroup (hCtx, pGroupInfo);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>pGroupInfo</td>
<td>&quot;ESS_USERINFO_T, ESS_GROUPINFO_T&quot; on page 192</td>
<td>Pointer to group info structure.</td>
</tr>
</tbody>
</table>

**Notes**

- The name of the group to set is a field in the group info structure, which must always be specified.
- The only field in the group info structure which may be changed using this function is the Access field (the other fields are used for users of for information only). See the description of the ESS_GROUPINFO_T structure for more information.

**Return Value**

None.

**Access**

This function requires the caller to have Create/Delete User privilege (ESS_PRIV_USERCREATE) for the logged in server.

**Example**

```c
ESS_FUNC_M
ESS_SetGroup (ESS_HCTX_T hCtx)
{
    ESS_FUNC_M sts = ESS_STS_NOERR;
    ESS_USERINFO_T Group;

    strcpy(Group.Name,"PowerUsers");
    strcpy(Group.AppName,"Sample");
    strcpy(Group.DbName,"Basic");
```
Group.Access = ESS_ACCESS_SUPER;

sts = EssSetGroup(hCtx, &Group);
return (sts);
}

See Also
● EssGetGroup
● EssListGroups

EssSetGroupList

Sets the list of users who are members of a group.

Syntax

ESS_FUNC_M EssSetGroupList (hCtx, GroupName, Count, pUserList);

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>GroupName</td>
<td>ESS_STR_T</td>
<td>Group name or user name.</td>
</tr>
<tr>
<td>Count</td>
<td>ESS_USHORT_T</td>
<td>Count of user names.</td>
</tr>
<tr>
<td>pUserList</td>
<td>ESS_PUSERNAME_T</td>
<td>Pointer to an array of user name strings.</td>
</tr>
</tbody>
</table>

Notes

This function can also be used to set the list of groups to which a user belongs by using a user name as the GroupName argument and passing a list of groups as the pUserList argument.

An administrator that is not an Essbase administrator, in order to administer privileges on users and groups, must have higher privileges than those users and groups on the applications on which they are administering access. To bypass this restriction, use MaxL for adding users to groups.

Return Value

None.

Access

This function requires the caller to have Create/Delete User privilege (ESS_PRIV_USERCREATE) for the logged in server.

Example

ESS_FUNC_M
EssSetGroupList (ESS_HCTX_T hCtx)
{
    ESS_FUNC_M        sts = ESS_STS_NOERR;
    ESS_USERNAME_T   UserList[3];
    ESS_USHORT_T     Count;
    ESS_STR_T        GroupName;
GroupName = "Powerusers";
strcpy(UserList[0],"App Designer");
strcpy(UserList[1],"Db Designer");
strcpy(UserList[2],"User Creator");
Count = 3;

sts = EssSetGroupList (hCtx, GroupName, Count, UserList);
return (sts);
}

See Also
- EssSetGroupListEx
- EssAddToGroup
- EssDeleteFromGroup
- EssListGroups

### EssSetGroupListEx

Sets the list of users who are members of a group. Similar to EssSetGroupList, but can accept a user directory specification or unique identity attribute for EntityId.

**Syntax**

```
ESS_FUNC_M EssSetGroupListEx (hCtx, EntityId, bIsIdentity, entityType, Count, pIdList, bUsingIdentity);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle (input).</td>
</tr>
<tr>
<td>EntityId</td>
<td>ESS_STR_T</td>
<td>User or group name (input). Can be specified as name@provider or as a unique identity attribute.</td>
</tr>
<tr>
<td>bIsIdentity</td>
<td>ESS_BOOL_T</td>
<td>Input. Indicates if EntityId is a name or an identity. If TRUE, EntityId is an identity.</td>
</tr>
<tr>
<td>entityType</td>
<td>ESS_USHORT_T</td>
<td>Input. Indicates if EntityId is a group or a user. If specified as user, then the list can consist of only groups. If specified as group, then the list can contain only groups in native security mode. but in EPM System security mode, the list can consist of both users and groups.</td>
</tr>
<tr>
<td>Count</td>
<td>ESS_USHORT_T</td>
<td>Count of identities (input).</td>
</tr>
<tr>
<td>pIdList</td>
<td>ESS_PSTR_T</td>
<td>Pointer to an array of identities (input).</td>
</tr>
<tr>
<td>bUsingIdentity</td>
<td>ESS_BOOL_T</td>
<td>Input. Indicates if EntityId is a name or an identity. If TRUE, EntityId is an identity.</td>
</tr>
</tbody>
</table>

**Notes**

This function can also be used to set the list of groups to which a user belongs by using a user name as the EntityID argument and passing a list of groups as the pUserIDList argument.

An administrator that is not an Essbase administrator, in order to administer privileges on users and groups, must have higher privileges than those users and groups on the applications on which they are administering access.
**Return Value**

None.

**Access**

This function requires the caller to have Create/Delete User privilege (ESS_PRIV_USERCREATE) for the logged in server.

**Example**

**See Also**

- EssAddToGroupEx
- EssDeleteFromGroupEx
- EssListGroupsInfoEx

**EssSetGroupToSS**

Migrates a group to EPM System security mode. This might be useful if the group migration failed using EssSetSSSecurityMode.

**Syntax**

```c
ESS_FUNC_M EssSetGroupToSS (hCtx, GroupName);
```

**Parameter** | **Data Type** | **Description**
---|---|---

|  |  |  
|---|---|---|
| hCtx | ESS_HCTX_T | API context handle. |
| GroupName | ESS_STR_T | Name of the group to convert to Shared Services (input). |

**Return Value**

Returns 0 if successful; otherwise, returns an error.

**Access**

This function requires the caller to be an Administrator.

**Example**

```c
ESS_FUNC_M ESS_SS_SetGroupToSS(ESS_HCTX_T hCtx, ESS_HINST_T hInst)
{
    ESS_STS_T         sts = ESS_STS_NOERR;
    ESS_STR_T         groupName = ESS_NULL;

    sts = EssAlloc(hInst, sizeof(ESS_USERNAME_T), &groupName);
    if(sts)
        return (sts);
    memset(groupName, 0, sizeof(ESS_USERNAME_T));
    strcpy( groupName, "essgrp");

    sts = EssSetGroupToSS(hCtx, groupName);
    if(sts)
```
printf("Failed to migrate Group %s to Shared Services mode.\n", groupName);

    if (groupName)
        EssFree(hInst, groupName);

    return (sts);
}

See also an extended Appendix B

**EssSetGroupsToSS**

Migrates all groups to EPM System security mode. This might be useful if the group migration failed using **EssSetSSSecurityMode**.

**Syntax**

```
ESS_FUNC_M EssSetGroupsToSS (hCtx);
```

**Parameter** | **Data Type** | **Description**
--- | --- | ---
 hCtx | ESS_HCTX_T | API context handle.

**Return Value**

Returns 0 if successful; otherwise, returns an error.

**Access**

This function requires the caller to be an Administrator.

**Example**

```
ESS_FUNC_M ESS_SS_SetGroupsToSS(ESS_HCTX_T  hCtx)
{  
    ESS_STS_T         sts = ESS_STS_NOERR;

    sts = EssSetGroupsToSS(hCtx);

    if(sts)
        printf("Failed to migrate Groups to Shared Services mode.\n");

    return (sts);
}
```

See also an extended Appendix B

**EssSetPassword**

Sets a user’s password, erasing the existing password.

**Syntax**

```
ESS_FUNC_M   EssSetPassword (hCtx, UserName, Password);
```
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>UserName</td>
<td>ESS_STR_T</td>
<td>User name.</td>
</tr>
<tr>
<td>Password</td>
<td>ESS_STR_T</td>
<td>New password for user.</td>
</tr>
</tbody>
</table>

**Notes**
- To change a password, the caller must either have supervisor access, or be changing their own password.
- The new password will take effect the next time the user logs in.

**Return Value**
None.

**Access**
This function requires callers to have Create/Delete User privilege (ESS_PRIV_USERCREATE) for the logged in server, unless they are setting their own password.

**Example**
```c
ESS_FUNC_M
ESS_SetPassword (ESS_HCTX_T  hCtx)
{
    ESS_FUNC_M     sts = ESS_STS_NOERR;
    ESS_STR_T     UserName;
    ESS_STR_T     Password;

    UserName = "Jim Smith";
    Password = "newpwd";
    sts = EssSetPassword (hCtx,UserName, Password);
    return (sts);
}
```

**See Also**
- EssListUsers
- EssSetUser

### EssSetPath

Sets the ESSBASEPATH environment variable for the runtime process.

**Syntax**
```c
ESS_FUNC_M EssSetPath (pszPath);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pszPath</td>
<td>ESS_STR_T</td>
<td>Pointer to the string describing the ESSBASEPATH environment variable</td>
</tr>
</tbody>
</table>
Notes

- Call *EssSetPath()* before calling *EssInit()*.
- *pszPath* cannot exceed 120 characters, as defined in ESS_PATHLEN.
- *pszPath* applies only to the current process.
- Essbase DLLs must be accessible from the system path. *EssSetPath()* does not resolve the path for the Essbase DLLs.

Return Value

- If successful, returns ESS_STS_NOERR.
- If *pszPath* is too long, returns API_NAME_TOO_LONG (1030009).

Example

```c
ESS_STS_T
ESS_SetPath()
{
    ESS_STS_T sts;
    ESS_STR_T pszPath = "C:\Hyperion\products\Essbase";
    sts = EssSetPath (pszPath);
    return sts;
}
```

**EssSetServerMode**

Sets the mode of Essbase Server to be Unicode or non-Unicode. Only when it is in Unicode mode does Essbase Server allow creation of Unicode mode applications or migration of non-Unicode mode applications to Unicode mode. Setting a Unicode mode server to non-Unicode does not affect the existing Unicode-related mode of existing applications.

Syntax

```c
ESS_FUNC_M EssSetServerMode(hCtx, bUnicode);
```

Parameter | Data Type   | Description
----------|-------------|-------------
hCtx      | ESS_HCTX_T  | API context handle (logged in)
bUnicode  | ESS_BOOL_T  | The pass-in parameter, *bUnicode*, where *bUnicode* can be one of the following values:
- ESS_TRUE—Sets the server mode to Unicode. Essbase Server allows creation of Unicode mode applications or migration of non-Unicode mode applications to Unicode mode.
- ESS_FALSE—Sets the server mode to non-Unicode. Essbase Server does not allow creation of Unicode mode applications or migration of non-Unicode mode applications to Unicode mode.

Return Value

None.
Access

This function requires the caller to have (AD_ACCESS_SUPER) privilege for the logged in server.

See Also

- EssGetServerMode

**EssSetSpanRelationalPartition**

Sets the Boolean bSpanRelPart field informing Essbase that pertinent data exists in an attached relational store. Some other API functions, such as EssQueryDatabaseMembers, read bSpanRelPart and access the relational store if bSpanRelPart is set.

**Syntax**

```c
ESS_FUNC_M EssSetSpanRelationalPartition (hCtx);
```

**Parameter Data Type Description**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
</tbody>
</table>

**Notes**

Several API functions have been enhanced to retrieve information from relational stores.

- **EssQueryDatabaseMembers** - returns member names from the relational store.
- **EssGetMemberInfo** - returns information on members in the relational store.
- **EssCheckMemberName** - checks in the relational store for valid member names.
- **EssGetMemberCalc** - recognizes a relational member passed as input and returns a null string for all relational members.

**Return Value**

None.

Access

This function requires the caller to have read privilege (ESS_PRIV_READ) to one or more members in the active database.

**Example**

```c
ESS_FUNC_M
ESS_Report (ESS_HCTX_T hCtx,
            ESS_HINST_T hInst)
{
    ESS_FUNC_M sts = ESS_STS_NOERR;
    ESS_STR_T rString = NULL;
    sts = EssBeginReport (hCtx, ESS_TRUE, ESS_FALSE);
    if (!sts)
        sts = EssSendString (hCtx, "<Desc Year !");
    if (!sts)
        return ESS_STS_NOERR;
}
```
ss = EssSetSpanRelationalPartition(hCtx);
/**************
* Get report *
**************/
if (!ss)
ss = EssGetString(hCtx, &rString);
while ((!ss) && (rString != NULL))
{
printf("%s", rString);
EssFree(hInst, rString);
ss = EssGetString(hCtx, &rString);
}
printf("
"
);
return(ss);
}

See Also
- EssClrSpanRelationalSource

**EssSetSSSecurityMode**

Migrates Essbase Server and any existing users and groups to EPM System security mode.

**Syntax**

ESS_FUNC_M EssSetSSSecurityMode (hCtx, option, NewPassword, FileName, flag);

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>option</td>
<td>ESS_USHORT_T</td>
<td>Integer representing the desired password creation method for migrated users.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- 0—Use an administrator-specified password.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- 1—Create passwords that are the same as user names for users being migrated to Shared Services.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Note: The passwords are created as lower-case, even if there are upper-case letters in the user name.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- 2—Automatically generate new passwords for the users being migrated to Shared Services.</td>
</tr>
<tr>
<td>NewPassword</td>
<td>ESS_STR_T</td>
<td>Password string (if option 2 is used).</td>
</tr>
<tr>
<td>FileName</td>
<td>ESS_STR_T</td>
<td>Name of file to contain saved passwords. If not provided, the default file is $ARBORPATH/bin/MigratedUsersPassword.txt.</td>
</tr>
<tr>
<td>flag</td>
<td>ESS_USHORT_T</td>
<td>Whether the passwords file, if already existing, should be overwritten.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_FILE_OVERWRITE—Overwrite</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_FILE_NOOVERWRITE—Do not overwrite; return an error.</td>
</tr>
</tbody>
</table>

636  C Main API Functions
**Return Value**

Returns 0 if successful; otherwise, returns an error.

**Access**

This function requires the caller to be an Administrator.

**Example**

```c
ESS_FUNC_M ESS_SS_SetSSSecurityMode(ESS_HCTX_T hCtx)
{
    ESS_STS_T         sts = ESS_STS_NOERR;
    ESS_STR_T         newPassword = ESS_NULL;
    ESS_USHORT_T      option;
    ESS_STR_T         fName = ESS_NULL;
    ESS_USHORT_T      flag = 0;

    /* New Shared Services Native User Password Option:  
     * 0 to use user provided password  
     * 1 to use the user name as password  
     * 2 to automatically generate a password  
    **/
    option = 1; /* Using user name as password */

    sts = EssSetSSSecurityMode(hCtx, option, newPassword, fName, flag);

    if(sts)
        printf("Failed to migrate Essbase Server to Shared Services mode.\n");

    return (sts);
}
```

See also an extended Appendix B

**See Also**

- EssSetUserToSS
- EssSetGroupToSS

**EssSetUserToSS**

Migrates a user to EPM System security mode. This might be useful if the user migration failed using EssSetSSSecurityMode.

**Syntax**

```c
ESS_FUNC_M EssSetUserToSS (hCtx, UserName, option, NewPassword, FileName, flag);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>UserName</td>
<td>ESS_STR_T</td>
<td>Name of the user to convert to Shared Services (input).</td>
</tr>
<tr>
<td>Parameter</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>option</td>
<td>ESS_USHORT_T</td>
<td>Integer representing the desired password creation method for migrated users.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[ ] 0—Use an administrator-specified password.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[ ] 1—Create passwords that are the same as user names for users being migrated to Shared Services.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note:</strong> The passwords are created as lower-case, even if there are upper-case letters in the user name.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[ ] 2—Automatically generate new passwords for the users being migrated to Shared Services.</td>
</tr>
<tr>
<td>NewPassword</td>
<td>ESS_STR_T</td>
<td>Password string (if option 2 is used).</td>
</tr>
<tr>
<td>FileName</td>
<td>ESS_STR_T</td>
<td>Name of file to contain saved passwords. If not provided, the default file is $ARBORPATH/bin/MigratedUsersPassword.txt.</td>
</tr>
<tr>
<td>flag</td>
<td>ESS_USHORT_T</td>
<td>Whether the passwords file, if already existing, should be overwritten.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[ ] ESS_FILE_OVERWRITE—Overwrite</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[ ] ESS_FILE_NOOVERWRITE—Do not overwrite; return an error.</td>
</tr>
</tbody>
</table>

**Return Value**

Returns 0 if successful; otherwise, returns an error.

**Access**

This function requires the caller to be an Administrator.

**Example**

```c
ESS_FUNC_M ESS_SS_SetUserToSS(ESS_HCTX_T hCtx, ESS_HINST_T hInst)
{
    ESS_STS_T         sts = ESS_STS_NOERR;
    ESS_USHORT_T      option;
    ESS_STR_T         userName = ESS_NULL;
    ESS_STR_T         newPassword = ESS_NULL;
    ESS_STR_T         fName = ESS_NULL;
    ESS_USHORT_T      flag = ESS_FILE_OVERWRITE;

    sts = EssAlloc(hInst, sizeof(ESS_USERNAME_T), &userName);
    if(sts)
        return (sts);
    memset(userName, 0, sizeof(ESS_USERNAME_T));
    strcpy( userName, "essexer");

    /* New Shared Services Native User Password Option: */
    /* *
    * 0 to use user provided password
    * 1 to use the user name as password
    * 2 to automatically generate a password
    **/

    option = 2; /* Generate password */
```
sts = EssSetUserToSS(hCtx, userName, option, newPassword, fName, flag);

if (sts)
    printf("Failed to migrate User %s to Shared Services mode.\n", userName);

if (userName)
    EssFree(hInst, userName);

return (sts);
}

See also an extended Appendix B

See Also

- **EssSetUsersToSS**
- **EssSetGroupToSS**
- **EssSetSSSecurityMode**

---

### EssSetUser

Sets a user information structure, which contains security information for the user.

**Syntax**

```c
ESS_FUNC_M EssSetUser (hCtx, pUserInfo);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>pUserInfo</td>
<td>&quot;ESS_USERINFO_T, ESS_GROUPINFO_T&quot; on page 192</td>
<td>Pointer to user info structure.</td>
</tr>
</tbody>
</table>

**Notes**

- The name of the user to set is a field in the user info structure, which must always be specified.
- The only fields in the user info structure which may be changed using this function are the `Access`, `Expiration`, and `PwdChgNow` fields (the other fields are for information only). See the description of the ESS_USERINFO_T structure for more information.
- The caller cannot give the specified user any access privileges that they themselves do not already have.
- The new user settings will take effect the next time the user logs in.

**Return Value**

Returns zero (0) if successful.

**Access**

This function requires the caller to have Create/Delete User privilege (ESS_PRIV_USERCREATE) for the logged in server.
Example

```c
ESS_FUNC_M
ESS_SetUser (ESS_HCTX_T hCtx)
{
    ESS_FUNC_M        sts  = ESS_STS_NOERR;
    ESS_USERINFO_T    User;

    strcpy(User.Name, "Jim Smith");
    strcpy(User.AppName, "Sample");
    strcpy(User.DbName, "Basic");
    User.Access = ESS_ACCESS_SUPER;

    sts = EssSetUser (hCtx, &User);
    return (sts);
}
```

See Also

- EssGetUser
- EssListUsers
- EssSetApplicationAccess
- EssSetPassword

**EssSetUserEx**

Sets a user information structure, which contains security information for the user.

**Syntax**

```c
ESS_FUNC_M EssSetUserEx (hCtx, pUserInfo);
```

**Parameter** | **Data Type** | **Description**
---|---|---
 hCtx | ESS_HCTX_T | API context handle.
 pUserInfo | “ESS_USERINFOEX_T” on page 195 | Pointer to info structure of externally authenticated user.

**Notes**

- The name of the user to set is a field in the user info structure, which must always be specified.
- The only fields in the user info structure which may be changed using this function are the `Access`, `Expiration`, and `PwdChgNow` fields (the other fields are for information only). See the description of the `ESS_USERINFO_T` structure for more information.
- The caller cannot give the specified user any access privileges that they themselves do not already have.
- The new user settings will take effect the next time the user logs in.

**Return Value**

Returns zero (0) if successful.
Access

This function requires the caller to have Create/Delete User privilege (ESS_PRIV_USERCREATE) for the logged in server.

Example

ESS_FUNC_M
ESS_SetUser (ESS_HCTX_T hCtx)
{
    ESS_FUNC_M        sts  = ESS_STS_NOERR;
    ESS_USERINFO_T   User;

    strcpy(User.Name,"Jim Smith");
    strcpy(User.AppName,"Sample");
    strcpy(User.DbName,"Basic");
    User.Access = ESS_ACCESS_SUPER;

    sts = EssSetUserEx (hCtx,&User);
    return (sts);
}

See Also

- EssGetUser
- EssListUsers
- EssCreateExtUser
- EssGetUserEx
- “ESS_USERINFOEX_T” on page 195
- EssSetApplicationAccess
- EssSetPassword

EssSetUsersToSS

Migrates all users to Oracle Hyperion Enterprise Performance Management System security mode. This might be useful if the user migration failed using EssSetSSSecurityMode.

Syntax

ESS_FUNC_M EssSetUsersToSS (hCtx, option, NewPassword, FileName, flag);

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>----------------</td>
<td>-------------</td>
</tr>
<tr>
<td>option</td>
<td>ESS_USHORT_T</td>
<td>Integer representing the desired password creation method for migrated users.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- 0—Use an administrator-specified password.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- 1—Create passwords that are the same as user names for users being migrated to Shared Services.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Note: The passwords are created as lower-case, even if there are upper-case letters in the user name.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- 2—Automatically generate new passwords for the users being migrated to Oracle's Hyperion® Shared Services.</td>
</tr>
<tr>
<td>NewPassword</td>
<td>ESS_STR_T</td>
<td>Password string (if option 2 is used).</td>
</tr>
<tr>
<td>FileName</td>
<td>ESS_STR_T</td>
<td>Name of file to contain saved passwords. If null, the default file is $ARBORPATH/bin/MigratedUsersPassword.txt.</td>
</tr>
<tr>
<td>flag</td>
<td>ESS_USHORT_T</td>
<td>Whether the passwords file, if already existing, should be overwritten.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_FILE_OVERWRITE—Overwrite</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_FILE_NOOVERWRITE—Do not overwrite; return an error.</td>
</tr>
</tbody>
</table>

**Return Value**

Returns 0 if successful; otherwise, returns an error.

**Access**

This function requires the caller to be an Administrator.

**Example**

```c
ESﬆ_FUNC_M ESS_SS_SetUsersToSS(ESS_HCTX_T hCtx, ESS_HINST_T hInst)
{
    ESS_STS_T         sts = ESS_STS_NOERR;
    ESS_USHORT_T      option;
    ESS_STR_T         newPassword = ESS_NULL;
    ESS_STR_T         fName = "PasswordList.txt";

    /* New Shared Services Native User Password Option: */
    *  
    * 0 to use user provided password
    * 1 to use the user name as password
    * 2 to automatically generate a password
    **/

    option = 2; /* Generate a password */

    sts = EssSetUsersToSS(hCtx, option, newPassword, fName, ESS_FILE_OVERWRITE);
```
if(sts)
    printf("Failed to migrate Users to Shared Services mode.\n");

return (sts);
}

See also an extended Appendix B

See Also
● EssSetUserToSS
● EssSetGroupToSS
● EssSetSSSecurityMode

EssSetUserType

Enables you to define the application access type for a user. You can add, remove, or replace different application access types for a user name.

The application access type is a user property. If a user is created in Oracle Hyperion Planning, Fusion Edition, it automatically has an application access type of planning; if a user is created in Administration Services, it automatically has an application access type of Essbase. Once a user is created, the application access type can be modified in the corresponding application using EssSetUserType.

Syntax

ESS_FUNC_M EssSetUserType (hCtx, UserName, UserType, Cmd)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>Essbase API context handle.</td>
</tr>
<tr>
<td>UserName</td>
<td>ESS_STR_T</td>
<td>Name of user.</td>
</tr>
<tr>
<td>UserType</td>
<td>ESS_USER_TYPE_T</td>
<td>ESS_USER_ESSBASE is the application access type, if no application access type is specified. This user will be enabled with all functionality.</td>
</tr>
<tr>
<td>Cmd</td>
<td>ESS_USERTYPE_CMD_T</td>
<td>Whether to add/remove/replace the type specified. Only the Essbase type, ESS_USER_ESSBASE, can be added or removed.</td>
</tr>
</tbody>
</table>

● ESS_USERTYPE_CMD_ADD—Adds the new type specified to the existing application access type.
● ESS_USERTYPE_CMD_REMOVE—Removes the types specified from the existing application access type.

Return Value

Returns the status of the API.

Example

ESS_FUNC_M
ESS_SetUserType (ESS_HCTX_T hCtx)
EssBase Agent. This function sends a request to the Agent (ESSBASE.EXE) to shut itself down. The Agent then goes through its normal shutdown procedure, including committing data, stopping all applications and databases, and logging users off before stopping.

Only users with Supervisor privilege can shut down the Agent.

This function can be called at any time, however, it is normally called to shut down an Agent process which was started in the background. See the Oracle Essbase Database Administrator’s Guide for details.

**Syntax**

```
ESS_FUNC_M EssShutdownServer (hInstance, Server, UserName, Password);
```

**Parameter** | **Data Type** | **Description**
--- | --- | ---
| hInstance | ESS_HINST_T | API instance handle.
| Server | ESS_STR_T | Network server name string. Specifies the name of the server to shut down.
| UserName | ESS_STR_T | User name string. Specifies the user who is requesting the shutdown.
| Password | ESS_STR_T | Password string. Specifies the password of the user requesting the shutdown.

**Return Value**

Possible error conditions resulting from this function include:

- Insufficient privilege for this operation, AD_AMSG_IPO
- Incorrect password, AD_AMSG_IPW
- User does not exist, AD_AMSG_UNE
- Cannot shutdown application, AD_MSGAR_NOSHUTDOWN
- Network Error: Unable To Locate In Hosts File, NET_TCP_HOSTS
- Network error: Cannot locate server, NET_NP_NOSERVER

**Access**

This function requires Supervisor privilege.
Example

```c
ESS_FUNC_M
ESS_ShutdownServer (ESS_HINST_T hInst)
{
    ESS_FUNC_M     sts = ESS_STS_NOERR;
    ESS_STR_T     Server;
    ESS_STR_T     UserName;
    ESS_STR_T     Password;
    Server = "Rainbow";
    UserName = "Admin";
    Password = "password";
    sts = EssShutdownServer(hInst, Server,
                           UserName, Password);
    return (sts);
}
```

See Also

- EssSetPassword
- EssUnloadApplication
- EssUnloadDatabase

EssTerm

Terminates the API and releases all system resources used by the API. This function should normally be called after all other API calls have been completed, immediately prior to terminating your program.

Syntax

```c
ESS_FUNC_M   EssTerm (hInstance);
```

Parameter | Data Type | Description
--- | --- | ---
hInstance | ESS_HINST_T | API instance handle.

Notes

Because this function terminates use of the Essbase API, any API functions (other than EssInit()) called after this function has been executed will return an error.

Return Value

None.

Access

This function requires no special access.

Example

```c
/* Terminate the Essbase API */
if ((sts = EssTerm (hInstance)) != ESS_STS_NOERR)
{
    /* error terminating API */
```

exit ((ESS_USHORT_T) sts);
}

See Also

- EssInit

**EssUnloadApplication**

Stops an application on the server.

**Syntax**

```c
ESS_FUNC_M EssUnloadApplication (hCtx, AppName);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle</td>
</tr>
<tr>
<td>AppName</td>
<td>ESS_STR_T</td>
<td>Name of application to load</td>
</tr>
</tbody>
</table>

**Notes**

To unload an application, the connected user must have load access to the application. An application cannot be unloaded if Essbase is restructuring a database associated with the application.

**Return Value**

None.

**Access**

This function requires the caller to have Application Load/Unload privilege (ESS_PRIV_APPLOAD) for the specified application.

**Example**

```c
ESS_FUNC_M
ESS_UnloadApplication (ESS_HCTX_T hCtx)
{
    ESS_FUNC_M    sts = ESS_STS_NOERR;
    ESS_STR_T     AppName;

    AppName = "Sample";
    sts = EssUnloadApplication(hCtx, AppName);

    return (sts);
}
```

See Also

- EssLoadApplication
- EssUnloadDatabase
**EssUnloadDatabase**

Stops a database within an application on the server.

**Syntax**

```c
ESS_FUNC_M EssUnloadDatabase (hCtx,AppName,DbName);
```

**Parameter** | **Data Type** | **Description**
--- | --- | ---
ffmpeg hbox | ESS_HCTX_T | API context handle.
AppName | ESS_STR_T | Application name.
DbName | ESS_STR_T | Name of database to unload.

**Return Value**

None.

**Access**

This function requires the caller to have database load/unload privilege (ESS_PRIV_APPLOAD).

**Example**

```c
ESS_FUNC_M
Ess_UnloadDb (ESS_HCTX_T hCtx)
{
    ESS_FUNC_M stes = ESS_STS_NOERR;
    ESS_STR_T AppName;
    ESS_STR_T DbName;
    AppName = "Sample";
    DbName = "Basic";
    /*
    * IF the current active is the same as the
    * unload db, ClearActive first
    */
    stes = EssClearActive(hCtx);
    /*
    * ELSE
    */
    stes = EssUnloadDatabase(hCtx,AppName,
                            DbName);
    return (stes);
}
```

**See Also**

- EssLoadDatabase
- EssUnloadDatabase

**EssUnlockObject**

Unlocks a locked object on the server or client object system.
Syntax

```c
ESS_FUNC_M EssUnlockObject (hCtx, ObjType, AppName, DbName, ObjName);
```

Parameter | Data Type  | Description
---|---|---
hCtx | ESS_HCTX_T | API context handle. Can be local context handle returned by EssCreateLocalContext().
ObjType | ESS_OBJTYPE_T | Object type (must be single type). Refer to “Bitmask Data Types (C)” on page 96 for a list of possible values.
AppName | ESS_STR_T | Application name
DbName | ESS_STR_T | Databasename. If NULL, uses the Application subdirectory.
ObjName | ESS_STR_T | Name of object to unlock

Notes

To unlock an object, the object must already exist and be locked by the caller.

Return Value

None.

Access

This function requires the caller to have Application or Database Design privilege (ESS_PRIV_APPDESIGN or ESS_PRIV_DBDESIGN) for the specified application or database containing the object.

Example

```c
ESS_FUNC_M
ESS_UnlockObject (ESS_HCTX_T hCtx)
{
    ESS_FUNC_M         sts = ESS_STS_NOERR;
    ESS_STR_T         AppName;
    ESS_STR_T         DbName;
    ESS_STR_T         ObjName;
    ESS_OBJTYPE_T     ObjType;

    AppName = "Sample";
    DbName  = "Basic";
    ObjName = "Basic";
    ObjType = ESS_OBJTYPE_OUTLINE;

    sts = EssUnlockObject(hCtx, ObjType, AppName, DbName, ObjName);
    if(!sts)
        printf("The Object is unlocked\r\n");
    return (sts);
}
```

See Also

- EssGetObject
- EssGetObjectInfo
EssUpdate

Sends an update specification to the active database as a single string.

Syntax

```c
ESS_FUNC_M EssUpdate (hCtx, Store, Unlock, UpdtSpec);
```

Parameter | Data Type | Description
---|---|---

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>Store</td>
<td>ESS_BOOL_T</td>
<td>Controls storage of data. If TRUE, data is stored in the server; if FALSE, no data is stored.</td>
</tr>
<tr>
<td>Unlock</td>
<td>ESS_BOOL_T</td>
<td>Controls unlocking of data blocks. If TRUE, all relevant blocks which are locked will be unlocked (after data is stored, if necessary). If FALSE, no blocks are unlocked.</td>
</tr>
<tr>
<td>UpdtSpec</td>
<td>ESS_STR_T</td>
<td>The update specification, as a single string (must be less than 32 KB).</td>
</tr>
</tbody>
</table>

Notes

- This function is equivalent to making a call to `EssBeginUpdate()`, followed by calls to `EssSendString()` and finally `EssEndUpdate()`. The update data can either be stored in the database, or just verified and any errors returned. Also, any data blocks locked for update can be unlocked by this call.
- The update specification string must be less than 32 KB long.
- If this function causes data to be stored (`Store` flag is TRUE), the relevant data blocks must previously have been locked for update (for example, by calling `EssReport()` with the `Lock` flag set to TRUE).
- If the caller attempts to write data to a member it does not have permission to write to, a warning is generated, and the member is not updated.
- If both the `Store` and `Unlock` flags are set to FALSE, the database merely performs a syntax check of the update specification.

Return Value

None.

Access

This function requires the caller to have write privilege (ESS_PRIV_WRITE) to the active database. If the caller attempts to write information.

Example

```c
ESS_FUNC_M
ESS_Update (ESS_HCTX_T   hCtx)
```
ESS_FUNC_M     sts = ESS_STS_NOERR;

    sts = EssUpdate (hCtx, ESS_TRUE, ESS_FALSE,
                   "Year Market Scenario Measures Product 100");
    return(sts);
}

See Also

- EssBeginUpdate
- EssEndUpdate
- EssReport
- EssSendString
- EssUpdateFile

**EssUpdateBakFile**

Compares the security backup file `essbase_timestamp.bak` to the security file `essbase.sec` and overwrites the backup file with the security file if the two files do not match.

**Syntax**

```c
ESS_FUNC_M EssUpdateBakFile (hCtx);
```

**Parameter**  | **Data Type**  | **Description**
--- | --- | ---

hCtx;  | ESS_HCTX_T  | Context handle

**Notes**

Essbase compares the security file and the backup file every time the server starts. The MaxL statement `alter system sync security_backup` can be used to automatically compare the security backup file to the security file at specified intervals or on demand. When Essbase compares the files, it updates the backup file to match the security file.

**Return Value**

If successful, returns zero.

**Access**

This function requires the caller to be an Administrator.

**Example**

```c
ESS_FUNC_M
Ess_UpdateBakFile (ESS_HCTX_T hCtx)
{
    ESS_FUNC_M     sts = ESS_STS_NOERR;

    sts = EssUpdateBakFile(hCtx);

    return sts;
}
```
EssUpdateDrillThruURL

Updates a drill-through URL, with the given name, within the active database outline.

See “Drill-through URL Limits” on page 1729.

Syntax

ESS_FUNC_M EssUpdateDrillThruURL (hCtx, ESS_PDURLINFO_T pUrl);

Parameter | Data Type | Description
--- | --- | ---
hCtx | ESS_HCTX_T | API context handle
pUrl | ESS_PDURLINFO_T | URL definition
bMerge | ESS_BOOL_T | If True, add drill-through region definitions in pUrl to the existing list of drill-through regions in the named URL definition. If False, replace the existing list of drill-through region definitions with the list in pUrl.

Return Value

- If successful, updates the named drill-through URL in the active database by replacing the URL XML and either updating or replacing the drill-through region list with the corresponding fields in pUrl.
- If there is no URL with the given name, returns an error code.

Access

- Caller must have database Design privilege (ESS_PRIV_DBDESIGN) for the specified database.
- Caller must have selected the specified database as the active database using EssSetActive().

Example

/* Sample Code for EssUpdateDrillThruURL */

ESS_STS_T sts = ESS_STS_NOERR;
ESS_DURLINFO_T url;
ESS_PDURLINFO_T urlInfo;
ESS_STR_T fileName = "";
ESS_CHAR_T xmlString[XML_CHAR_MAX];
ESS_BOOL_T bMerge;
ESS_USHORT_T i;
memset(&url, '\0', sizeof(ESS_DURLINFO_T));
fileName = "F:\testarea\mainapi\sample1.xml"
GetFileContent(fileName, xmlString);

/* Update URL*/
url.bIsLevel0 = ESS_TRUE;
url.cpURLName = "Drill Through to EPMI";
url.cpURLXml = xmlString;
url.iURLXmlSize = (ESS_SHORT_T) strlen(xmlString)+1;
url.iCountOfDrillRegions = 1;
sts = EssAlloc (hInst, sizeof(ESS_STR_T) * url.iCountOfDrillRegions, &url.cppDrillRegions);

/* With hMerge = ESS_FALSE, update Drill Regions */

bMerge = ESS_FALSE;   // replace
url.cppDrillRegions[0] = "Mar";
sts = EssUpdateDrillThruURL(hCtx, &url, bMerge);
printf("EssUpdateDrillThruURL sts: %ld\n", sts);

---

**EssUpdateEx**

Sends an update specification to the active database as a single string, capturing any data load errors in `ppMbrError`.

**Syntax**

```c
ESS_FUNC_M EssUpdateEx (hCtx, Store, Unlock, UpdtSpec, ppMbrError);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>Store</td>
<td>ESS_BOOL_T</td>
<td>Controls storage of data. If TRUE, data is stored in the server; if FALSE, no data is stored.</td>
</tr>
<tr>
<td>Unlock</td>
<td>ESS_BOOL_T</td>
<td>Controls unlocking of data blocks. If TRUE, all relevant blocks which are locked will be unlocked (after data is stored, if necessary). If FALSE, no blocks are unlocked.</td>
</tr>
<tr>
<td>UpdtSpec</td>
<td>ESS_STR_T</td>
<td>The update specification, as a single string (must be less than 32 KB).</td>
</tr>
<tr>
<td>ppMbrError</td>
<td>ESS_PPMBRERR_T</td>
<td>Pointer to linked list of errors contained in ESS_MBRERR_T. Possible errors are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- AD_MSGDL_ERRORLOAD—Unable to do dataload at Item/Record [number].</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_MBRERR_BADDATA—Invalid member [membername] in data column.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_MBRERR_DBACCESS—You have insufficient access privilege to perform a lock on this database.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_MBRERR_DUPLICATE—Duplicate members from the same dimension on data record, [number] records completed.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_MBRERR_UNKNOWN—Unknown member [membername] in dataload, [number] records returned.</td>
</tr>
</tbody>
</table>

**Notes**

- The update data can either be stored in the database, or just verified and any errors returned. Also, any data blocks locked for update can be unlocked by this call.
- The update specification string must be less than 32 KB long.
- If the caller attempts to write data to a member it does not have permission to write to, a warning is generated, and the member is not updated.
- If both the `Store` and `Unlock` flags are set to FALSE, the database merely performs a syntax check of the update specification.
**Return Value**

Returns zero if successful; otherwise, returns an error code and the records that caused the error.

**Access**

This function requires the caller to have write privilege (ESS_PRIV_WRITE) to the active database.

**Example**

```c
void TestUpdateEx()
{
    ESS_STS_T sts = ESS_STS_NOERR;
    ESS_PMBERR_T pMbrError;
    ESS_STR_T updtSpec = "";

    sts = EssUpdateEx(hCtx, ESS_TRUE, ESS_FALSE, "'Jan' 'New York' 'Actual' 'Sales' '100-10' 123 \n '100-20' 345 \n '100-30' 678", &pMbrError);
    printf("EssUpdateEx sts: %ld\n",sts);
    if(!sts)
    {
        printf("\nVerify data:\n");
        VerifyDataload("'Jan' 'New York' 'Actual' 'Sales' <IDESC '100'!");

        printf("\nMember Error Info:\n");
        if(pMbrError)
            DisplayError(pMbrError);
        else
            printf("\tError structure is empty.\n");
    }

    if(pMbrError)
        EssFree(hInst, pMbrError);
}
```

**See Also**

- EssUpdateFileASO
- EssUpdateFileASOEx
- EssUpdateFileEx
- EssUpdateFileUTF8ASOEx
- EssUpdateFileUtf8Ex
- EssUpdateFileUTF8ASO
- EssUpdateUtf8Ex

**EssUpdateFile**

Sends an update specification to the active database from a file. The update data can either be stored in the database, or just verified and any errors returned. Also, any data blocks locked for update can be unlocked by this call.

**Syntax**

```c
ESS_FUNC_M EssUpdateFile (hDestCtx, hSrcCtx, AppName, DbName, FileName, Store, Unlock);
```
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hDestCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle of target database on the server.</td>
</tr>
<tr>
<td>hSrcCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle for report file location. The report file can reside on the client or on the same server as the target database.</td>
</tr>
<tr>
<td>AppName</td>
<td>ESS_STR_T</td>
<td>Application name for update file location.</td>
</tr>
<tr>
<td>DbName</td>
<td>ESS_STR_T</td>
<td>Database name for update file location.</td>
</tr>
<tr>
<td>FileName</td>
<td>ESS_STR_T</td>
<td>Name of update specification file.</td>
</tr>
<tr>
<td>Store</td>
<td>ESS_BOOL_T</td>
<td>Controls storage of data. If TRUE, data is stored in the server; if FALSE, no data is stored.</td>
</tr>
<tr>
<td>Unlock</td>
<td>ESS_BOOL_T</td>
<td>Controls unlocking of data blocks. If TRUE, all relevant blocks which are locked will be unlocked (after data is stored, if necessary). If FALSE, no blocks are unlocked.</td>
</tr>
</tbody>
</table>

**Notes**

- If this function causes data to be stored (Store flag is TRUE), the relevant data blocks must previously have been locked for update (e.g. by calling EssReport() with the Lock flag set to TRUE).
- If both the Store and Unlock flags are set to FALSE, the database merely performs a syntax check of the update specification.

**Return Value**

None.

**Access**

This function requires the caller to have write privilege (ESS_PRIV_WRITE) to the active database.

**Example**

```c
ESS_FUNC_M
ESS_UpdateFile (ESS_HCTX_T hCtx) {
    ESS_FUNC_M sts = ESS_STS_NOERR;
    ESS_HCTX_T hSrcCtx;
    ESS_BOOL_T isStore;
    ESS_BOOL_T isUnlock;
    ESS_STR_T  AppName;
    ESS_STR_T  DbName;
    ESS_STR_T  FileName;

    AppName = "Sample";
    DbName  = "Basic";
    hSrcCtx = hCtx;
    isStore = ESS_TRUE;
    isUnlock = ESS_FALSE;
    sts = EssUpdateFile (hCtx, hSrcCtx, AppName, DbName, FileName, isStore, isUnlock);
    return(sts);
}
```
EssUpdateFileASO

Sends an update specification to the active aggregate storage database from a file.

Syntax

```c
ESS_FUNC_M EssUpdateFileASO (hDestCtx, hSrcCtx, AppName, DbName, FileName, Store, Unlock, ulBufferId);
```

Parameter | Data Type | Description
--- | --- | ---
hDestCtx | ESS_HCTX_T | API context handle of target database on the server
hSrcCtx | ESS_HCTX_T | API context handle for update file location. The update file can reside on the client or on the same server as the target database.
AppName | ESS_STR_T | Application name for update file location
DbName | ESS_STR_T | Database name for update file location
FileName | ESS_STR_T | Name of update specification file
Store | ESS_BOOL_T | Controls storage of data. If TRUE, data is stored in the server; if FALSE, no data is stored.
Unlock | ESS_BOOL_T | Not supported for aggregate storage databases. You must always pass ESS_FALSE for this parameter.
ulBufferId | ESS_ULONG_T | ID number for the data load buffer.

Notes

If the `Store` flag is set to FALSE, the database merely performs a syntax check of the update specification.

Return Value

Returns zero if successful; otherwise, returns an error code.

Access

This function requires the caller to have write privilege (ESS_PRIV_WRITE) to the active database.

Example

```c
void TestUpdateFileASO(ESS_HCTX_T hCtx, ESS_STR_T AppName, ESS_STR_T DbName) {
    ESS_STS_T       sts = ESS_STS_NOERR;
    ESS_HCTX_T      hSrcCtx;
    ESS_BOOL_T      isStore;
    ESS_BOOL_T      isUnlock;
```
ESS_STR_T FileName;
ESS_ULONG_T ulBufferId;
ESS_ULONG_T ulDuplicateAggregationMethod;
ESS_ULONG_T ulOptionsFlags;
ESS_ULONG_T ulSize;
ESS_ULONG_T ulBufferCnt;
ESS_ULONG_T ulCommitType;
ESS_ULONG_T ulActionType;
ESS_ULONG_T ulOptions;
ESS_ULONG_T ulBufferIdAry[1];

ulDuplicateAggregationMethod = ESS_ASO_DATA_LOAD_BUFFER_DUPLICATES_ADD;
ulOptionsFlags = ESS_ASO_DATA_LOAD_BUFFER_IGNORE_MISSING_VALUES;
ulSize = 100;
ulBufferId = 101;
sts = EssLoadBufferInit(hCtx, AppName, DbName, ulBufferId,
ulDuplicateAggregationMethod,
    ulOptionsFlags, ulSize);
printf("EssLoadBufferInit sts: %ld\n", sts);

/* Update from server*/
hSrcCtx = hCtx;
isStore = ESS_TRUE;
isUnlock = ESS_FALSE;
FileName = "data1.txt";

sts = EssUpdateFileASO (hCtx, hSrcCtx, AppName, DbName, FileName, isStore,
isUnlock, ulBufferId);
printf("EssUpdateFileASO sts: %ld\n",sts);

/* Commit and delete the buffer */
ulBufferCnt = 1;
ulBufferIdAry[0] = ulBufferId;
ulCommitType = ESS_ASO_DATA_LOAD_BUFFER_STORE_DATA;
ulActionType = ESS_ASO_DATA_LOAD_BUFFER_COMMIT;
printf("\nLoad data to main slice and destroy buffer: \n");
ulOptions = ESS_ASO_DATA_LOAD_INCR_TO_MAIN_SLICE;
sts = EssLoadBufferTerm(hCtx, AppName, DbName, ulBufferCnt, ulBufferIdAry,
ulCommitType,
    ulActionType, ulOptions);
printf("EssLoadBufferTerm sts: %ld\n",sts);
}

See Also
● EssUpdateEx
● EssUpdateFileASOEx
● EssUpdateFileEx
● EssUpdateFileUTF8ASOEx
● EssUpdateFileUtf8Ex
● EssUpdateFileUTF8ASO
● EssUpdateUtf8Ex
**EssUpdateFileASOEx**

Sends an update specification to the active aggregate storage database from a file, capturing any data load errors in *ppMbrError*.

**Syntax**

```c
ESS_FUNC_M EssUpdateFileASOEx (hDestCtx, hSrcCtx, AppName, DbName, FileName, Store, Unlock, ulBufferId, ppMbrError);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hDestCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle of target database on the server.</td>
</tr>
<tr>
<td>hSrcCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle for update file location. The update file can reside on the client or on the same server as the target database.</td>
</tr>
<tr>
<td>AppName</td>
<td>ESS_STR_T</td>
<td>Application name for update file location.</td>
</tr>
<tr>
<td>DbName</td>
<td>ESS_STR_T</td>
<td>Database name for update file location</td>
</tr>
<tr>
<td>FileName</td>
<td>ESS_STR_T</td>
<td>Name of update specification file.</td>
</tr>
<tr>
<td>Store</td>
<td>ESS_BOOL_T</td>
<td>Controls storage of data. If TRUE, data is stored in the server; if FALSE, no data is stored.</td>
</tr>
<tr>
<td>Unlock</td>
<td>ESS_BOOL_T</td>
<td>Not supported for aggregate storage databases. You must always pass ESS_FALSE for this parameter.</td>
</tr>
<tr>
<td>ulBufferId</td>
<td>ESS_ULONG_T</td>
<td>ID number for the data load buffer.</td>
</tr>
<tr>
<td>ppMbrError</td>
<td>ESS_PPMBRERR_T</td>
<td>Pointer to linked list of errors contained in ESS_MBRERR_T. Possible errors are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● AD_MSGDL_ERRORLOAD—Unable to do dataload at Item/Record [number].</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESS_MBRERR_BADDATA—Invalid member [membername] in data column.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESS_MBRERR_DBACCESS—You have insufficient access privilege to perform a lock on this database.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESS_MBRERR_DUPLICATE—Duplicate members from the same dimension on data record, [number] records completed.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESS_MBRERR_UNKNOWN—Unknown member [membername] in dataload, [number] records returned.</td>
</tr>
</tbody>
</table>

**Notes**

If the Store flag is set to FALSE, the database merely performs a syntax check of the update specification.

**Return Value**

Returns zero if successful; otherwise, returns an error code and the records that caused the error.

**Access**

This function requires the caller to have write privilege (ESS_PRIV_WRITE) to the active database.
Example

```c
void TestUpdateFileASOEx()
{
    ESS_STS_T sts = ESS_STS_NOERR;
    ESS_HCTX_T hSrcCtx;
    ESS_BOOL_T isStore;
    ESS_BOOL_T isUnlock;
    ESS_STR_T FileName;
    ESS_ULONG_T ulBufferId;
    ESS_ULONG_T ulDuplicateAggregationMethod;
    ESS_ULONG_T ulOptionsFlags;
    ESS_ULONG_T ulSize;
    ESS_ULONG_T ulBufferCnt;
    ESS_ULONG_T ulCommitType;
    ESS_ULONG_T ulActionType;
    ESS_ULONG_T ulOptions;
    ESS_ULONG_T ulBufferIdAry[1];
    ESS_PMBRERR_T pMbrError;

    ulDuplicateAggregationMethod = ESS_ASO_DATA_LOAD_BUFFER_DUPLICATES_ADD;
    ulOptionsFlags = ESS_ASO_DATA_LOAD_BUFFER_IGNORE_MISSING_VALUES;
    ulSize = 1;
    ulBufferId = 101;
    sts = EssLoadBufferInit(hCtx, AppName, DbName, ulBufferId,
                           ulDuplicateAggregationMethod,
                           ulOptionsFlags, ulSize);
    printf("EssLoadBufferInit sts: %ld\n", sts);

    /* Update from server*/
    hSrcCtx = hCtx;
    isStore = ESS_TRUE;
    isUnlock = ESS_FALSE;
    FileName = "apgeaso1.txt";

    sts = EssUpdateFileASOEx (hCtx, hSrcCtx, AppName, DbName, FileName, isStore,
                             isUnlock, ulBufferId, &pMbrError);
    printf("EssUpdateFileASOEx sts: %ld\n",sts);
    if(!sts)
    {
        printf("\nMember Error Info:\n");
        if(pMbrError)
            DisplayError(pMbrError);
        else
            printf("\nError structure is empty.\n");
    }

    ulBufferCnt = 1;
    ulBufferIdAry[0] = ulBufferId;
    ulCommitType = ESS_ASO_DATA_LOAD_BUFFER_STORE_DATA;
    ulActionType = ESS_ASO_DATA_LOAD_BUFFER_COMMIT;
    printf("\nIncrement to main slice and destroy buffer:\n");
    ulOptions = ESS_ASO_DATA_LOAD_INCR_TO_MAIN_SLICE;
    sts = EssLoadBufferTerm(hCtx, AppName, DbName, ulBufferCnt, ulBufferIdAry,
                            ulCommitType,
                            ulActionType, ulOptions);
    printf("EssLoadBufferTerm sts: %ld\n",sts);
}
```

658  C Main API Functions
if(!sts)
{
    VerifyDataload("'Mar' 'Sale' 'Curr Year' 'Original Price' '017589' '13668'
    'Cash' 'No Promotion' '1 to 13 Years' 'Under 20,000' 'Digital Cameras' 10
    'Camcorders' 20
    'Photo Printers' 30 !");
}
if(pMbrError)
    EssFree(hInst, pMbrError);
}

See Also

- EssUpdateEx
- EssUpdateFileEx
- EssUpdateFileASO
- EssUpdateFileUTF8ASOEx
- EssUpdateFileUtf8Ex
- EssUpdateFileUTF8ASO
- EssUpdateUtf8Ex

---

**EssUpdateFileEx**

Sends an update specification to the active database from a file, capturing any data load errors in `ppMbrError`.

**Syntax**

```c
ESS_FUNC_M EssUpdateFileEx (hDestCtx, hSrcCtx, AppName, DbName, FileName, Store, Unlock, ppMbrError);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hDestCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle of the target database on the server.</td>
</tr>
<tr>
<td>hSrcCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle for update file location. The update file can reside on</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the client or on the same server as the target database.</td>
</tr>
<tr>
<td>AppName</td>
<td>ESS_STR_T</td>
<td>Application name for update file location.</td>
</tr>
<tr>
<td>DbName</td>
<td>ESS_STR_T</td>
<td>Database name for update file location.</td>
</tr>
<tr>
<td>FileName</td>
<td>ESS_STR_T</td>
<td>Name of update specification file.</td>
</tr>
<tr>
<td>Store</td>
<td>ESS_BOOL_T</td>
<td>Controls storage of data. If TRUE, data is stored in the server; if FALSE,</td>
</tr>
<tr>
<td>Unlock</td>
<td>ESS_BOOL_T</td>
<td>Controls unlocking of data blocks. If TRUE, all relevant blocks which are</td>
</tr>
<tr>
<td></td>
<td></td>
<td>locked will be unlocked (after data is stored, if necessary). If FALSE,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>no blocks are unlocked.</td>
</tr>
</tbody>
</table>


Parameter | Data Type | Description
--- | --- | ---
ppMbrError | ESS_PPMBRERR_T | Pointer to linked list of errors contained in ESS_MBRERR_T. Possible errors are:
  - AD_MSGDL_ERRORLOAD—Unable to do dataload at Item/Record [number].
  - ESS_MBRERR_BADDATA—Invalid member [membername] in data column.
  - ESS_MBRERR_DBACCESS—You have insufficient access privilege to perform a lock on this database.
  - ESS_MBRERR_DUPLICATE—Duplicate members from the same dimension on data record, [number] records completed.
  - ESS_MBRERR_UNKNOWN—Unknown member [membername] in dataload, [number] records returned.

Notes
- The update data can either be stored in the database, or just verified and any errors returned. Also, any data blocks locked for update can be unlocked by this call.
- If the caller attempts to write data to a member it does not have permission to write to, a warning is generated, and the member is not updated.
- If both the Store and Unlock flags are set to FALSE, the database merely performs a syntax check of the update specification.

Return Value
Returns zero if successful; otherwise, returns an error code and the records that caused the error.

Access
This function requires the caller to have write privilege (ESS_PRIV_WRITE) to the active database.

Example
```c
void TestUpdateFileEx()
{
    ESS_STS_T sts = ESS_STS_NOERR;
    ESS_HCTX_T hSrcCtx;
    ESS_BOOL_T isStore;
    ESS_BOOL_T isUnlock;
    ESS_STR_T FileName;
    ESS_PMBRERR_T pMbrError;

    hSrcCtx = hCtx;
    FileName = "apgebso1.txt";
    isStore = ESS_TRUE;
    isUnlock = ESS_FALSE;

    sts = EssUpdateFileEx (hCtx, hSrcCtx, AppName, DbName, FileName, isStore, isUnlock, &pMbrError);
    printf("EssUpdateFileEx sts: %ld\n",sts);
    if(!sts)
    {
        printf("\nVerify data:\n");
    }
```
VerifyDataload("'Jan' 'New York' 'Actual' 'Sales' <IDESC '100'!");

printf("\nMember Error Info: \n");
if(pMbrError)
    DisplayError(pMbrError);
else
    printf("\tError structure is empty.\n");
}

if(pMbrError)
    EssFree(hInst, pMbrError);
}

See Also

- **EssUpdateEx**
- **EssUpdateFileASO**
- **EssUpdateFileASOEx**
- **EssUpdateFileUTF8ASOEx**
- **EssUpdateFileUtf8Ex**
- **EssUpdateFileUTF8ASO**
- **EssUpdateUtf8Ex**

## EssUpdateFileUTF8ASO

Sends an update specification to the active aggregate storage database from a UTF-8-encoded file.

### Syntax

```c
ESS_FUNC_M EssUpdateFileUTF8ASO (hDestCtx, hSrcCtx, AppName, DbName, FileName, Store, Unlock, ulBufferId);```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hDestCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle of target database on the server</td>
</tr>
<tr>
<td>hSrcCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle for update file location. The update file can reside on the client or on the same server as the target database.</td>
</tr>
<tr>
<td>AppName</td>
<td>ESS_STR_T</td>
<td>Application name for update file location</td>
</tr>
<tr>
<td>DbName</td>
<td>ESS_STR_T</td>
<td>Database name for update file location</td>
</tr>
<tr>
<td>FileName</td>
<td>ESS_STR_T</td>
<td>Name of update specification file</td>
</tr>
<tr>
<td>Store</td>
<td>ESS_BOOL_T</td>
<td>Controls storage of data. If TRUE, data is stored in the server; if FALSE, no data is stored.</td>
</tr>
<tr>
<td>Unlock</td>
<td>ESS_BOOL_T</td>
<td>Not supported for aggregate storage databases. You must always pass ESS_FALSE for this parameter.</td>
</tr>
<tr>
<td>ulBufferId</td>
<td>ESS_ULONG_T</td>
<td>ID number for the data load buffer.</td>
</tr>
</tbody>
</table>
Notes

If the Store flag is set to FALSE, the database merely performs a syntax check of the update specification.

Return Value

Returns zero if successful; otherwise, returns an error code and the records that caused the error.

Access

This function requires the caller to have write privilege (ESS_PRIV_WRITE) to the active database.

Example

See example for EssUpdateFileASO.

See Also

- EssUpdateFileASO
- EssUpdateFileASOEx
- EssUpdateFileEx
- EssUpdateFileUTF8ASOEx
- EssUpdateFileUtf8Ex
- EssUpdateUtf8Ex

EssUpdateFileUTF8ASOEx

Sends an update specification to the active aggregate storage database from a UTF-8-encoded file, capturing any data load errors in ppMbrError.

Syntax

```c
ESS_FUNC_M EssUpdateFileUTF8ASOEx (hDestCtx, hSrcCtx, AppName, DbName, FileName, Store, Unlock, ulBufferId, ppMbrError);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hDestCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle of target database on the server</td>
</tr>
<tr>
<td>hSrcCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle for update file location. The update file can reside on the client or on the same server as the target database.</td>
</tr>
<tr>
<td>AppName</td>
<td>ESS_STR_T</td>
<td>Application name for update file location</td>
</tr>
<tr>
<td>DbName</td>
<td>ESS_STR_T</td>
<td>Database name for update file location</td>
</tr>
<tr>
<td>FileName</td>
<td>ESS_STR_T</td>
<td>Name of update specification file</td>
</tr>
<tr>
<td>Store</td>
<td>ESS_BOOL_T</td>
<td>Controls storage of data. If TRUE, data is stored in the server; if FALSE, no data is stored.</td>
</tr>
<tr>
<td>Unlock</td>
<td>ESS_BOOL_T</td>
<td>Not supported for aggregate storage databases. You must always pass ESS_FALSE for this parameter.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ulBufferId</td>
<td>ESS_UULONG_T</td>
<td>ID number for the data load buffer.</td>
</tr>
<tr>
<td>ppMbrError</td>
<td>ESS_PPMBERR_T</td>
<td>Pointer to linked list of errors contained in ESS_MBRERR_T. Possible errors are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>* AD_MSGDL_ERRORLOAD—Unable to do dataload at Item/Record [number].</td>
</tr>
<tr>
<td></td>
<td></td>
<td>* ESS_MBRERR_BADDATA—Invalid member [membername] in data column.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>* ESS_MBRERR_DBACCESS—You have insufficient access privilege to perform a lock on this database.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>* ESS_MBRERR_DUPLICATE—Duplicate members from the same dimension on data record, [number] records completed.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>* ESS_MBRERR_UNKNOWN—Unknown member [membername] in dataload, [number] records returned.</td>
</tr>
</tbody>
</table>

**Notes**

If the Store flag is set to FALSE, the database merely performs a syntax check of the update specification.

**Return Value**

Returns zero if successful; otherwise, returns an error code and the records that caused the error.

**Access**

This function requires the caller to have write privilege (ESS_PRIV_WRITE) to the active database.

**Example**

See example for EssUpdateFileAso.

**See Also**

- EssUpdateEx
- EssUpdateFileEx
- EssUpdateFileASO
- EssUpdateFileASOEx
- EssUpdateFileUtf8Ex
- EssUpdateFileUTF8ASO
- EssUpdateUtf8Ex

**EssUpdateFileUtf8Ex**

Sends an update specification to the active database from a UTF-8-encoded file, capturing any data load errors in ppMbrError.

**Syntax**

```c
ESS_FUNC_M EssUpdateUtf8Ex (hDestCtx, hSrcCtx, AppName, DbName, FileName, Store, Unlock, ppMbrError);
```
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hDestCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle of the target database on the server.</td>
</tr>
<tr>
<td>hSrcCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle for update file location. The update file can reside on the client or on the same server as the target database.</td>
</tr>
<tr>
<td>AppName</td>
<td>ESS_STR_T</td>
<td>Application name for update file location.</td>
</tr>
<tr>
<td>DbName</td>
<td>ESS_STR_T</td>
<td>Database name for update file location.</td>
</tr>
<tr>
<td>FileName</td>
<td>ESS_STR_T</td>
<td>Name of update specification file.</td>
</tr>
<tr>
<td>Store</td>
<td>ESS_BOOL_T</td>
<td>Controls storage of data. If TRUE, data is stored in the server; if FALSE, no data is stored.</td>
</tr>
<tr>
<td>Unlock</td>
<td>ESS_BOOL_T</td>
<td>Controls unlocking of data blocks. If TRUE, all relevant blocks which are locked will be unlocked (after data is stored, if necessary). If FALSE, no blocks are unlocked.</td>
</tr>
<tr>
<td>ppMbrError</td>
<td>ESS_PPMBERR_T</td>
<td>Pointer to linked list of errors contained in ESS_MBRERR_T. Possible errors are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESS_MBRERR_BADDATA—Invalid member [membername] in data column.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESS_MBRERR_DBACCESS—You have insufficient access privilege to perform a lock on this database.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESS_MBRERR_DUPLICATE—Duplicate members from the same dimension on data record, [number] records completed.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESS_MBRERR_ERRORLOAD—Unable to do dataload at Item/Record [number].</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESS_MBRERR_UNKNOWN—</td>
</tr>
</tbody>
</table>

Notes

• The update data can either be stored in the database, or just verified and any errors returned. Also, any data blocks locked for update can be unlocked by this call.

• If the caller attempts to write data to a member it does not have permission to write to, a warning is generated, and the member is not updated.

• If both the Store and Unlock flags are set to FALSE, the database merely performs a syntax check of the update specification.

Return Value

Returns zero if successful; otherwise, returns an error code and the records that caused the error.

Access

This function requires the caller to have write privilege (ESS_PRIV_WRITE) to the active database.

See Also

• EssUpdateEx
• EssUpdateFileASO
• EssUpdateFileASOEx
• EssUpdateFileEx
• EssUpdateFileUTF8ASOEx
EssUpdateUtf8Ex

Sends an update specification to the active database as a single UTF-8-encoded string.

**Syntax**

```c
ESS_FUNC_M EssUpdateUtf8Ex (hCtx, Store, Unlock, UpdtSpec, ppMbrError);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>Store</td>
<td>ESS_BOOL_T</td>
<td>Controls storage of data. If TRUE, data is stored in the server; if FALSE, no data is stored.</td>
</tr>
<tr>
<td>Unlock</td>
<td>ESS_BOOL_T</td>
<td>Controls unlocking of data blocks. If TRUE, all relevant blocks which are locked will be unlocked (after data is stored, if necessary). If FALSE, no blocks are unlocked.</td>
</tr>
<tr>
<td>UpdtSpec</td>
<td>ESS_STR_T</td>
<td>The update specification, as a single string (must be less than 32 KB).</td>
</tr>
<tr>
<td>ppMbrError</td>
<td>ESS_PPMBRERR_T</td>
<td>Pointer to linked list of errors contained in ESS_MBRERR_T. Possible errors are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_MBRERR_BADDATA—Invalid member [membername] in data column.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_MBRERR_DBACCESS—You have insufficient access privilege to perform a lock on this database.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_MBRERR_DUPLICATE—Duplicate members from the same dimension on data record, [number] records completed.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_MBRERR_ERRORLOAD—Unable to do dataload at Item/Record [number].</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_MBRERR_UNKNOWN—</td>
</tr>
</tbody>
</table>

**Notes**

- The update data can either be stored in the database, or just verified and any errors returned. Also, any data blocks locked for update can be unlocked by this call.
- The update specification string must be less than 32 KB long.
- If the caller attempts to write data to a member it does not have permission to write to, a warning is generated, and the member is not updated.
- If both the Store and Unlock flags are set to FALSE, the database merely performs a syntax check of the update specification.

**Return Value**

Returns zero if successful; otherwise, returns an error code and the records that caused the error.

**Access**

This function requires the caller to have write privilege (ESS_PRIV_WRITE) to the active database.
See Also

- EssUpdateEx
- EssUpdateFileASO
- EssUpdateFileASOEx
- EssUpdateFileEx
- EssUpdateFileUTF8ASOEx
- EssUpdateFileUtf8Ex
- EssUpdateFileUTF8ASO

## EssValidateDB

Checks the database for data integrity.

### Syntax

```c
ESS_FUNC_M EssValidateDB (hCtx, DbName, FileName);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>DbName</td>
<td>ESS_STR_T</td>
<td>Database name. Required, cannot be NULL.</td>
</tr>
<tr>
<td>FileName</td>
<td>ESS_STR_T</td>
<td>Error log file name, to be placed in the app\db directory on the server. Required.</td>
</tr>
</tbody>
</table>

### Notes

- This function runs the validation checks to ensure the integrity of the database.
- Precede this call with a call to EssSetActive().
- This function is asynchronous, so you must continue to call EssGetProcessState() until the validation process is finished.
- This command validates the current database. You must select a database before issuing the EssValidateDB() command.
- EssValidateDB() checks for data integrity in each block. Reading from top to bottom, the validation process goes through the entire database and checks blocks, sections, block type, and block length, and checks for validity in floating point numbers.
- This command writes blocks and information about bad blocks to the log file.
- If this command finds integrity errors, it writes validation process error messages to a text-format log file. The default location for the file is in the application\database directory; for example: %ARBORPATH%\APP\DB\VALIDATE.LST
- The Essbase index contains an index for every data block. For every Read operation, this command automatically compares the index key in the index page with the index key in the corresponding data block and checks other header information in the block. If it encounters a mismatch, EssValidateDB() displays an error message and continues processing until it has checked the entire database.
**Return Value**
None.

**Access**
This function requires the caller to have database designer privilege (ESS_PRIV_DBDESIGN) for the specified database.

**Example**

```c
ESS_VOID_T
ESS_V alidateDB (ESS_HCTX_T hCtx)
{
    ESS_FUNC_M    sts = ESS_STS_NOERR;
    ESS_STR_T     DbName;
    ESS_STR_T     FileName;
    ESS_PROCSTATE_T pState;

    DbName   = "Basic";
    FileName =  "D:\AnalyticServices\app\sample\basic\Validate.lst";
    sts = EssValidateDB (hCtx, DbName, FileName);

    if (!sts)
    {
        sts = EssGetProcessState (hCtx, &pState);
        while (!sts && (pState.State !=
                ESS_STATE_DONE))
            sts = EssGetProcessState (hCtx, &pState);
    }
}
```

**See Also**
- EssSetActive
- EssGetProcessState

**EssValidateHCtx**

Validates a specific context handle (hCtx).

**Syntax**

```c
ESS_FUNC_M EssValidateHCtx (hCtx);
```

**Parameter**  **Data Type**  **Description**

| hCtx     | ESS_HCTX_T | The API context handle to validate |

**Notes**
This function can be used after any extended wait period to ensure the program’s context handle is still recognized by the server.
Return Value

This function returns 0 if the context handle is valid, otherwise it returns an error code to indicate the invalid context handle. Possible reasons for an invalid context handle include the login might have timed out or the user was explicitly logged out by the supervisor.

Access

This function requires no special access.

Example

```c
#include <essapi.h>

char sApplication[] = "accept";
char sDbName[] = "basic";
char sFilename[] = "basic";
char SvrName[] = "local";
char User[] = "test";
char Password[] = "testing";

ESS_HINST_T  hInst;
ESS_HCTX_T hCtx;
FILE *fpOutfile;

void ESS_Init()
{
    ESS_STS_T    sts;
    ESS_INIT_T InitStruct = { ESS_API_VERSION, /* This should be set to ESS_API_VERSION */
                           NULL,             /* void pointer to user's message context */
                           0L,               /* max number of context handles required */
                           255,              /* max size of buffer that can be allocated */
                           NULL,             /* local path to use for file operations */
                           NULL,             /* full path name of message database file */
                           NULL,             /* user-defined memory allocation function */
                           NULL,             /* user-defined memory reallocation function */
                           NULL,             /* user-defined memory free function */
                           NULL,             /* user-defined message callback function */
                           NULL,             /* user-defined help file path */
                           0L                /* reserved for internal use */
                        };

    if ((sts = EssInit(&InitStruct, &hInst)) != ESS_STS_NOERR) {
        fprintf(stdout, "EssInit failure: %ld\n", sts);
        exit ((int) sts);
    } else {
        fprintf(stdout, "EssInit sts: %ld\n", sts);
    }
}
```

668  C Main API Functions
void ESS_Login ()
{
    ESS_STS_T sts = ESS_STS_NOERR;
    ESS_USHORT_T Items;
    ESS_PAPPDB_T pAppsDbs = NULL;

    sts = EssLogin (hInst, SvrName, User, Password, &Items, &pAppsDbs, &hCtx);
    printf("EssLogin sts: %ld\r\n", sts);
}

void ESS_Term()
{
    ESS_STS_T sts = ESS_STS_NOERR;
    if ((sts = EssTerm(hInst)) != ESS_STS_NOERR)
    {
        /* error terminating API */
        exit((ESS_USHORT_T) sts);
    }
    fprintf(stdout, "EssTerm sts: %ld\r\n", sts);
}

void ESS_Logout()
{
    ESS_STS_T sts = ESS_STS_NOERR;
    sts = EssLogout (hCtx);
    fprintf(stdout, "\n\nEssLogout sts: %ld\n",sts);
}

void ESS_SetActive()
{
    ESS_STS_T sts = ESS_STS_NOERR;
    ESS_ACCESS_T Access;
    ESS_STR_T AppName;
    ESS_STR_T DbName;
    AppName = sApplication;
    DbName = sDbName;
    sts = EssSetActive(hCtx, AppName, DbName, &Access);
    fprintf(stdout, "EssSetActive sts: %ld\r\n",sts);
}

/*****************************************************/
/*************** MAIN FUNCTION ***************/
void main(int argc, char ** argv)
{
    ESS_STS_T sts;
    ESS_Init();
    ESS_Login();
    ESS_SetActive();
    /* Do something else, not related to Essbase*/
    sts = EssValidateHCtx (hCtx);
    if (sts) {
        ESS_Login()
        ESS_SetActive();
    }
    /* Do the actual processing now */
    EssClearActive(hCtx);
    ESS_Logout();
EssVerifyFilter

Verifies the syntax of a series of filter row strings against a specified database.

Syntax

```c
ESS_FUNC_M EssVerifyFilter (hCtx,AppName, DbName);
```

Parameter  Data Type     Description
---------  ------------     ------------
hCtx       ESS_HCTX_T     API context handle.
AppName    ESS_STR_T      Application name.
DbName     ESS_STR_T      Database name.

Notes

Follow this call with successive calls to EssVerifyFilterRow() to verify all rows for the filter.

Return Value

None.

Access

This function requires the caller to have database designer privilege (ESS_PRIV_DBDESIGN) for the specified database.

Example

```c
ESS_VOID_T Ess_VerifyFilter (ESS_HCTX_T hCtx)
{
    ESS_FUNC_M stts = ESS_STS_NOERR;
    ESS_STR_T   AppName;
    ESS_STR_T   DbName;
    ESS_USHORT_T Count = 4;
    ESS_STR_T   RowString[4];
    ESS_USHORT_T ind;

    AppName = "Sample";
    DbName = "Basic";
    /* Initialize Filter Row */

    RowString[0] = "@IDESCENDANTS(Scenario)";
    RowString[1] = "@IDESCENDANTS(Product)";
    RowString[2] = "Qtr1, @IDESCENDANTS("Colas")";
```
RowString[3] = "";

/* Verify Filter */
sts = EssVerifyFilter(hCtx, AppName, DbName);

/* Verify Count Filter Rows */
if(!sts)
{
    for (ind = 0; ind < Count; ind++)
        sts = EssVerifyFilterRow(hCtx, RowString[ind]);
}

See Also
- EssGetFilter
- EssVerifyFilterRow

**EssVerifyFilterRow**

Verifies the syntax of a single filter row strings against a specified database.

**Syntax**

```
ESS_FUNC_M EssVerifyFilterRow (hCtx, RowString);
```

**Parameter** | **Data Type** | **Description**
--- | --- | ---

hCtx | ESS_HCTX_T | API context handle.
RowString | ESS_STR_T | Filter row string.

**Notes**

This function should be called repeatedly after calling `EssVerifyFilter()`, once for each row of the filter, terminating the row list with a NULL row string pointer.

**Return Value**

None.

**Access**

This function requires the caller to have database designer privilege (ESS_PRIV_DBDESIGN) for the specified database.

**Example**

See the example of `EssVerifyFilter`.

**See Also**

- EssGetFilter
- EssVerifyFilter
**EssVerifyFormula**

Verifies the syntax of the specified formula. This function is called by **EssOtlVerifyFormula**, which provides more information on returned errors.

**Syntax**

```
ESS_FUNC_M EssVerifyFormula (hCtx, FormulaName);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>FormulaName</td>
<td>ESS_STR_T</td>
<td>The name of the formula to verify.</td>
</tr>
</tbody>
</table>

**Notes**

This function is not meant to be called directly. Instead, use the corresponding Outline API function **EssOtlVerifyFormula**.

**Return Value**

This function returns zero if successful, otherwise it returns an error number.

**See Also**

- **EssOtlVerifyOutline**
- **EssOtlVerifyOutlineEx**
- **EssOtlVerifyFormula**

**EssVerifyRulesFile**

Verifies the syntax of the specified rules file.

**Syntax**

```
ESS_FUNC_M EssVerifyRulesFile (hCtx, ruleFileName, pNmColumns, ppColumnErrors);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>ruleFileName</td>
<td>ESS_STR_T</td>
<td>The name of the rules file to verify.</td>
</tr>
<tr>
<td>pNmColumns</td>
<td>ESS_PULONG_T</td>
<td>Pointer to the number of columns in the rules file.</td>
</tr>
<tr>
<td>ppColumnErrors</td>
<td>ESS_ULONG_T</td>
<td>Pointer to the array of errors found.</td>
</tr>
</tbody>
</table>

**Notes**

- This function requires that a specific database be active; that is, EssSetActive() is required.
- This function is intended to be used after the rules file has been put on the server.
There is one value in the array `ppColumnErrors` for each column in the rules file. The `n`th value in the array corresponds to errors found for the `n`th column in the rules file. Each error value may be zero or more of the following error codes combined with logical OR.

<table>
<thead>
<tr>
<th>Error code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAT_VERIFY_INVALIDMBR</td>
<td>There is an unknown member (or no member) in the field name.</td>
</tr>
<tr>
<td>DAT_VERIFY_INVALIDHDR</td>
<td>There is an unknown member in the header. (This error code applies to the header, and will</td>
</tr>
<tr>
<td>DAT_VERIFY_SAMENAME</td>
<td>This field has the same field name as another field.</td>
</tr>
<tr>
<td>DAT_VERIFY_DIMUSED</td>
<td>The dimension name is used in another field name or in the header.</td>
</tr>
<tr>
<td>DAT_VERIFY_MBRUSED</td>
<td>A member name used as part of a combination in this field is used as a single member name in</td>
</tr>
<tr>
<td>DAT_VERIFY_DIMINCROSSDIM</td>
<td>A dimension name is used in a cross-dimensional reference in the field name.</td>
</tr>
<tr>
<td>DAT_VERIFY_DATAFIELD</td>
<td>Only one field can have the Data Field attribute.</td>
</tr>
<tr>
<td>DAT_VERIFY_SIGNFLIPDIM</td>
<td>The dimension used for Sign Flip checking is not in the associated outline.</td>
</tr>
<tr>
<td>DAT_VERIFY_DUPINHEADER</td>
<td>This field name is also defined in the header definition.</td>
</tr>
<tr>
<td>DAT_VERIFY_DATEANDDATA</td>
<td>A field may be designated a Data Field or Date Field, but not both.</td>
</tr>
<tr>
<td>DAT_VERIFY_DATEFIELDNAME</td>
<td>The field name of a date field must be the name of a date dimension.</td>
</tr>
<tr>
<td>DAT_VERIFY_DATEFORMAT</td>
<td>There is an unrecognized date format for this date column.</td>
</tr>
</tbody>
</table>

Return Value

If successful, returns the number of columns in the rules file as `pNmColumns` and the array of errors found in `ppColumnErrors`.

Access

This function requires no special privileges.

Example

```c
{
    ESS_STS_T      sts = ESS_STS_NOERR;
    ESS_ULONG_T    numColumns = 0, i;
    ESS_PULONG_T   pColumnErrors = NULL;

    sts = EssVerifyRulesFile(hCtx, "rule_file", &numColumns, &pColumnErrors);
    if(!sts)
    {
        if(numColumns && pColumnErrors)
        {
            printf("NumColumns: %d\n", numColumns);
            for(i=0; i<numColumns; i++)
            {
                printf("Column[%d]:\n", i+1);
                if( pColumnErrors[i] == 0 )
                    printf(" No error\n");
                else
                    printf(" Error: %s\n",
```
if( pColumnErrors[i] & DAT_VERIFY_INVALIDMBR )
    printf("     There is an unknown member (or no member) in the field name.
"); 
if( pColumnErrors[i] & DAT_VERIFY_INVALIDHDR )
    printf("     There is an unknown member in the header.
"); 
if( pColumnErrors[i] & DAT_VERIFY_SAMENAME )
    printf("     This field has the same field name as another field.
"); 
if( pColumnErrors[i] & DAT_VERIFY_DIMUSED )
    printf("     The dimension name is used in another field name or in the header.
"); 
if( pColumnErrors[i] & DAT_VERIFY_MBRUSED )
    printf("     A member name used as part of a combination in this field is used as a single member name in another field.
"); 
if( pColumnErrors[i] & DAT_VERIFY_DIMINCROSSDIM )
    printf("     A dimension name is used in a cross-dimensional reference in the field name.
"); 
if( pColumnErrors[i] & DAT_VERIFY_DATAFIELD )
    printf("     Only one field can have the Data Field attribute.
"); 
if( pColumnErrors[i] & DAT_VERIFY_SIGNFLIPDIM )
    printf("     The dimension used for Sign Flip checking is not in the associated outline.
"); 
if( pColumnErrors[i] & DAT_VERIFY_DUPINHEADER )
    printf("     This field name is also defined in the header definition.
"); 
if( pColumnErrors[i] & DAT_VERIFY_DATEANDDATA )
    printf("     A field may be designated a Data Field or a Date Field, but not both.
"); 
if( pColumnErrors[i] & DAT_VERIFY_DATEFIELDNAME )
    printf("     The field name of a date field must be the name of a date dimension.
"); 
if( pColumnErrors[i] & DAT_VERIFY_DATEFORMAT )
    printf("     There is an unrecognized date format for this date column.
"); 
}
EssFree(hInst, pColumnErrors);

See Also

- EssVerifyFormula
- EssOtlVerifyFormula
- EssOtlVerifyOutlineEx

EssWriteToFile

Writes a message to the Essbase Server log file (essbase.log), or to the application log file (appname.log).

Syntax

```c
ESS_FUNC_M EssWriteToFile (hCtx, AgentLog, Message);
```
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle.</td>
</tr>
<tr>
<td>AgentLog</td>
<td>ESS_BOOL_T</td>
<td>If TRUE, message is written to the Essbase Server log file, essbase.log. If FALSE, message is written to the application log file, appname.log.</td>
</tr>
<tr>
<td>Message</td>
<td>ESS_STR_T</td>
<td>Message to be logged to the Essbase Server log file (essbase.log), or to the application log file (appname.log).</td>
</tr>
</tbody>
</table>

**Notes**
- Use EssGetLogFile() to view message logs.
- For the locations of essbase.log and appname.log, see the Oracle Essbase Database Administrator's Guide.

**Return Value**
Returns a zero if successful.

**Access**
The caller must have supervisor privilege (ESS_ACCESS_SUPER) for the specified application.

**Example**
```c
ESS_FUNC_M ESS_WriteToLogFile (ESS_HCTX_T hCtx)
{
    ESS_STR_T     Message = NULL;
    ESS_FUNC_M     sts = ESS_STS_NOERR;

    Message = "Received login request";

    /*
     * Writes the message (Received login request) to the Agent log file.
     */
    sts = EssWriteToLogFile(hCtx, ESS_TRUE, Message);
    return(sts);
}
```

**See Also**
- EssDeleteLogFile
- EssGetLogFile
- EssLogSize
In C Outline API:

- Using the C Outline API
- C Outline API Declarations
- C Outline API Functions
- C Outline API Examples
C Outline API Overview

The Outline API is a set of functions for creating, maintaining, and manipulating Essbase outlines from within a custom application. With the Outline API, you have the same ability to manipulate database outlines from within code as you have using the Outline Editor in Administration Services.

The Outline API is an important part of the Essbase API, with interfaces for C and Visual Basic. The Outline API is used in conjunction with the Essbase API and requires a server connection.

C Outline API Error Handling

Outline API functions return 0 when they succeed; if they fail they return an error status value as defined in esserror.h for C and esberror.bas for Visual Basic. Functions of the main API use the error message callback routine and pass an error number to the message handler. The handler uses the essbase.mdb message database to determine the error message and display an error message to the user.

Outline API functions do not ordinarily use the error message callback routine when returning an error status. The error callback routine is called in the following situations:

- If you call functions that use the network (EsxOtlOpenOutline(), EsxOtlWriteOutline(), and EsxOtlRestructure()), and they incur errors on non-outline related actions.
- If a NULL is found during routine checking when passed into the Outline API, and API_NULL_ARG is returned.
If a bad outline handle (HOUTLINE) is passed into any call requiring an outline handle, and OTLAPI_BAD_HOUTLINE is returned.

C Outline API Server Outline Queries

Several functions support a query interface to the outline API such that the outline does not need to be downloaded from the server and completely read into memory. These Outline API functions support only server outlines. Prior to opening the outline, the user must log in to a server, setting up a valid Essbase login context.

Error handling for these functions is done via the standard API error handling mechanism. Therefore, any message callback that the caller has specified from EsxInit() is called on errors.

Here’s the way it works:

1. The programmer initializes the API as always by calling EsxInit() and EsxLogin().
2. The programmer calls EsxOtlOpenOutlineQuery() to "open" the outline from the server and bring back some initial information. The information brought back from the server is all information in the ESX_OUTLINEINFO_T structure and for each dimension, all relevant information in the ESX_OTLMBR_T internal structure, which includes the ESX_MBRINFO_T structure.
3. The caller needs to get information about members, so he calls EsxOtlQueryMembers() with the appropriate flags to get an array of member handles back. The EsxOtlQueryMembers() call returns all relevant information in the ESX_OTLMBR_T internal structure. The user can then call any of the EsxOtlGetXxx() calls that relate to a specific member by passing in one of the returned member handles. See the comments section in the EsxOtlQueryMembers() call for more information about which calls are supported when the outline is opened in "query" mode.
4. When the caller is done with the data returned from an EsxOtlQueryMembers() call, he should call EsxOtlFreeMembers() or EsbOtlFreeMember() to free the array of members.
5. The caller should call EsxOtlCloseOutline() when complete to clean up internal data structures.
6. The caller terminates the API as always by calling EsxLogout() and EsxTerm().

C Outline API Outline Verification

The Outline API is designed to prevent the caller from creating an illegal outline. To check the outline, use the EsxOtlVerifyOutline() function to verify it before saving it to the server. The Outline API calls EsxOtlVerifyOutline() automatically when an outline is written to the server, if it was not called previously.

Each function call in the Outline API verifies that processing by the caller does not result in an illegal outline. For example, EsxOtlRenameMember() checks a new member name to make sure that it is valid and does not already exist in the outline. Here are a few exceptions to this automatic validation:
- `EsxOtlOpenOutline()` allows the caller to read in a previously created outline that is illegal. This outline could be illegal because the Outline Editor in Administration Services allows you to save an invalid outline to a local file. Any existing errors are detected when `EsxOtlVerifyOutline()` is called. Also, some individual operations are illegal during processing if the outline starts out as illegal.

- `EsxOtlDeleteMember()` and `EsxOtlDeleteDimension()` do not check for any alias combinations that contain a deleted member. `EsxOtlVerifyOutline()` detects this condition.

- `EsxOtlSetMemberFormula()` allows you to enter an illegal formula, and `EsxOtlVerifyOutline()` does not check member formulas. An illegal member formula causes failure during restructure. `EsxGetProcessState()` displays the error message returned from the server.

### C Outline API Memory Allocation

The Essbase API provides a set of memory management functions, `EsxAlloc()`, `EsxRealloc()`, and `EsxFree()`. These functions, plus all internal API memory allocations, call memory allocation routines pointed to by the `AllocFunc`, `ReallocFunc`, and `FreeFunc` fields of the `ESX_INIT_T` initialization structure.

If you are using your own custom memory allocation functions, make sure your memory allocation scheme can handle allocating many small memory buffers.

### C Outline API Security Requirements

Because you can use the Outline API to create, edit, and delete outlines, you must be aware of some security issues when creating an application that uses the Outline API. These issues impact only programs that create, edit, or save outlines during a session.

To manipulate outlines through the Outline Editor in Administration Services, you must have Application Manager or higher privileges. You also need these privileges to use a program that uses the Outline API during execution. If you do not have these privileges, Outline API calls that read or write outlines from the server do not work. See the Oracle Essbase Database Administrator’s Guide for more detailed information on security and privilege levels.

For example, you are writing a new EIS end-user application that allows your users to explore a number of "what-if" situations during a session. To do this, the program dynamically creates a number of Essbase databases during a session. These databases (and their outlines) are temporary and are not saved after the session terminates. You can approach this situation in several ways:

- If you want the user to be able to create an application and multiple databases during a session, give the user the Create/Delete Application privilege. This privilege must be assigned by an Essbase administrator prior to running the program. This is a relatively high privilege level in Essbase, but if the user does not have access to other programs, there is little impact on the overall system security.
If you do not need multiple databases available at the same time, you can have the Essbase administrator create a temporary application and database during the installation of your program. The program itself manipulates the temporary database without having to create a new database for each "what-if" situation.

With the second approach, a user requires only the lower and more restricted Database Manager privilege. You could have the Essbase administrator set up a special group with Database Manager privilege only for your temporary application and database. Users can be assigned to that group. The users would revert to ordinary user privilege for any other access to the system. This approach offers less security exposure, but does require more set up prior to running your program.

**C Outline API Function Call Sequence**

When you use the Outline API, your program must call some API functions before others. Follow this basic call sequence:

1. Call `EsxInit()` before any other API function.
   The API returns an instance handle.
2. Call `EsxLogin()` or `EsxAutoLogin()` to log on to the server.
   The API returns a context handle.
3. Call `EsxOtlOpenOutline()` or `EsxOtlNewOutline()` to open or create an outline.
   The API returns an outline handle.
4. Call `EsxOtlWriteOutline()` to write the current outline to the server.
   `EsxOtlVerifyOutline()` is called automatically by the API before the outline is saved, unless you call it before this.
5. Call `EsxOtlRestructure()` to restructure the database based on the changes made to the outline.
6. Call `EsxUnlockObject()` to unlock the outline object if it is locked when the outline is opened.
7. Call `EsxOtlCloseOutline()` to free resources associated with the outline.
8. Call `EsxLogout()` to log off the server.
   This invalidates the context handle.
9. Call `EsxTerm()` to end the session.
   This invalidates the instance handle.

**Typical C Outline API Task Sequence**

This is a typical order of operations for a simple Outline API application.

1. Create and initialize an `ESX_INIT_T` structure.
2. Initialize the Outline API by calling `EsxInit()`.
3. Allocate any local static or global structures.
4. Log on to the required server by calling `EsxLogin()` or `EsxAutoLogin()`.
5. Create and initialize an `ESX_OUTLINEINFO_T` structure (only for a new outline).
6. Open an existing outline or create a new outline by calling `EsxOtlOpenOutline()` or `EsxOtlNewoutline()`.
7. Work on the outline.
8. Verify the outline by calling `EsxOtlVerifyOutline()`.
9. Write the verified outline to the server by calling `EsxOtlWriteOutline()`.
   The outline is saved with an .OTN extension.
10. Restructure the database by calling `EsxOtlRestructure()`.
    The .OTN file is changed to an .OTL file. This is an asynchronous function call; therefore, you should call `EsxGetProcessState()` until the process is complete.
11. Unlock the outline (if it was locked on opening) by calling `EsxUnlockObject()`.
12. Free all information associated with the outline by calling `EsxOtlCloseOutline()`.
13. Log off the server by calling `EsxLogout()`.
14. Free any local static or global structures.
15. Terminate the API by calling `EsxTerm()`.
C Outline API Error Return Values

Table 7 describes the error status constants returned when an Outline API call fails. These values are defined in the Outline API C language header file `esserror.h`.

For a more complete list, see `esserror.h`.

### Table 7  C Outline API Error Return Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OTLAPI_BAD_ALIASABLE</td>
<td>Illegal alias table</td>
</tr>
<tr>
<td>OTLAPI_BAD_CONSOL</td>
<td>Invalid consolidation type (+,-,etc.)</td>
</tr>
<tr>
<td>OTLAPI_BAD_GENLEVELNAME</td>
<td>Invalid generation or level name</td>
</tr>
<tr>
<td>OTLAPI_BAD_HOUTLINE</td>
<td>Invalid outline handle passed to EssOtl... function</td>
</tr>
<tr>
<td>OTLAPI_BAD_MBRNAME</td>
<td>Invalid member name</td>
</tr>
<tr>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>OTLAPI_BAD_MEMBER</td>
<td>Invalid member handle</td>
</tr>
<tr>
<td>OTLAPI_BAD_MOVE</td>
<td>Illegal move of member. Can't move member to its descendant.</td>
</tr>
<tr>
<td>OTLAPI_BAD_OBJTYPE</td>
<td>Illegal object type</td>
</tr>
<tr>
<td>OTLAPI_BAD_OUTLINETYPE</td>
<td>Invalid outline type</td>
</tr>
<tr>
<td>OTLAPI_BAD_PERSPECTIVE2</td>
<td>Invalid perspective</td>
</tr>
<tr>
<td>OTLAPI_BAD_RENAMESHARE</td>
<td>A shared member cannot be renamed</td>
</tr>
<tr>
<td>OTLAPI_BAD_RESTRUCTTYPE</td>
<td>Invalid restructure type</td>
</tr>
<tr>
<td>OTLAPI_BAD_SCA_VALIDITYSET_TYPE</td>
<td>Perspectives/validity sets do not support this validity set type</td>
</tr>
<tr>
<td>OTLAPI_BAD.SMARTLISTNAME</td>
<td>Invalid text list name</td>
</tr>
<tr>
<td>OTLAPI_BAD_SORT_COMPAREFUNC</td>
<td>Invalid sorting compare function</td>
</tr>
<tr>
<td>OTLAPI_BAD_SORTTYPE</td>
<td>Invalid sort type</td>
</tr>
<tr>
<td>OTLAPI_BAD_TRANSTYPE</td>
<td>Unknown transaction type when creating a transaction (internal error)</td>
</tr>
<tr>
<td>OTLAPI_BAD_USERATTR</td>
<td>Invalid user attribute</td>
</tr>
<tr>
<td>OTLAPI_CUR_NOACCOUNTS</td>
<td>There is no Accounts dimension. You need an Accounts dimension to create a currency database.</td>
</tr>
<tr>
<td>OTLAPI_CUR_NOCOUNTRY</td>
<td>There is no Country dimension. You need a Country dimension to create a currency database.</td>
</tr>
<tr>
<td>OTLAPI_CUR_NOTIME</td>
<td>There is no Time dimension. You need a Time dimension to create a currency database.</td>
</tr>
<tr>
<td>OTLAPI_ERR_ADDDELETEDIMDYNAHICCALC</td>
<td>Member in which to store data is type Dynamic Calc</td>
</tr>
<tr>
<td>OTLAPI_ERR_ADDNAMEUSED</td>
<td>Member name already used (add operation)</td>
</tr>
<tr>
<td>OTLAPI_ERR_ALIASTABLEEXISTS</td>
<td>Alias table already exists</td>
</tr>
<tr>
<td>OTLAPI_ERR_ALIASLANGUAGE_UNAVAILABLE</td>
<td>Alias table languages are unavailable for outline versions before 11.1.2.0.00.</td>
</tr>
<tr>
<td>OTLAPI_ERR_ALIASTABLENAME</td>
<td>Illegal alias table name</td>
</tr>
<tr>
<td>OTLAPI_ERRAlreadyCURRENCY</td>
<td>The outline is a currency outline. You are trying to create a currency outline, and the initial outline is already a currency outline.</td>
</tr>
<tr>
<td>OUTAPI_ERR.ASO_COMPRESSIONMUSTBEDYNAMIC</td>
<td>Aggregate storage outlines require compression dimension to be a single dynamic hierarchy</td>
</tr>
<tr>
<td>OUTAPI_ERR.ASO_DIFFERENTNUMBEROFSHARES</td>
<td>This prototype should have same number of shared members as its previous sibling</td>
</tr>
<tr>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>OUTAPI_ERR_ASO_SHAREDMEMBERSNOTSAMEORDER</td>
<td>This prototype must have each of its shared members as next sibling to its previous sibling’s shared members</td>
</tr>
<tr>
<td>OTLAPI_ERR_ATTR_ATTACHED_WRONGLEVEL</td>
<td>This attribute is attached at wrong level at least once</td>
</tr>
<tr>
<td>OTLAPI_ERR_ATTRMBR_ALREADYASSOCIATED</td>
<td>Base member already associated with an attribute member from the same dimension</td>
</tr>
<tr>
<td>OTLAPI_ERR_BADDIM</td>
<td>Invalid dimension argument</td>
</tr>
<tr>
<td>OTLAPI_ERR_BADHIER</td>
<td>Invalid hierarchy type</td>
</tr>
<tr>
<td>OTLAPI_ERR_BADHIER_TOP</td>
<td>Invalid hierarchy member designation - Member must be at generation 1 or 2</td>
</tr>
<tr>
<td>OTLAPI_ERR_BADSHARE</td>
<td>Illegal share value</td>
</tr>
<tr>
<td>OTLAPI_ERR_BADSKIP</td>
<td>Illegal time balance skip value</td>
</tr>
<tr>
<td>OTLAPI_ERR_BADSTORAGE</td>
<td>Illegal dimension storage value</td>
</tr>
<tr>
<td>OTLAPI_ERR_BADSTORAGECATEGORY</td>
<td>Illegal storage category</td>
</tr>
<tr>
<td>OTLAPI_ERR_BADTIMEBAL</td>
<td>Illegal time balance value</td>
</tr>
<tr>
<td>OTLAPI_ERR_BSO_SOLVEORDER</td>
<td>Block storage outlines that have not been enabled for member types cannot have a solve order</td>
</tr>
<tr>
<td>OTLAPI_ERR_CANTIDENTIFYMBR_DUPLICATEDNAME</td>
<td>Cannot uniquely identify a member because the name is duplicated</td>
</tr>
<tr>
<td>OTLAPI_ERR_CNTS_INDEP_LAST</td>
<td>Independent dimension list must be ordered with continuous independent dimensions last</td>
</tr>
<tr>
<td>OTLAPI_ERR_CONFIGTOOMANYDIMS</td>
<td>Too many dimensions to configure automatically</td>
</tr>
<tr>
<td>OTLAPI_ERR_COPYALIASTABLE</td>
<td>Source and destination tables are the same</td>
</tr>
<tr>
<td>OTLAPI_ERR_CREATETEMP</td>
<td>Cannot create temporary file name. You are probably trying to create it on a read-only drive. We create a temporary file on the client every time you open or write an outline from/to the server.</td>
</tr>
<tr>
<td>OTLAPI_ERR_CURTOOMANYDIMS</td>
<td>Too many dimensions in a currency outline. A currency outline is limited to four dimensions.</td>
</tr>
<tr>
<td>OTLAPI_ERR_DELETEDEFALIAS</td>
<td>Cannot delete the default alias table</td>
</tr>
<tr>
<td>OTLAPI_ERR_DISCRETE_DIFFERENT</td>
<td>An independent range must have the same discrete start and end members</td>
</tr>
<tr>
<td>OTLAPI_ERR_DISCRETE_OR_CNTS</td>
<td>Independent dimension types must be either discrete or continuous</td>
</tr>
<tr>
<td>OTLAPI_ERR_DUP_LANGCODE</td>
<td>The language code is assigned to another alias table within the same database</td>
</tr>
<tr>
<td>OTLAPI_ERR_DUPLICATEALIAS</td>
<td>Duplicate alias</td>
</tr>
<tr>
<td>OTLAPI_ERR_DUPLICATEUSERNAME</td>
<td>Duplicate member name</td>
</tr>
<tr>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------------------</td>
<td>-------------------------------------------------------------------</td>
</tr>
<tr>
<td>OTLAPI_ERR_DUPGENLEVNAME</td>
<td>Cannot add, rename, or set a member name or alias that duplicates a generation or level name</td>
</tr>
<tr>
<td>OTLAPI_ERR_EXPORT_INCORRECT_FLAGS</td>
<td>There are invalid export flags. Export cannot be enabled to limit extraction to the tree and alias table</td>
</tr>
<tr>
<td>OTLAPI_ERR_EXPORT_INVALID_ALIASABLE</td>
<td>An invalid alias table is specified in the export options</td>
</tr>
<tr>
<td>OTLAPI_ERR_EXPORT_INVALID_DIMLIST</td>
<td>The number of dimensions or the dimension list specified in the export options is invalid</td>
</tr>
<tr>
<td>OTLAPI_ERR_EXPORT_INVALIDDIMDIMLIST</td>
<td>The dimension name specified in the export options dimension list is invalid</td>
</tr>
<tr>
<td>OTLAPI_ERR_EXPORT_INVALID_VERSION</td>
<td>This export version is invalid. Enter a valid export version</td>
</tr>
<tr>
<td>OTLAPI_ERR_EXPORT_UNABLE_FILE</td>
<td>Cannot open the file to export the outline</td>
</tr>
<tr>
<td>OTLAPI_ERR_EXPORT_UNABLE_PROCESS</td>
<td>Cannot process the outline because of unsupported outline type</td>
</tr>
<tr>
<td>OTLAPI_ERR_FAILED_GET_ALIASNAMES</td>
<td>Failed to get all alias names due to failed alias identifier lookup</td>
</tr>
<tr>
<td>OTLAPI_ERR_FEATURE_UNAVAILABLE</td>
<td>The feature is unavailable in this outline version; please migrate outline first</td>
</tr>
<tr>
<td>OTLAPI_ERR_FILEIO</td>
<td>Could not read from or write to file</td>
</tr>
<tr>
<td>OTLAPI_ERR_FILEOPEN</td>
<td>Could not open file</td>
</tr>
<tr>
<td>OTLAPI_ERR_FORMATSTRING_MISMATCH</td>
<td>Implied share or label-only member has a different format string than the original member; original member's format string will be applied</td>
</tr>
<tr>
<td>OTLAPI_ERR_FORMATSTRING_NOT_MEMBTYPE_ENABLED</td>
<td>The use of format strings require the outline to be member-type enabled. This outline is not member type enabled</td>
</tr>
<tr>
<td>OTLAPI_ERR_FORMATSTRINGTOOLONG</td>
<td>Format String too long for single locale configuration</td>
</tr>
<tr>
<td>OTLAPI_ERR_FUNCTION_OBSOLETE</td>
<td>Function is obsolete</td>
</tr>
<tr>
<td>OTLAPI_ERR_GENLEVELEXISTS</td>
<td>Generation or level already has a name</td>
</tr>
<tr>
<td>OTLAPI_ERR_GENLEVELNAMEEXISTS</td>
<td>Generation or level name already exists</td>
</tr>
<tr>
<td>OTLAPI_ERR_GENLEVELVALUE</td>
<td>Illegal generation or level value</td>
</tr>
<tr>
<td>OTLAPI_ERR_GENLEVNAMEMBR</td>
<td>Cannot add a generation or level name that duplicates a member name or alias</td>
</tr>
<tr>
<td>OTLAPI_ERR_ILLEGALALIAS_STRING</td>
<td>Illegal member combinational for alias</td>
</tr>
<tr>
<td>OTLAPI_ERR_ILLEGALCOMBOALIAS</td>
<td>Illegal combinational alias name</td>
</tr>
<tr>
<td>OTLAPI_ERR_ILLEGALCURRENCY</td>
<td>Illegal currency member</td>
</tr>
<tr>
<td>OTLAPI_ERR_ILLEGALDEFALIAS</td>
<td>Illegal default alias name</td>
</tr>
<tr>
<td>OTLAPI_ERR_ILLEGALNAME</td>
<td>Illegal member name</td>
</tr>
<tr>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>OTLAPI_ERR_ILLEGALTAG</td>
<td>Illegal dimension tag (category)</td>
</tr>
<tr>
<td>OTLAPI_ERR_ILLEGALOPTION</td>
<td>Occurs when the user passes in an invalid option to <code>EssOtlGetGenNames()</code> or <code>EssOtlGetLevelNames()</code></td>
</tr>
<tr>
<td>OTLAPI_ERR IMPLIED_SHARE_OLD_VERSION</td>
<td>The outline version is too old to set Implied Share</td>
</tr>
<tr>
<td>OTLAPI_ERR INCORRECT_MembERTYPE</td>
<td>Member type can be only set to numeric or date types</td>
</tr>
<tr>
<td>OTLAPI_ERR INVALID_SMARTLIST_HANDLE</td>
<td>Invalid text list handle</td>
</tr>
<tr>
<td>OTLAPI_ERR INVALID_SMARTLIST_IMPORTFILE</td>
<td>Input file for importing text lists is invalid</td>
</tr>
<tr>
<td>OTLAPI_ERR INVALIDID_SMARTLIST_IMPORTFILE</td>
<td>Invalid or duplicate ID in text list import file</td>
</tr>
<tr>
<td>OTLAPI_ERR_LANGCODE_TOOLONG</td>
<td>Alias table language code exceeds the maximum length</td>
</tr>
<tr>
<td>OTLAPI_ERR_LEAFLABEL</td>
<td>Leaf member defined as a label member</td>
</tr>
<tr>
<td>OTLAPI_ERR_MAXALIASTABLES</td>
<td>Maximum number of alias tables has been reached</td>
</tr>
<tr>
<td>OTLAPI_ERR MembERCALC</td>
<td>Illegal member formula</td>
</tr>
<tr>
<td>OTLAPI_ERR_MEMBERTYPE_OFF</td>
<td>Cannot turn off the member type enabled setting of an outline</td>
</tr>
<tr>
<td>OTLAPI_ERR_MBRCOMMENTEXLEN</td>
<td>Extended member comment is too long</td>
</tr>
<tr>
<td>OTLAPI_ERR MISSINGTEXT_SMARTLIST_IMPORTFILE</td>
<td>Missing text for ID in text list import file</td>
</tr>
<tr>
<td>OTLAPI_ERR_MULT_DATE_DIMS</td>
<td>An outline can have at most one dimension with date types on static members</td>
</tr>
<tr>
<td>OTLAPI_ERR_MULTHIER_NOT_ENABLED</td>
<td>Cannot set hierarchy type; multiple hierarchies not enabled for dimension</td>
</tr>
<tr>
<td>OTLAPI_ERR_MULT_SMARTLIST_DIMS</td>
<td>An outline can have at most one dimension with smartlists on static members</td>
</tr>
<tr>
<td>OTLAPI_ERR MUSTSAVE_BEFORE_EDIT</td>
<td>The outline must be saved and re-opened before it can be edited</td>
</tr>
<tr>
<td>OTLAPI_ERR_NOALIAS</td>
<td>No alias for this member</td>
</tr>
<tr>
<td>OTLAPI_ERR_NOALIASCODE</td>
<td>Get/Set alias table language code is not yet implemented</td>
</tr>
<tr>
<td>OTLAPI_ERR_NOALIASASCOMBO</td>
<td>No alias combination</td>
</tr>
<tr>
<td>OTLAPI_ERR_NOATTRONCOMPRESSEDDIM</td>
<td>Attributes are not allowed on the compressed dimension</td>
</tr>
<tr>
<td>OTLAPI_ERR_NOFORMULA</td>
<td>No formula for this member</td>
</tr>
<tr>
<td>OUTAPI_ERR NOMEMBERTYPE</td>
<td>This outline version does not support typed members</td>
</tr>
<tr>
<td>OTLAPI_ERR NOSHAREPROTO</td>
<td>Shared member with no actual member</td>
</tr>
<tr>
<td>OUTAPI_ERR NOSMARTLISTS</td>
<td>This outline version does not support text lists</td>
</tr>
<tr>
<td>OTLAPI_ERR NOTADIM</td>
<td>Dimension name expected</td>
</tr>
<tr>
<td>OTLAPI_ERR NOT_A_TIME_MBR</td>
<td>Invalid argument passed. Not a date-time dimension member</td>
</tr>
<tr>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>-----------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>OTLAPI_ERR_NOT_LINKEDATTRIBUTEDIM</td>
<td>Not a linked attribute dimension handle</td>
</tr>
<tr>
<td>OTLAPI_ERR_NOT_MEMBTYPE_ENABLED</td>
<td>This outline is not member type enabled</td>
</tr>
<tr>
<td>OTLAPI_ERR_NOTIMEDIM</td>
<td>No time dimension defined (can't do time balance operations without a time dimension)</td>
</tr>
<tr>
<td>OTLAPI_ERR_NOTVERIFIED</td>
<td>Outline has errors (when saving to the server)</td>
</tr>
<tr>
<td>OTLAPI_ERR_OBJ_NOTFOUND</td>
<td>Object not found</td>
</tr>
<tr>
<td>OTLAPI_ERR_OBJTYPE_NOTSUPPORTED</td>
<td>Function not supported in server side edit mode</td>
</tr>
<tr>
<td>OTLAPI_ERR_OPENMODE</td>
<td>File was opened in the wrong mode to make this call. If you call EssOtlOpenOutlineQuery() to open the outline, not all of the calls will work.</td>
</tr>
<tr>
<td>OTLAPI_ERR_OTLDATEFORMAT</td>
<td>Invalid outline property: date format</td>
</tr>
<tr>
<td>OTLAPI_ERR_OTLSHARED_FORMAT</td>
<td>Outline member's format string cannot be set for shared members</td>
</tr>
<tr>
<td>OTLAPI_ERR_OTLSHARED_TYPE</td>
<td>Outline member's type cannot be set for shared members</td>
</tr>
<tr>
<td>OTLAPI_ERR_QUERYHINT_INVALIDARRAYSIZE</td>
<td>Invalid query hint array size</td>
</tr>
<tr>
<td>OTLAPI_ERR_RENAMEDEFALIAS</td>
<td>Cannot rename the default alias table</td>
</tr>
<tr>
<td>OTLAPI_ERR_RENAMENAMEUSED</td>
<td>Member name already used (rename operation)</td>
</tr>
<tr>
<td>OTLAPI_ERR_SCA_NOT_ENABLED</td>
<td>This outline is not enabled for varying attributes</td>
</tr>
<tr>
<td>OTLAPI_ERR_SCA_UNAVAILABLE</td>
<td>Varying attributes feature is unavailable in this version</td>
</tr>
<tr>
<td>OTLAPI_ERR_SHAREDMEMBERFORMULA</td>
<td>Shared member cannot have a formula</td>
</tr>
<tr>
<td>OTLAPI_ERR_SHARENOTLEVEL0</td>
<td>Shared member not at level 0 (a shared member cannot be a parent of another member)</td>
</tr>
<tr>
<td>OTLAPI_ERR_SHAREUDA</td>
<td>Cannot set a user attribute for a shared member</td>
</tr>
<tr>
<td>OTLAPI_ERR_SMARTLISTNAMEUSED</td>
<td>Cannot add text list; text list name already used</td>
</tr>
<tr>
<td>OTLAPI_ERR_SMARTLIST_MAPMAXREACHED</td>
<td>Cannot add more than $n$ text list texts to id mappings</td>
</tr>
<tr>
<td>OTLAPI_ERR_SMARTLISTMAXREACHED</td>
<td>Cannot add text list; $n$ maximum text lists supported</td>
</tr>
<tr>
<td>OTLAPI_ERR_SMARTLIST_MISSING</td>
<td>Missing text list association for text-typed member</td>
</tr>
<tr>
<td>OTLAPI_ERR_TBTAGS_WITH_DYN_HIERARCHY</td>
<td>This member has a TB-Tag. That requires TIME dimension to only have STORED hierarchies</td>
</tr>
<tr>
<td>OTLAPI_ERR_TIMESPARSE</td>
<td>Accounts dimension is dense and time dimension sparse-is not used</td>
</tr>
<tr>
<td>OTLAPI_ERR_TYPED_ATTR_LEVEL0</td>
<td>Attribute members and non level-0 aggregate storage members cannot be set to Date or Text type</td>
</tr>
<tr>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>OTLAPI_ERR_TYPED_DIMS</td>
<td>Text typed members, date typed members, and stored members with format strings should be specified along the same dimension</td>
</tr>
<tr>
<td>OTLAPI_ERR_UNKNOWNOPTSMBR</td>
<td>Unknown DTS member</td>
</tr>
<tr>
<td>OTLAPI_ERR_VALIDITYSET_MATCH</td>
<td>The validity set must match an existing set in the outline</td>
</tr>
<tr>
<td>OTLAPI_ERR_VIRTLEVONOFORMULA</td>
<td>Dynamic Calc members must have formulas or children, or else they cannot be calculated</td>
</tr>
<tr>
<td>OTLAPI_ERR_VIRTBADPARENT</td>
<td>When a single child member is Dynamic Calc or Dynamic Calc and Store, the parent must also be Dynamic Calc or Dynamic Calc and Store</td>
</tr>
<tr>
<td>OTLAPI_ERR_VIRTTOOMANYCHILDREN</td>
<td>Dynamic Calc member has more than 100 children</td>
</tr>
<tr>
<td>OTLAPI_FAILED_ASSIGN_DEFAULTGENNAMES</td>
<td>Failed to set time related generation names for the date-time dimension created</td>
</tr>
<tr>
<td>OTLAPI_ILLEGAL_SCA_TYPE_2</td>
<td>Varying attribute outlines do not allow duplicate names, and cannot be a currency outline</td>
</tr>
<tr>
<td>OTLAPI_INVALID_ARG</td>
<td>Invalid argument passed to ESSOTL function</td>
</tr>
<tr>
<td>OTLAPI_INVALID_QUERYID</td>
<td>Invalid query id argument passed</td>
</tr>
<tr>
<td>OTLAPI_INVALID_QUERY_OPTIONS</td>
<td>Invalid query options passed. Will be ignored</td>
</tr>
<tr>
<td>OTLAPI_NO_GENLEVELNAME</td>
<td>Cannot find generation or level name</td>
</tr>
<tr>
<td>OTLAPI_NO_USERATTR</td>
<td>Cannot find user attribute</td>
</tr>
<tr>
<td>OTLAPI_NULL_ARG</td>
<td>NULL argument passed to EssOtl... function</td>
</tr>
<tr>
<td>OTLAPI_OUTLINE_TOO_NEW</td>
<td>Outline is of a newer version than this program can understand</td>
</tr>
<tr>
<td>OTLAPI_SORT_TOOMANY</td>
<td>Too many members to sort (64K / 4 members is the maximum sorting capacity)</td>
</tr>
<tr>
<td>OTLAPI_SMARTLIST_ASSOC_EXISTS</td>
<td>Cannot delete a text list with existing associations</td>
</tr>
<tr>
<td>OTLAPI_SMARTLIST_DUP_IDORNAME</td>
<td>Duplicate text list element ID or name</td>
</tr>
<tr>
<td>OTLAPI_SMARTLIST_INVALID_TEXT</td>
<td>Invalid text list text</td>
</tr>
<tr>
<td>OTLAPI_WRONG_INDEPDIM_NM</td>
<td>The number of independent dimensions given in perspective does not match the outline</td>
</tr>
</tbody>
</table>

**C Outline API DTS Member Structures**

These structures contain information about Dynamic Time Series (DTS) members.

```c
/*
 * ESS_DTSMBRNAME_T, ESS_PDTSMBRNAME_T
 * DTS member name structure
 */
```
Data Type | Field       | Description                      
----------|------------|----------------------------------
ESS_DTSMBRNAME_T | szDTSMember | The name of the DTS member.       
ESS_MBRNAMELEN  | szName     | The length of the DTS member name.  

C Outline API Symbolic Constant Definitions

This section describes the symbolic constants used by the Outline API. These constants are defined in the Essbase Outline API C language header file essotl.h:

- “Account Member Currency Conversion Category Values” on page 693
- “Account Member Time Balance Skip Values” on page 693
- “Account Member Time Balance Values” on page 693
- “Dimension Categories” on page 693
- “Dimension Categories (Tags)” on page 694
- “Generation and Level Options” on page 695
- “Query Types” on page 695
- “Query Options” on page 695
- “Restructure Values” on page 696
- “Share Constants” on page 696
- “Sorting Options” on page 697
## Account Member Currency Conversion Category Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_CONV_NONE</td>
<td>Default conversion category. Member inherits category from parent.</td>
</tr>
<tr>
<td>ESS_CONV_CATEGORY</td>
<td>Define a Currency Conversion category for this member</td>
</tr>
<tr>
<td>ESS_CONV_NOCONV</td>
<td>No conversion for this member</td>
</tr>
</tbody>
</table>

## Account Member Time Balance Skip Values

Only valid if time balance is not ESS_TIMEBAL_NONE.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_SKIP_NONE</td>
<td>Don't skip anything</td>
</tr>
<tr>
<td>ESS_SKIP_MISSING</td>
<td>Skip the value if the data is #missing</td>
</tr>
<tr>
<td>ESS_SKIP_ZEROS</td>
<td>Skip the value if the data is 0</td>
</tr>
<tr>
<td>ESS_SKIP_BOTH</td>
<td>Skip the value if the data is #missing or 0</td>
</tr>
</tbody>
</table>

## Account Member Time Balance Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_TIMEBAL_NONE</td>
<td>No time balance</td>
</tr>
<tr>
<td>ESS_TIMEBAL_FIRST</td>
<td>First time balance member</td>
</tr>
<tr>
<td>ESS_TIMEBAL_LAST</td>
<td>Last time balance member</td>
</tr>
<tr>
<td>ESS_TIMEBAL_AVG</td>
<td>Average time balance member</td>
</tr>
</tbody>
</table>

## Dimension Categories

Used for optimizing storage when using storage auto-configure

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_STORECAT_ACCOUNTS</td>
<td>Accounts storage category</td>
</tr>
<tr>
<td>ESS_STORECAT_ATTRCALC</td>
<td>Attribute calculation (aggregation) storage category</td>
</tr>
<tr>
<td>ESS_STORECAT_ATTRIBUTE</td>
<td>Attribute storage category</td>
</tr>
<tr>
<td>ESS_STORECAT_BUSUNIT</td>
<td>Business Unit storage category</td>
</tr>
<tr>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>ESS_STORECAT_CUSTOMER</td>
<td>Customer storage category</td>
</tr>
<tr>
<td>ESS_STORECAT_DIST</td>
<td>Distribution Channel storage category</td>
</tr>
<tr>
<td>ESS_STORECAT_GEOG</td>
<td>Geographical Location storage category</td>
</tr>
<tr>
<td>ESS_STORECAT_MARKET</td>
<td>Market storage category</td>
</tr>
<tr>
<td>ESSSTORECAT орган</td>
<td>Organization storage category</td>
</tr>
<tr>
<td>ESS_STORECAT_OTHER</td>
<td>None or don't know storage category</td>
</tr>
<tr>
<td>ESS_STORECAT_PRODUCT</td>
<td>Product storage category</td>
</tr>
<tr>
<td>ESS_STORECAT_SCENARIO</td>
<td>Scenario storage category</td>
</tr>
<tr>
<td>ESS_STORECAT_TIME</td>
<td>Time storage category</td>
</tr>
<tr>
<td>ESS_STORECAT_UNITS</td>
<td>Units storage category</td>
</tr>
</tbody>
</table>

**Dimension Categories (Tags)**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_CAT_ACCOUNTS</td>
<td>Accounts dimension</td>
</tr>
<tr>
<td>ESS_CAT_ATTRCALC</td>
<td>Attribute calculation dimension or member. Used internally for aggregation.</td>
</tr>
<tr>
<td>ESS_CAT_ATTRIBUTE</td>
<td>Attribute dimension or member</td>
</tr>
<tr>
<td>ESS_CAT_COUNTRY</td>
<td>Country dimension</td>
</tr>
<tr>
<td>ESS_CAT_CURPARTITION</td>
<td>Currency partition dimension. Valid only in non-currency databases.</td>
</tr>
<tr>
<td>ESS_CAT_NONE</td>
<td>No category</td>
</tr>
<tr>
<td>ESS_CAT_TIME</td>
<td>Time dimension</td>
</tr>
<tr>
<td>ESS_CAT_TYPE</td>
<td>Type dimension. Valid only in currency databases.</td>
</tr>
</tbody>
</table>

**Member Types**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_MEMBERTYPE_NONE</td>
<td>No type</td>
</tr>
<tr>
<td>ESS_MEMBERTYPE_NUMERIC</td>
<td>Numeric type</td>
</tr>
<tr>
<td>ESS_MEMBERTYPE_SMARTLIST</td>
<td>Text List (SmartList) type</td>
</tr>
</tbody>
</table>
### Generation and Level Options

You can use with `EssOtlGetGenNames()` and `EssOtlGetLevelNames()`

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_GENLEV_ALL</td>
<td>Returns default and user-defined names</td>
</tr>
<tr>
<td>ESS_GENLEV_ACTUAL</td>
<td>Returns only names that are user-defined</td>
</tr>
<tr>
<td>ESS_GENLEV_DEFAULT</td>
<td>Returns all default names, including the default names for generations and levels that also have user-defined names</td>
</tr>
<tr>
<td>ESS_GENLEV_NOACTUAL</td>
<td>Returns all default names, excluding the default names for generations and levels that also have user-defined names</td>
</tr>
</tbody>
</table>

### Query Options

You can specify for certain query types in “ESS_PREDICATE_T” on page 709

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_MEMBERSONLY</td>
<td>Valid for ESS_SEARCH, ESS_WILDSEARCH</td>
</tr>
<tr>
<td>ESS_ALIASESONLY</td>
<td>Valid for ESS_SEARCH, ESS_WILDSEARCH</td>
</tr>
<tr>
<td>ESS_MEMBERSANDALIASES</td>
<td>Valid for ESS_SEARCH, ESS_WILDSEARCH</td>
</tr>
<tr>
<td>ESS_COUNTONLY</td>
<td>Valid for any query type. Queries the outline without returning any data. Returns a count of how many members meet the query type by filling in the ulTotalCount field in “ESS_MBRCOUNTS_T” on page 699.</td>
</tr>
<tr>
<td>ESS_INCLUDEHYBRIDANALYSIS</td>
<td>Includes relational sources if present.</td>
</tr>
<tr>
<td>ESS_EXCLUDEHYBRIDANALYSIS</td>
<td>Excludes relational sources if present.</td>
</tr>
</tbody>
</table>

### Query Types

Used for defining the operation to perform in “ESS_PREDICATE_T” on page 709:

- ESS_CHILDREN
- ESS_DESCENDANTS
- ESS_BOTTOMLEVEL
- ESS_SIBLINGS
- ESSSAMELEVEL
- ESSSAMEGENERATION
- ESSPARENT
- ESSDIMENSION
- ESSNAMEDGENERATION
- ESSNAMEDLEVEL
- ESSSEARCH
- ESWILDSEARCH
- ESSUSERATTRIBUTE
- ESSANCESTORS
- ESSDTSMEMBERS
- ESSDIMUSERATTRIBUTES
- ESSINDEPDIMS
- ESSSIBLINGS65
- ESSINDEPDIMS_DISCRETE
- ESSINDEPDIMS_CONTINUOUS

**Restructure Values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_DOR_ALLDATA</td>
<td>Keep all data</td>
</tr>
<tr>
<td>ESS_DOR_NODATA</td>
<td>Discard all data</td>
</tr>
<tr>
<td>ESS_DOR_LOWDATA</td>
<td>Keep only level 0 data</td>
</tr>
<tr>
<td>ESS_DOR_INDATA</td>
<td>Keep only input data</td>
</tr>
<tr>
<td>ESS_DOR_FORCE_ALLDATA</td>
<td>Reload all data</td>
</tr>
</tbody>
</table>

**Share Constants**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_SHARE_DYNCALCNOSTORE</td>
<td>Shared member. A member tagged as no Dynamic Calc and Store.</td>
</tr>
<tr>
<td>ESS_SHARE_DYNCALCSTORE</td>
<td>Shared member. A member tagged as Dynamic Calc and Store.</td>
</tr>
<tr>
<td>ESS_SHARE_DATA</td>
<td>Normal member (default value)</td>
</tr>
<tr>
<td>ESS_SHARE_LABEL</td>
<td>Label member. Do not store data for this member.</td>
</tr>
</tbody>
</table>
### Sorting Options

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_SORT_ASCENDING</td>
<td>Sort in ascending order</td>
</tr>
<tr>
<td>ESS_SORT_DESCENDING</td>
<td>Sort in descending order</td>
</tr>
<tr>
<td>ESS_SORT_USERDEFINED</td>
<td>User supplies a custom sorting routine</td>
</tr>
</tbody>
</table>

### ESS_ATTRIBUTEQUERY_T

Used by `EssOtlQueryAttributes` for complex queries concerning attributes.

typedef struct ESS_ATTRIBUTEQUERY_T
{
    ESS_BOOL_T bInputMemberIsHandle;
    union
    {
        ESS_HMEMBER_T hMember;
        ESS_STR_T szMember;
    } uInputMember;
    ESS_USHORT_T usInputMemberType;
    ESS_USHORT_T usOutputMemberType;
    ESS_ATTRIBUTEVALUE_T Attribute;
    ESS_USHORT_T usOperation;
} ESS_ATTRIBUTEQUERY_T, *ESS_PATTRIBUTEQUERY_T, **ESS_PPATTRIBUTEQUERY_T;

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_BOOL_T</td>
<td><code>bInputMemberIsHandle</code></td>
<td>Boolean value:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• TRUE: attribute query by member handle</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• FALSE: attribute query by member name string</td>
</tr>
<tr>
<td>ESS_HMEMBER_T</td>
<td><code>hInputMember</code></td>
<td>A union variable for the following member reference values:</td>
</tr>
<tr>
<td>ESS_STR_T</td>
<td><code>szInputMember</code></td>
<td>• Member handle</td>
</tr>
<tr>
<td></td>
<td><code>hInputMember</code></td>
<td>• Member name string</td>
</tr>
<tr>
<td>Data Type</td>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ESS_USHORT_T</td>
<td>usInputMemberType</td>
<td>A constant identifier indicating the data type of the member queried:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_ATTRIBUTE_DIMENSION</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_ATTRIBUTE_MEMBER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_STANDARD_DIMENSION</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_STANDARD_MEMBER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_BASE_DIMENSION</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_BASE_MEMBER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_ATTRIBUTED_MEMBER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>See Table 6, “C API Attributes Terminology,” on page 102.</td>
</tr>
<tr>
<td>ESS_USHORT_T</td>
<td>usOutputMemberType</td>
<td>A constant identifier indicating the data type of the member returned:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_ATTRIBUTE_DIMENSION</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_ATTRIBUTE_MEMBER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_STANDARD_DIMENSION</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_STANDARD_MEMBER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_BASE_DIMENSION</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_BASE_MEMBER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_ATTRIBUTED_MEMBER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_INVALID_MEMBER</td>
</tr>
<tr>
<td>“ESS_ATTRIBUTEVALUE_T” on page 118</td>
<td>Attribute</td>
<td>A structure defining the attribute value for query input</td>
</tr>
<tr>
<td>ESS_USHORT_T</td>
<td>usOperation</td>
<td>A constant identifier indicating the type of query operation:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_EQ: equal to</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_NEQ: not equal to</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_GT: greater than</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_LT: less than</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_GTE: greater than or equal to</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_LTE: less than or equal to</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_TYPEOF</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_ALL</td>
</tr>
</tbody>
</table>

**ESS_GENLEVELNAME_T**

Contains information about generation and level names.

typedef struct ESS_GENLEVELNAME_T
{
  ESS_USHORT_T  usNumber;
  ESS_MBRNAME_T szName;
} ESS_GENLEVELNAME_T, *ESS_PGENLEVELNAME_T, **ESS_PPGENLEVELNAME_T;
<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_USHORT_T</td>
<td>usNumber</td>
<td>Generation or level number.</td>
</tr>
<tr>
<td>ESS_MBRNAME_T</td>
<td>szName</td>
<td>Generation or level name.</td>
</tr>
</tbody>
</table>

**ESS_GENLEVELNAMEEX_T**

Contains information about generation and level names.

typedef struct ESS_GENLEVELNAMEEX_T
{
    ESS_USHORT_T  usNumber;
    ESS_BOOL_T    bNameUnique;
    ESS_MBRNAME_T szName;
} ESS_GENLEVELNAMEEX_T, *ESS_PGENLEVELNAMEEX_T, **ESS_PPGENLEVELNAMEEX_T;

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_USHORT_T</td>
<td>usNumber</td>
<td>Generation or level number.</td>
</tr>
<tr>
<td>ESS_MBRNAME_T</td>
<td>szName</td>
<td>Generation or level name.</td>
</tr>
</tbody>
</table>

**ESS_MBRCOUNTS_T**

Contains information about member counts for queries.

typedef struct ESS_MBRCOUNTS_T
{
    ESS_ULONG_T    ulStart;
    ESS_ULONG_T    ulMaxCount;
    ESS_ULONG_T    ulTotalCount;
    ESS_ULONG_T    ulReturnCount;
} ESS_MBRCOUNTS_T, *ESS_PMBRCOUNTS_T, **ESS_PMBRCOUNTS_T;

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_ULONG_T</td>
<td>ulStart</td>
<td>Starting member for retrieval of information.</td>
</tr>
<tr>
<td>ESS_ULONG_T</td>
<td>ulMaxCount</td>
<td>Maximum number of members to retrieve.</td>
</tr>
<tr>
<td>ESS_ULONG_T</td>
<td>ulTotalCount</td>
<td>Return of the total count of members that exist in the results of the query. This could be more than ulMaxCount.</td>
</tr>
<tr>
<td>ESS_ULONG_T</td>
<td>ulReturnCount</td>
<td>Return count of returned member handles. This should never be more than ulMaxCount.</td>
</tr>
</tbody>
</table>

**ESS_MBRINFO_T**

Contains information about an outline member.
typedef struct ESS_MBRINFO_T {
    ESS_MBRNAME_T     szMember;
    ESS_USHORT_T      usLevel;
    ESS_USHORT_T      usGen;
    ESS_USHORT_T      usConsolidation;
    ESS_BOOL_T        fTwoPass;
    ESS_BOOL_T        fExpense;
    ESS_USHORT_T      usConversion;
    ESS_MBRNAME_T     szCurMember;
    ESS_USHORT_T      usTimeBalance;
    ESS_USHORT_T      usSkip;
    ESS_USHORT_T      usShare;
    ESS_USHORT_T      usStorage;
    ESS_USHORT_T      usCategory;
    ESS_USHORT_T      usStorageCategory;
    ESS_MBRCOMMENT_T     szComment;
    ESS_ULONG_T       ulChildCount;
    ESS_MBRNAME_T     szDimName;
    ESS_BOOL_T        fAttributed;
    ESS_ATTRIBUTEVALUE_T Attribute;
    ESS_BOOL_T        fHasRelDesc;
    ESS_BOOL_T        fHasHAEnabled;
    ESS_PVOID_T      pLastSibling;
    ESS_ULONG_T       ulSiblingCount;
    ESS_BOOL_T        fFormula;
    ESS_BOOL_T        fUda;
    ESS_BOOL_T        fAlias;
    ESS_BOOL_T        fIndependentDim;
    ESS_UCHAR_T      ucHierarchyType;
    ESS_UCHAR_T      ucDimSolveOrder;
    ESS_UCHAR_T      ucSolveOrder;
    ESS_BOOL_T        fNonUniqueName;
    ESS_BOOL_T        fFlow;
} ESS_MBRINFO_T, *ESS_PMBRINFO_T, **ESS_PPMBRINFO_T;

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_MBRNAME_T</td>
<td>szMember</td>
<td>Member name. This field can be set only by the caller when creating the member.</td>
</tr>
<tr>
<td>ESS_USHORT_T</td>
<td>usLevel</td>
<td>Level of the member in the outline. This field cannot be modified.</td>
</tr>
<tr>
<td>ESS_USHORT_T</td>
<td>usGen</td>
<td>Generation of the member in the outline. This field cannot be modified.</td>
</tr>
<tr>
<td>ESS_USHORT_T</td>
<td>usConsolidation</td>
<td>Unary consolidation type. It can be one of the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESS_UCALC_ADD</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESS_UCALC_SUB</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESS_UCALC_MULT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESS_UCALC_DIV</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESS_UCALC_PERCENT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESS_UCALC_NOOP</td>
</tr>
<tr>
<td>ESS_BOOL_T</td>
<td>fTwoPass</td>
<td>ESS_TRUE if two-pass calculation member.</td>
</tr>
<tr>
<td>Data Type</td>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ESS_BOOL_T</td>
<td>fExpense</td>
<td>ESS_TRUE if expense member.</td>
</tr>
<tr>
<td>ESS_USHORT_T</td>
<td>usConversion</td>
<td>Currency Conversion type. This is valid only for members of the Accounts</td>
</tr>
<tr>
<td></td>
<td></td>
<td>dimension. It can be one of the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_CONV_NONE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_CONV_CATEGORY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_CONV_NOCONV</td>
</tr>
<tr>
<td>ESS_MBRNAME_T</td>
<td>szCurMember</td>
<td>If member is of the Accounts dimension and usConversion is ESS_CONV_</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CATEGORY. This field defines the currency category. If member is of</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the Country dimension. This field defines the currency name. This field</td>
</tr>
<tr>
<td></td>
<td></td>
<td>is undefined in all other situations.</td>
</tr>
<tr>
<td>ESS_USHORT_T</td>
<td>usTimeBalance</td>
<td>Time balance option. Valid field only for members of the Accounts dimension.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>It can be one of the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_TIMEBAL_NONE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_TIMEBAL_FIRST</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_TIMEBAL_LAST</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_TIMEBAL_AVG</td>
</tr>
<tr>
<td>ESS_USHORT_T</td>
<td>usSkip</td>
<td>Time balance skip option. Valid field only for members of the Accounts</td>
</tr>
<tr>
<td></td>
<td></td>
<td>dimension if usTimeBalance is not equal to ESS_TIMEBAL_NONE. It can be</td>
</tr>
<tr>
<td></td>
<td></td>
<td>one of the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_SKIP_NONE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_SKIP_MISSING</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_SKIP_ZEROS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_SKIP_BOTH</td>
</tr>
<tr>
<td>ESS_USHORT_T</td>
<td>usShare</td>
<td>Share option. It can be one of the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_SHARE_DATA (default value)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_SHARE_DYNCALCSTORE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_SHARE_DYNCALCNOSTORE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_SHARE_LABEL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_SHARE_NEVER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_SHARE_SHARE (Valid for level 0 members only)</td>
</tr>
<tr>
<td>ESS_USHORT_T</td>
<td>usStorage</td>
<td>Dimension storage type. This field is valid only for dimension members and</td>
</tr>
<tr>
<td></td>
<td></td>
<td>can be one of the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_DIMTYPE_DENSE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_DIMTYPE_SPARSE</td>
</tr>
<tr>
<td>Data Type</td>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| ESS_USHORT_T  | usCategory     | Dimension category. This field is valid only for dimensions and attribute members. It can be one of the following:  
|               |                | ● ESS_CAT_ACCOUNTS  
|               |                | ● ESS_CAT_ATTRCALC (for internal use only)  
|               |                | ● ESS_CAT_ATTRIBUTE  
|               |                | ● ESS_CAT_COUNTRY  
|               |                | ● ESS_CAT_CURPARTITION (for non-currency databases only)  
|               |                | ● ESS_CAT_NONE  
|               |                | ● ESS_CAT_TIME  
|               |                | ● ESS_CAT_TYPE (for currency databases only)  
| ESS_USHORT_T  | usStorageCategory | Dimension storage category. This field is valid only for dimensions and attribute members. Optimizes the storage types of dimensions when the outline is configured for automatic optimization. It can be one of the following:  
|               |                | ● ESS_STORECAT_ACCOUNTS  
|               |                | ● ESS_STORECAT_ATTRCALC (for internal use only)  
|               |                | ● ESS_STORECAT_ATTRIBUTE  
|               |                | ● ESS_STORECAT_BUSUNIT  
|               |                | ● ESS_STORECAT_CUSTOMER  
|               |                | ● ESS_STORECAT_DIST  
|               |                | ● ESS_STORECAT_GEOG  
|               |                | ● ESS_STORECAT_MARKET  
|               |                | ● ESS_STORECAT_ORGAN  
|               |                | ● ESS_STORECAT_OTHER  
|               |                | ● ESS_STORECAT_PRODUCT  
|               |                | ● ESS_STORECAT_SCENARIO  
|               |                | ● ESS_STORECAT_TIME  
|               |                | ● ESS_STORECAT_UNITS  
| ESS_MBRCOMMENT_T | szComment   | Member comment array  
| ESS_ULONG_T   | ulChildCount   | This field contains the total number of children of the member specified in ESS_MBRNAME_T.  
| ESS_MBRNAME_T | szDimName      | Attribute dimension name  
| ESS_BOOL_T    | fAttributed    | Indicates whether the member has attributes associated with it. Values: ESS_TRUE and ESS_FALSE.  
| "ESS_ATTRIBUTEVALUE_T" on page 118 | Attribute | Attribute value  
| ESS_BOOL_T    | fHasRelDesc    | The member has relational descendants.  
| ESS_BOOL_T    | fHasHAEenabled | The dimension has relational partitions enabled. Valid only for Dimension members.  


<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSS_PVOID_T</td>
<td>pLastSibling</td>
<td>Last sibling pointer</td>
</tr>
<tr>
<td>ESS_UULONG_T</td>
<td>uSiblingCount</td>
<td>Sibling count</td>
</tr>
<tr>
<td>ESS_BOOL_T</td>
<td>fFormula</td>
<td>Indicates whether has a formula</td>
</tr>
<tr>
<td>ESS_BOOL_T</td>
<td>fUda</td>
<td>Indicates whether has UDA</td>
</tr>
<tr>
<td>ESS_BOOL_T</td>
<td>fAlias</td>
<td>Indicates whether has alias</td>
</tr>
<tr>
<td>ESS_BOOL_T</td>
<td>fIndependentDim</td>
<td>For dimensions on varying attribute outlines; indicates if an independent dimension</td>
</tr>
<tr>
<td>ESS_UCHAR_T</td>
<td>ucHierarchyType</td>
<td>Defines the type of hierarchy based on the generation. If the member is generation 1, then:</td>
</tr>
<tr>
<td>ESS_UCHAR_T</td>
<td>udDimSolveOrder</td>
<td>Defines the solve order for the dimension.</td>
</tr>
<tr>
<td>ESS_UCHAR_T</td>
<td>udSolveOrder</td>
<td>Indicates the solve order value. The solve order can be 0-127.</td>
</tr>
<tr>
<td>ESS_BOOL_T</td>
<td>fNonUniqueName</td>
<td>Indicates whether the member name is unique</td>
</tr>
<tr>
<td>ESS_BOOL_T</td>
<td>fFlow</td>
<td>Indicates that member is type Flow</td>
</tr>
</tbody>
</table>

**ESS_OTLQUERYERRORLIST_T**

Stores a list of errors encountered during an extended member query; that is, while calling `EssOtlQueryMembersEx`.

typedef struct ESS_OTLQUERYERRORLIST_T
{
    ESS_UULONG_T ulCount;
    ESS_OTLQUERYERROR_T* ErrorArray;
} ESS_OTLQUERYERRORLIST_T, *ESS_POTLQUERYERRORLIST_T, **ESS_PPOTLQUERYERRORLIST_T;

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_UULONG_T</td>
<td>ulCount</td>
<td>Number of errors returned during a query</td>
</tr>
<tr>
<td>ESS_OTLQUERYERROR_T*</td>
<td>ErrorArray</td>
<td>Pointer to an array of errors returned during a query</td>
</tr>
</tbody>
</table>
**ESS_OUTERROR_T**

Returns the errors for each member when verifying an outline. The errors are bit field values returned in a 32-bit status word. Each error value corresponds to a function call error return value described in Table 7, “C Outline API Error Return Values,” on page 685.

```c
typedef struct ESS_OUTERROR_T
{
    ESS_HMEMBER_T    hMember;
    ESS_ULONG_T      ulErrors;
} ESS_OUTERROR_T, *ESS_POUTERROR_T, **ESS_PPOUTERROR_T;
```

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_HMEMBER_T</td>
<td>hMember</td>
<td>Handle to member with errors.</td>
</tr>
<tr>
<td>ESS_ULONG_T</td>
<td>ulErrors</td>
<td>Bitmask of errors for the member. See Values for ulErrors.</td>
</tr>
</tbody>
</table>

### Values for ulErrors

The following are possible values for `ulErrors`:

- `ESS_OUTERROR_ALIAS zobaczyć`<br>
- `ESS_OUTERROR3_ASO_BAD_AGGREGATION_OPERATOR`<br>
- `ESS_OUTERROR3_ASO_BAD_NONLEAFMBR`<br>
- `ESS_OUTERROR3_ASO_DYNASSOCD`<br>
- `ESS_OUTERROR3_ASO_EITHERLABELORFORMULA`<br>
- `ESS_OUTERROR3_ASO_INVALID_AGGLEVELUSAGE`<br>
- `ESS_OUTERROR3_ASO_INVALIDATTRCALC`<br>
- `ESS_OUTERROR3_ASO_ISDUPLICATESHAREINHIERARCHY`<br>
- `ESS_OUTERROR3_ASO_LABEL_SPAN`<br>
- `ESS_OUTERROR3_ASO_LEVELPRODUCT_TOO_LARGE`<br>
- `ESS_OUTERROR3_ASO_NOATTRIBUTE_ON_ACCOUNTS`<br>
- `ESS_OUTERROR3_ASO_NOFORMULA`<br>
- `ESS_OUTERROR4_ASO_PROTOLEVELZERO`<br>
- `ESS_OUTERROR3_ASO_SHAREDMBR`<br>
- `ESS_OUTERROR3_ASO_TWOCHILDRENFORTHISOPER`<br>
- `ESS_OUTERROR3_ASO_WHOLEACCOUNTSDIMVIRTUAL`<br>
- `ESS_OUTERROR2_ATTRCALCABSENT`<br>
- `ESS_OUTERROR2_ATTRDIMNOTASSOCIATED`<br>
- `ESS_OUTERROR_BADATTRIBUTECODE`<br>
- `ESS_OUTERROR_BADCATEGORY`<br>
- `ESS_OUTERROR_BADSHARE`
- ESS_OUTERROR_BADSKIP
- ESS_OUTERROR_BADSTORAGE
- ESS_OUTERROR_BADSTORAGECATEGORY
- ESS_OUTERROR_BADTIMEBAL
- ESS_OUTERROR2_BOOLEANNAMESETTING
- ESS_OUTERROR2_CHILDDCOUNT
- ESS_OUTERROR_CURTOOMANYDIMS
- ESS_OUTERROR2_DATATYPEPEMISMATCH
- ESS_OUTERROR_DUPGENLEVNAME
- ESS_OUTERROR_DUPLICATEALIAS
- ESS_OUTERROR2_DUPLICATEATTRCALC
- ESS_OUTERROR_DUPLICATENAME
- ESS_OUTERROR4_DUPNAME_INDIMENSION
- ESS_OUTERROR4_DUPNAME_INGENERATION
- ESS_OUTERROR4_DUPNAME_INLEVEL
- ESS_OUTERROR4_FLOWTAGINCOMPLETE
- ESS_OUTERROR_ILLEGALALIASSTRING
- ESS_OUTERROR2_ILLEGALATTRCALC
- ESS_OUTERROR2_ILLEGALATTRCALCSET
- ESS_OUTERROR2_ILLEGALATTRVALUE
- ESS_OUTERROR2_ILLEGALATTRIBUTEPEARENT
- ESS_OUTERROR2_ILLEGALATTRSET
- ESS_OUTERROR_ILLEGALCOMBOALIAS
- ESS_OUTERROR_ILLEGALCURRENCY
- ESS_OUTERROR2_ILLEGALDATATYPE
- ESS_OUTERROR_ILLEGALDEFAlias
- ESS_OUTERROR_ILLEGALNAME
- ESS_OUTERROR2_ILLEGALORDER
- ESS_OUTERROR2_ILLEGALSCAASSOCS
- ESS_OUTERROR_ILLEGALTAG
- ESS_OUTERROR2_ILLEGALALUDA
- ESS_OUTERROR2_INDEPMBR_BADORDER
- ESS_OUTERROR2_INDEPMBR_NOTLEVEL0
- ESS_OUTERROR2_INDEPMBR_SHAREORLABEL
- ESS_OUTERROR_LEAFLABEL
ESS_OUTERROR2_LEVELMISMATCH
ESS_OUTERROR_MEMBERCALC
ESS_OUTERROR_NOSHAREPROTO
ESS_OUTERROR2_NOTATTRIBUTE
ESS_OUTERROR_NOTIMEDIM
ESS_OUTERROR2_NOTLEVEL0
ESS_OUTERROR4_PROTO_NONUNIQUE
ESS_OUTERROR_SHAREDMEMBERFORMULA
ESS_OUTERROR_SHARENOTLEVEL0
ESS_OUTERROR_SHAREUDA
ESS_OUTERROR4_TI_INCORRECT_MBRTIMESPANS
ESS_OUTERROR4_TI_INVALIDCONSOLIDATION
ESS_OUTERROR4_TI_LINKATTR_INVALID
ESS_OUTERROR4_TI_LINKATTR_INVALIDASSOC
ESS_OUTERROR4_TI_LINKATTR_UNBALANCEDHIER
ESS_OUTERROR4_TI_ONLYONE_SINGLEHIER
ESS_OUTERROR_TIMESPARSE
ESS_OUTERROR2_TWOPASSPARENTNONTWOPASS
ESS_OUTERROR_VIRTLEV0NOFORMULA
ESS_OUTERROR_VIRTBADCHILD
ESS_OUTERROR_VIRTBADPARENT
ESS_OUTERROR_VIRTWHOLEDIMVIRTUAL
ESS_OUTERROR4_20DUPNAME_INPATH

**ESS_OUTLINEINFO_T**

Contains information about the outline.

```c
typedef struct ESS_OUTLINEINFO_T
{
    ESS_BOOL_T     fCaseSensitive;
    ESS_USHORT_T   usOutlineType;
    ESS_BOOL_T     fAutoConfigure;
    ESS_USHORT_T   usNumAliasTables;
    ESS_ALIASNAME_T pAliasTables[1];
    ESS_BOOL_T     fEnableVaryingAttrs;
    ESS_BOOL_T     fNonUniqueName;
    ESS_UCHAR_T    ucImpliedShareSetting;
    ESS_BOOL_T,   fEnableMemberType;
} ESS_OUTLINEINFO_T, *ESS_POUTLINEINFO_T, **ESS_PPOUTLINEINFO_T;
```

706  C Outline API Declarations
<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_BOOL_T</td>
<td>fCaseSensitive</td>
<td>Case-sensitive member names flag.</td>
</tr>
<tr>
<td>ESS_USHORT_T</td>
<td>usOutlineType</td>
<td>Type of the outline. It can be one of the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_DBTYPE_NORMAL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Normal database</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_DBTYPE_CURRENCY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Currency database</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_DBTYPE_NORMALMDX</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Database with MDX type formula</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_DBTYPE_ASO</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Aggregate storage database</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_DBTYPE_ROLAP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ROLAP database</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_DBTYPE_ASO71</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Aggregate storage database with version 7.1 otl file</td>
</tr>
<tr>
<td>ESS_BOOL_T</td>
<td>fAutoConfigure</td>
<td>ESS_TRUE to automatically configure the dimension storage (dense/sparse) when a block-storage outline is saved.</td>
</tr>
<tr>
<td>ESS_USHORT_T</td>
<td>usNumAliasTables</td>
<td>Number of alias tables. This is a read-only field and will be ignored in the ESSSetOutlineInfo() call.</td>
</tr>
<tr>
<td>ESS_ALIASNAME_T</td>
<td>pAliasTables</td>
<td>Array of alias table names existing in the outline. The usNumAliasTables field defines the number of entries in this array. This is a read-only field and will ignored in the ESSSetOutlineInfo() call.</td>
</tr>
<tr>
<td>ESS_BOOL_T</td>
<td>fEnableVaryingAttrs</td>
<td>ESS_TRUE indicates the outline supports varying attributes.</td>
</tr>
<tr>
<td>ESS_BOOL_T</td>
<td>fNonUniqueName</td>
<td>Indicates whether the outline supports duplicate member names.</td>
</tr>
<tr>
<td>ESS_UCHAR_T</td>
<td>ucImpliedShareSetting</td>
<td>Implied Share setting:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- TRUE (default) means that Implied Share is ON</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- FALSE means that Implied Share is OFF</td>
</tr>
<tr>
<td>ESS_BOOL_T</td>
<td>fEnableMemberType</td>
<td>ESS_TRUE indicates member types are enabled.</td>
</tr>
</tbody>
</table>

**ESS_OUTLINEINFOEX_T**

Contains information about the outline.

```c
typedef struct ESS_OUTLINEINFOEX_T {
    ESS_BOOL_T fCaseSensitive;
    ESS_USHORT_T usOutlineType;
    ESS_BOOL_T fAutoConfigure;
    ESS_BOOL_T fNonUniqueName;
    ESS_USHORT_T usNumAliasTables;
} ESS_OUTLINEINFOEX_T;
```
typedef struct ESS_PERSPECTIVE_T
{
    ESS_USHORT_T, usValiditySetType;
    ESS_USHORT_T, usFiller;
    ESS_STR_T, szValiditySetExpr;
    ESS_INT32_T, countOfIndepDims;
    ESS_INT32_T, countOfIndepRanges;
    ESS_PVOID_T*, pIndepMbrs;
} ESS_PERSPECTIVE_T; *ESS_PPERSPECTIVE_T

ESS_PERSPECTIVE_T
Contains information about perspectives and validity sets.

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_BOOL_T</td>
<td>fCaseSensitive</td>
<td>Case-sensitive member names flag.</td>
</tr>
<tr>
<td>ESS_USHORT_T</td>
<td>usOutlineType</td>
<td>Type of the outline. It can be one of these:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>* ESS_DBTYPE_NORMAL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>* ESS_DBTYPE_CURRENCY</td>
</tr>
<tr>
<td>ESS_BOOL_T</td>
<td>fAutoConfigure</td>
<td>ESS_TRUE to automatically configure the dimension storage (dense/sparse) when a block-storage outline is saved.</td>
</tr>
<tr>
<td>ESS_BOOL_T</td>
<td>fNonUniqueName</td>
<td>Indicates whether the outline supports duplicate member names.</td>
</tr>
<tr>
<td>ESS_USHORT_T</td>
<td>usNumAliasTables</td>
<td>Number of alias tables. This is a read-only field and is ignored in the EssOtlSetOutlineInfo() call.</td>
</tr>
<tr>
<td>ESS_ALIASNAME_T</td>
<td>pAliasTables</td>
<td>Array of alias table names existing in the outline. The usNumAliasTables field defines the number of entries in this array. This is a read-only field and is ignored in the EssOtlSetOutlineInfo call.</td>
</tr>
<tr>
<td>ESS_BOOL_T</td>
<td>fEnableVaryingAttrs</td>
<td>ESS_FALSE indicates the outline supports varying attributes.</td>
</tr>
<tr>
<td>ESS_UCHAR_T</td>
<td>ucImpliedShareSetting</td>
<td>Implied share setting for the outline. Possible values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>* ESS_IMPLIEDSHARE_DEFAULT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>* ESS_IMPLIEDSHARE_DEFAULT_ON</td>
</tr>
<tr>
<td></td>
<td></td>
<td>* ESS_IMPLIEDSHARE_DEFAULT_OFF</td>
</tr>
<tr>
<td></td>
<td></td>
<td>* ESS_IMPLIEDSHARE_FORCE_ON</td>
</tr>
<tr>
<td></td>
<td></td>
<td>* ESS_IMPLIEDSHARE_FORCE_OFF</td>
</tr>
<tr>
<td>ESS_BOOL_T</td>
<td>fEnableMemberType</td>
<td></td>
</tr>
<tr>
<td>ESS_UCHAR_T</td>
<td>cSMDateFormatValue</td>
<td></td>
</tr>
<tr>
<td>Data Type</td>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ESS_USHORT_T</td>
<td>usValiditySetType</td>
<td>How members are specified. Possible values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_VALIDITYSET_TYPE_MBRHDLS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_VALIDITYSET_TYPE_MBRNAMS</td>
</tr>
<tr>
<td>ESS_USHORT_T</td>
<td>usFiller</td>
<td>Set to zero</td>
</tr>
<tr>
<td>ESS_STR_T</td>
<td>szValiditySetExpr</td>
<td>MDX expression specified by the MDX type</td>
</tr>
<tr>
<td>ESS_INT32_T</td>
<td>countOfIndepDims</td>
<td>Size of each of the tuples</td>
</tr>
<tr>
<td>ESS_INT32_T</td>
<td>countOfIndepRanges</td>
<td>Number of tuple ranges</td>
</tr>
<tr>
<td>ESS_PVOID_T</td>
<td>pIndepMbrs</td>
<td>Array of member handles (ESS_HMEMBER_T) or member names (ESS_STR_T) depending on usValiditySetType</td>
</tr>
</tbody>
</table>

Descriptions

The terms **perspective** and **validity set** both designate collections of independent members.

- **Perspective** designates any combination of independent members, and is used when querying either the client or server for associations.
- **Validity set** designates the collection of independent members for which an association is true. The term also applies to the set of independent members used for an association or disassociation.

Independent members can be designated as:

- **ESS_VALIDITYSET_TYPE_MBRHDLS**: Independent members are specified as a sequence of ranges (in the XRange sense i.e. Mar 2003-Feb 2004 consists of 2003 starting with March and Jan/Feb of 2004) of member handles.
- **ESS_VALIDITYSET_TYPE_MBRNAMS**: Same as ESS_VALIDITYSET_TYPE_MBRHDLS, except that the ranges are specified with member names.

**ESS_PREDICATE_T**

Contains information about a query description.

typedef struct ESS_PREDICATE_T
{
    ESS_ULONG_T    ulQuery;
    ESS_ULONG_T    ulOptions;
    ESS_STR_T      pszDimension;
    ESS_STR_T      pszString1;
    ESS_STR_T      pszString2;
} ESS_PREDICATE_T, *ESS_PPREDICATE_T, **ESS_PPPREDICATE_T;

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_ULONG_T</td>
<td>ulQuery</td>
<td>Type of query. See EssOtlQueryMembers for more information.</td>
</tr>
<tr>
<td>Data Type</td>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>----------------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ESS_ULONG_T</td>
<td>ulOptions</td>
<td>Options dependent on the query type. See EssOtlQueryMembers for more information.</td>
</tr>
<tr>
<td>ESS_STR_T</td>
<td>pszDimension</td>
<td>Dimension name. See EssOtlQueryMembers for more information.</td>
</tr>
<tr>
<td>ESS_STR_T</td>
<td>pszString1</td>
<td>Input string value. See EssOtlQueryMembers for more information.</td>
</tr>
<tr>
<td>ESS_STR_T</td>
<td>pszString2</td>
<td>Input string value. See EssOtlQueryMembers for more information.</td>
</tr>
</tbody>
</table>

**ESS_SVROTLINFO_T**

Contains information about the outline. This structure can be used by EssGetSrvOutlineInfo.

typedef struct ESS_SVROTLINFO_T
{
    ESS_BOOL_T,      fCaseSensitive;
    ESS_USHORT_T,    usOutlineType;
    ESS_BOOL_T,      fNonUniqueName;
    ESS_USHORT_T,    usNumAliasTables;
    ESS_ALIASNAME_T, pAliasTables, 10;
} ESS_SVROTLINFO_T, *ESS_PSVROTLINFO_T;

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_BOOL_T</td>
<td>fCaseSensitive</td>
<td>Case-sensitive member names flag.</td>
</tr>
<tr>
<td>ESS_USHORT_T</td>
<td>usOutlineType</td>
<td>Type of the outline. It can be one of the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESS_DBTYPE_NORMAL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESS_DBTYPE_CURRENCY</td>
</tr>
<tr>
<td>ESS_BOOL_T</td>
<td>fNonUniqueName</td>
<td>Indicates whether the outline supports duplicate member names.</td>
</tr>
<tr>
<td>ESS_USHORT_T</td>
<td>usNumAliasTables</td>
<td>Number of alias tables. This is a read-only field and will be ignored in the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EssOtlSetOutlineInfo() call.</td>
</tr>
<tr>
<td>ESS_ALIASNAME_</td>
<td>pAliasTables</td>
<td>Array of alias table names existing in the outline. The usNumAliasTables field defines the number of entries in this array. This is a read-only field and will ignored in the EssOtlSetOutlineInfo() call.</td>
</tr>
<tr>
<td>T</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**ESS_VALIDITYSET_T**

Contains information about perspectives and validity sets.

typedef struct ESS_VALIDITYSET_T
{
    ESS_USHORT_T, usValiditySetType;
    ESS_USHORT_T, usFiller;
    ESS_STR_T, szValiditySetExpr;
    ESS_INT32_T, countOfIndepDims;
}

710  C Outline API Declarations
ESS_INT32_T, countOfIndepRanges;
ESS_PVOID_T*, pIndepMbrs;
) ESS_VALIDITYSET_T; *ESS_PVALIDITYSET_T

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_USHORT_T</td>
<td>usValiditySetType</td>
<td>How members are specified. Possible values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESS_VALIDITYSET_TYPE_MBRHDLS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESS_VALIDITYSET_TYPE_MBRNAMS</td>
</tr>
<tr>
<td>ESS_USHORT_T</td>
<td>usFiller</td>
<td>Set to zero</td>
</tr>
<tr>
<td>ESS_STR_T</td>
<td>szValiditySetExpr</td>
<td>MDX expression specified by the MDX type</td>
</tr>
<tr>
<td>ESS_INT32_T</td>
<td>countOfIndepDims</td>
<td>Size of each of the tuples</td>
</tr>
<tr>
<td>ESS_INT32_T</td>
<td>countOfIndepRanges</td>
<td>Number of tuple ranges</td>
</tr>
<tr>
<td>ESS_PVOID_T</td>
<td>pIndepMbrs</td>
<td>Array of member handles (ESS_HMEMBER_T) or member names (ESS_STR_T) depending on usValiditySetType</td>
</tr>
</tbody>
</table>

Description

The terms **perspective** and **validity set** both designate collections of independent members.

- **Perspective** designates any combination of independent members, and is used when querying either the client or server for associations.
- **Validity set** designates the collection of independent members for which an association is true. The term also applies to the set of independent members used for an association or disassociation.

Independent members can be designated as:

- **ESS_VALIDITYSET_TYPE_MBRHDLS**: Independent members are specified as a sequence of ranges (in the XRange sense i.e. Mar 2003-Feb 2004 consists of 2003 starting with March and Jan/Feb of 2004) of member handles.
- **ESS_VALIDITYSET_TYPE_MBRNAMS**: Same as **ESS_VALIDITYSET_TYPE_MBRHDLS**, except that the ranges are specified with member names.
C Outline API Functions

In This Chapter

C Outline API Function Categories ................................................................. 713
C Outline API Function Reference ................................................................. 721

C Outline API Function Categories

C Outline API functions by category:

- “C Outline API Alias Table Functions” on page 713
- “C Outline API Attributes Functions” on page 714
- “C Outline API Dynamic Time Series Functions” on page 715
- “C Outline API Generation Name Functions” on page 715
- “C Outline API Level Name Functions” on page 715
- “C Outline API Member Administration Functions” on page 715
- “C Outline API Member Alias Functions” on page 716
- “C Outline API Member Formula Functions” on page 716
- “C Outline API Member Traversal Functions” on page 717
- “C Outline API Outline Administration Functions” on page 717
- “C Outline API Outline Query Functions” on page 718
- “C Outline API Setup and Cleanup Functions” on page 718
- “C Outline API User-Defined Attributes Functions” on page 719
- “C Outline API User-Defined View Selection Functions” on page 719
- “C Outline API Varying Attributes Functions” on page 719

C Outline API Alias Table Functions

These functions perform operations on alias tables.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EssOtlCreateAliasTable()</td>
<td>Creates an empty alias table in the outline</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>EssOtlCopyAliasTable</td>
<td>Copies an alias table to another alias table</td>
</tr>
<tr>
<td>EssOtlRenameAliasTable</td>
<td>Renames an existing alias table</td>
</tr>
<tr>
<td>EssOtlClearAliasTable</td>
<td>Clears all entries from an existing alias table without deleting the alias table</td>
</tr>
<tr>
<td>EssOtlDeleteAliasTable</td>
<td>Deletes the alias table from the outline and clears all of its entries</td>
</tr>
<tr>
<td>EssOtlSetAliasTableLanguage</td>
<td>Sets a language code for the alias table. An alias table can have multiple language codes.</td>
</tr>
<tr>
<td>EssOtlGetAliasTableLanguages</td>
<td>Gets the set of language codes associated with the alias table.</td>
</tr>
<tr>
<td>EssOtlClearAliasTableLanguages</td>
<td>Clears the set of language codes from the alias table.</td>
</tr>
</tbody>
</table>

### C Outline API Attributes Functions

These C Outline functions are for attributes.

See also “C Outline API Varying Attributes Functions” on page 719.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EssOtlAssociateAttributeDimension</td>
<td>Associates an attribute dimension with a base dimension</td>
</tr>
<tr>
<td>EssOtlAssociateAttributeMember</td>
<td>Associates an attribute member with a base dimension member</td>
</tr>
<tr>
<td>EssOtlDisassociateAttributeDimension</td>
<td>Disassociates an attribute dimension from a base dimension</td>
</tr>
<tr>
<td>EssOtlDisassociateAttributeMember</td>
<td>Disassociates an attribute member from a base dimension member</td>
</tr>
<tr>
<td>EssOtlFindAttributeMembers</td>
<td>Returns all base dimension members that are associated with an attribute member</td>
</tr>
<tr>
<td>EssOtlFreeStructure</td>
<td>Frees memory dynamically allocated for string type attribute information</td>
</tr>
<tr>
<td>EssOtlGetAssociatedAttributes</td>
<td>Returns all attribute dimension members that are associated with a base dimension member or base dimension</td>
</tr>
<tr>
<td>EssOtlGetAttributeInfo</td>
<td>Returns attribute information for a given attribute member or dimension</td>
</tr>
<tr>
<td>EssOtlGetAttributeSpecifications</td>
<td>Retrieves attribute specifications for the outline</td>
</tr>
<tr>
<td>EssOtlQueryAttributes</td>
<td>Queries the outline for member attribute information</td>
</tr>
<tr>
<td>EssOtlQueryAttributesEx</td>
<td></td>
</tr>
<tr>
<td>EssOtlSetAttributeSpecifications</td>
<td>Sets attribute specifications for the outline</td>
</tr>
</tbody>
</table>

See “C Main API Attributes Functions ” on page 201.
C Outline API Dynamic Time Series Functions

These functions enable and work with Dynamic Time Series members and aliases.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EssOtlDeleteDTSMemberAlias</td>
<td>Deletes an alias name for a Dynamic Time Series member.</td>
</tr>
<tr>
<td>EssOtlEnableDTSMember</td>
<td>Enables a new Dynamic Time Series members for the outline.</td>
</tr>
<tr>
<td>EssOtlGetEnabledDTSMembers</td>
<td>Gets the defined Dynamic Time Series members for the outline.</td>
</tr>
<tr>
<td>EssOtlGetDTSMemberAlias</td>
<td>Gets an alias name for a Dynamic Time Series member.</td>
</tr>
<tr>
<td>EssOtlSetDTSMemberAlias</td>
<td>Sets an alias name for a Dynamic Time Series member.</td>
</tr>
</tbody>
</table>

C Outline API Generation Name Functions

These functions perform operations on generation names.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EssOtlGetGenName</td>
<td>Gets the generation name for the specified dimension and generation number</td>
</tr>
<tr>
<td>EssOtlGetGenNames</td>
<td>Retrieves all generation names specified for a particular dimension</td>
</tr>
<tr>
<td>EssOtlSetGenName</td>
<td>Sets the generation name for the specified dimension and generation number</td>
</tr>
<tr>
<td>EssOtlDeleteGenName</td>
<td>Deletes the generation name of the specified dimension and level number</td>
</tr>
</tbody>
</table>

C Outline API Level Name Functions

These functions perform operations on level names.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EssOtlGetLevelName</td>
<td>Gets the level name of the specified dimension</td>
</tr>
<tr>
<td>EssOtlGetLevelNames</td>
<td>Retrieves all level names specified for a particular dimension</td>
</tr>
<tr>
<td>EssOtlSetLevelName</td>
<td>Sets the level name of the specified dimension</td>
</tr>
<tr>
<td>EssOtlDeleteLevelName</td>
<td>Deletes the level name of the specified dimension</td>
</tr>
</tbody>
</table>

C Outline API Member Administration Functions

These functions assist in managing the members of an outline.
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EssOtlAddMember</td>
<td>Adds a member</td>
</tr>
<tr>
<td>EssOtlDeleteMember</td>
<td>Deletes a member</td>
</tr>
<tr>
<td>EssOtlAddDimension</td>
<td>Adds a dimension</td>
</tr>
<tr>
<td>EssOtlDeleteDimension</td>
<td>Deletes a dimension</td>
</tr>
<tr>
<td>EssOtlRenameMember</td>
<td>Renames a member</td>
</tr>
<tr>
<td>EssOtlMoveMember</td>
<td>Moves a member</td>
</tr>
<tr>
<td>EssOtlFindMember</td>
<td>Finds a member</td>
</tr>
<tr>
<td>EssOtlGetMemberCommentEx</td>
<td>Gets the extended comment for a specified member</td>
</tr>
<tr>
<td>EssOtlGetMemberInfo</td>
<td>Gets member information</td>
</tr>
<tr>
<td>EssOtlSetMemberCommentEx</td>
<td>Sets the extended comment for a specified member</td>
</tr>
<tr>
<td>EssOtlSetMemberInfo</td>
<td>Sets member information</td>
</tr>
<tr>
<td>EssOtlGetMemberSolveOrder</td>
<td>Gets member solve order</td>
</tr>
<tr>
<td>EssOtlSetMemberSolveOrder</td>
<td>Sets member solve order</td>
</tr>
<tr>
<td>EssOtlGetDimensionSolveOrder</td>
<td>Gets dimension solve order</td>
</tr>
<tr>
<td>EssOtlSetDimensionSolveOrder</td>
<td>Sets dimension solve order</td>
</tr>
</tbody>
</table>

**C Outline API Member Alias Functions**

These functions perform operations on member aliases.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EssOtlFindAlias</td>
<td>Finds a member with the specified alias name</td>
</tr>
<tr>
<td>EssOtlGetMemberAlias</td>
<td>Gets the default member alias for a specific member in a specific alias table</td>
</tr>
<tr>
<td>EssOtlSetMemberAlias</td>
<td>Sets the default member alias for a specific member in a specific alias table</td>
</tr>
<tr>
<td>EssOtlDeleteMemberAlias</td>
<td>Deletes the default member alias for a specific member in a specific alias table</td>
</tr>
</tbody>
</table>

**C Outline API Member Formula Functions**

These functions perform operations on member formulas.
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EssOtlGetMemberFormula</td>
<td>Gets the formula of the specified member</td>
</tr>
<tr>
<td>EssOtlGetMemberLastFormula</td>
<td>Returns the last formula used to calculate the member</td>
</tr>
<tr>
<td>EssOtlSetMemberFormula</td>
<td>Sets the formula for the specified member</td>
</tr>
<tr>
<td>EssOtlDeleteMemberFormula</td>
<td>Deletes the formula of the specified member</td>
</tr>
</tbody>
</table>

**C Outline API Member Traversal Functions**

These functions are used in traversing the outline tree.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EssOtlGetFirstMember</td>
<td>Returns a member handle to the first member in the outline; the first dimension defined in the outline</td>
</tr>
<tr>
<td>EssOtlGetChild</td>
<td>Returns a member handle to the child of a member</td>
</tr>
<tr>
<td>EssOtlGetParent</td>
<td>Returns a member handle to the parent of a member</td>
</tr>
<tr>
<td>EssOtlGetNextSibling</td>
<td>Returns a member handle to the next sibling of a member</td>
</tr>
<tr>
<td>EssOtlGetPrevSibling</td>
<td>Returns a member handle to the previous sibling of a member</td>
</tr>
<tr>
<td>EssOtlGetNextSharedMember</td>
<td>Returns a member handle to the next shared member of a real member</td>
</tr>
<tr>
<td>EssOtlQueryGetFirstDimension</td>
<td>Returns the dimension handle of the first dimension in the outline</td>
</tr>
<tr>
<td>EssOtlQueryGetNextDimension()</td>
<td>Returns the next dimension handle of the dimension in the outline opened in query mode</td>
</tr>
</tbody>
</table>

**C Outline API Outline Administration Functions**

These functions assist in managing outlines.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EssOtlGetOutlineInfo</td>
<td>Returns information about the outline file</td>
</tr>
<tr>
<td>EssOtlGetUpdateTime</td>
<td>Returns the timestamp for the specified outline</td>
</tr>
<tr>
<td>EssOtlSetOutlineInfo</td>
<td>Sets outline information</td>
</tr>
<tr>
<td>EssOtlVerifyOutline</td>
<td>Verifies that an outline is correct</td>
</tr>
<tr>
<td>EssOtlSortChildren</td>
<td>Sorts the children of an outline member</td>
</tr>
<tr>
<td>EssOtlGenerateCurrencyOutline</td>
<td>Generates a currency outline based on the existing outline</td>
</tr>
<tr>
<td>EssOtlGetASOCompressionDimension</td>
<td>Gets aggregate storage compression dimension</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td><strong>EssOtlSetASOCompressionDimension</strong></td>
<td>Sets aggregate storage compression dimension</td>
</tr>
</tbody>
</table>

### C Outline API Outline Query Functions

These functions assist in making outline queries.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>EssOtlGetMemberField</strong></td>
<td>Returns data for the specified field of a specified outline member</td>
</tr>
<tr>
<td><strong>EssOtlOpenOutlineQuery</strong></td>
<td>Opens an existing outline</td>
</tr>
<tr>
<td><strong>EssOtlQueryMembers</strong></td>
<td>Queries the outline, using a member handle</td>
</tr>
<tr>
<td><strong>EssOtlQueryMembersByName</strong></td>
<td>Queries the outline, using a member name string</td>
</tr>
<tr>
<td><strong>EssOtlQueryMembersEx</strong></td>
<td>Queries specific members and member fields, and returns an array of member handles</td>
</tr>
<tr>
<td><strong>EssOtlQueryAttributes</strong></td>
<td>Queries the outline for attribute information</td>
</tr>
<tr>
<td>** EssOtlQueryAttributesEx**</td>
<td></td>
</tr>
<tr>
<td><strong>EssOtlFreeMembers</strong></td>
<td>Frees the member array returned from <strong>EssOtlQueryMembers()</strong></td>
</tr>
</tbody>
</table>

### C Outline API Setup and Cleanup Functions

These functions start and finish editing operations on an outline.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>EssOtlNewOutline</strong></td>
<td>Creates a new outline</td>
</tr>
<tr>
<td><strong>EssOtlOpenOutline</strong></td>
<td>Opens an existing outline</td>
</tr>
<tr>
<td><strong>EssOtlOpenOutlineEx</strong></td>
<td>Opens an existing outline (for Unicode mode)</td>
</tr>
<tr>
<td><strong>EssOtlWriteOutline</strong></td>
<td>Writes the outline to the server</td>
</tr>
<tr>
<td><strong>EssOtlWriteOutlineEx</strong></td>
<td>Writes the outline to the server (for Unicode mode)</td>
</tr>
<tr>
<td><strong>EssOtlRestructure</strong></td>
<td>Restructures the database based on the newly saved outline</td>
</tr>
<tr>
<td><strong>EssOtlCloseOutline</strong></td>
<td>Frees resources associated with the outline</td>
</tr>
</tbody>
</table>

### C Outline API Unicode Mode Functions

The following functions help you work with the Essbase Server outlines in Unicode mode.
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EssOtlWriteOutlineEx</td>
<td>Writes the outline to the server, specifying whether to save in UTF-8 encoding or in non-Unicode encoding.</td>
</tr>
<tr>
<td>EssOtlOpenOutlineEx</td>
<td>Opens the outline of a Unicode-mode application.</td>
</tr>
</tbody>
</table>

## C Outline API User-Defined Attributes Functions

These functions perform operations on user-defined attributes (UDAs).

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EssOtlGetDimensionUserAttributes</td>
<td>Gets the UDAs of the specified dimension</td>
</tr>
<tr>
<td>EssOtlGetUserAttributes</td>
<td>Gets the UDAs of the specified member</td>
</tr>
<tr>
<td>EssOtlSetUserAttribute</td>
<td>Sets a UDA for the specified member</td>
</tr>
<tr>
<td>EssOtlDeleteUserAttribute</td>
<td>Deletes a UDA of the specified member</td>
</tr>
</tbody>
</table>

## C Outline API User-Defined View Selection Functions

These functions define view selection criteria for aggregation of aggregate storage databases.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EssOtlSetAggLevelUsage</td>
<td>Applies view selection properties to stored hierarchies</td>
</tr>
<tr>
<td>EssOtlGetAggLevelUsage</td>
<td>Returns the applied view selection properties on stored hierarchies</td>
</tr>
<tr>
<td>EssOtlAddQueryHint</td>
<td>Adds a query hint to the outline to aid in view selection</td>
</tr>
<tr>
<td>EssOtlGetQueryHint</td>
<td>Returns specified query hint defined on an outline</td>
</tr>
<tr>
<td>EssOtlSetQueryHint</td>
<td>Sets a query hint</td>
</tr>
<tr>
<td>EssOtlGetNumQueryHints</td>
<td>Returns the number of query hints</td>
</tr>
<tr>
<td>EssOtlGetQueryHintSize</td>
<td>Returns the size (in number of members) of query hints</td>
</tr>
<tr>
<td>EssOtlDeleteQueryHint</td>
<td>Deletes specified query hint and decreases the number of hints by one</td>
</tr>
</tbody>
</table>

## C Outline API Varying Attributes Functions

These C Outline functions are for varying attributes.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EssOtlQueryVaryingAttributes</td>
<td>Queries the outline for member varying attribute information</td>
</tr>
</tbody>
</table>
### Function Description

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EssOtlDetailQueryVaryingAttributes</td>
<td>Similar to EssOtlQueryVaryingAttributes.</td>
</tr>
<tr>
<td>EssOtlVaryingAssociateAttribute</td>
<td>Associates a varying attribute member with a base dimension member</td>
</tr>
<tr>
<td>EssOtlVaryingAssociateAttributeDimension</td>
<td>Associates a varying attribute dimension with a base dimension</td>
</tr>
<tr>
<td>EssOtlVaryingDisassociateAttribute</td>
<td>Disassociates a varying attribute dimension from a base dimension</td>
</tr>
<tr>
<td>EssOtlVaryingGetAssociatedAttributes</td>
<td>Returns all varying attribute members that are associated with a base dimension member or base dimension</td>
</tr>
<tr>
<td>EssOtlVaryingGetAttributeIndepDims</td>
<td>Returns the independent dimensions, if any, for the dimension containing the specified varying attribute member</td>
</tr>
</tbody>
</table>

These APIs may not be fully compatible with future implementations.

See “C Main API Attributes Functions ” on page 201.

### About Varying Attributes

Attribute associations can depend on outside factors. For example

- Over time a client can have different sales representatives assigned to it.
- Over time or based on market territory, packaging for a product can be different.

The varying attributes feature enables you to keep track of values for each factor. For example, consider the situation where the sales representative attribute association for Customer A gets changed in May. Customer sales totals and sales representative assignments over the first six months look like this:

<table>
<thead>
<tr>
<th>Jan</th>
<th>Feb</th>
<th>Mar</th>
<th>Apr</th>
<th>May</th>
<th>Jun</th>
</tr>
</thead>
<tbody>
<tr>
<td>5540</td>
<td>2190</td>
<td>1580</td>
<td>300</td>
<td>2455</td>
<td>3255</td>
</tr>
<tr>
<td>Jones</td>
<td>Jones</td>
<td>Jones</td>
<td>Jones</td>
<td>Smith</td>
<td>Smith</td>
</tr>
</tbody>
</table>

Using the varying attributes feature, retrievals can reflect that Jones sold Customer A $9610 (sum of Jan, Feb, and Mar) and Smith sold $5680 (sum of May and Jun). Without this feature, the only known representative is the current representative, Smith, and all sales ($15290) get attributed to him.

### Varying Attribute Terminology

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>independent dimensions</td>
<td>The dimension upon which varying attributes depend; in the above example, the Year dimension.</td>
</tr>
<tr>
<td>perspective</td>
<td>A combination of independent dimension members that is used when querying for associations. Defined in &quot;ESS_PERSPECTIVE_T&quot; on page 708.</td>
</tr>
<tr>
<td>validity set</td>
<td>The collection of independent dimension members for which an association is true. Defined in &quot;ESS_VALIDITYSET_T&quot; on page 710.</td>
</tr>
</tbody>
</table>
Outline Construction

Varying attributes are constructed in the API with the following flow:

<table>
<thead>
<tr>
<th>Item</th>
<th>Outline API Call</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Set the outline type to accept varying attributes</td>
<td>EssOtlSetOutlineInfo, where pOutlineInfo-&gt;fEnableVaryingAttrs = ESS_TRUE.</td>
</tr>
<tr>
<td>2. Identify the independent dimension</td>
<td>EssOtlSetMemberInfo, where pMemberInfo-&gt;fIndependentDim = ESS_TRUE</td>
</tr>
<tr>
<td>3. Associate the attribute dimension to the base dimension and identify independent dimensions</td>
<td>EssOtlVaryingAssociateAttributeDimension</td>
</tr>
<tr>
<td>4. Associate attribute dimension members independent dimension members with base dimension members</td>
<td>EssOtlVaryingAssociateAttribute</td>
</tr>
<tr>
<td>5. Save and restructure the outline.</td>
<td>The same as when making other outline changes.</td>
</tr>
</tbody>
</table>

Maintenance Tasks

<table>
<thead>
<tr>
<th>Item</th>
<th>Outline API Call</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add a new association to independent members.</td>
<td>EssOtlVaryingAssociateAttribute</td>
</tr>
<tr>
<td>Remove independent member associations</td>
<td>EssOtlVaryingDisassociateAttribute</td>
</tr>
<tr>
<td>View existing independent dimension member associations</td>
<td>EssOtlQueryVaryingAttributes or EssOtlVaryingGetAssociatedAttributes</td>
</tr>
<tr>
<td>Disassociate attribute dimensions from base dimensions</td>
<td>EssOtlDisassociateAttributeDimension (disassociates all attribute dimensions).</td>
</tr>
</tbody>
</table>

C Outline API Function Reference

Consult the Contents page for the list of C Outline API functions, which are prefaced with EssOtl.

EssOtlAddDimension

Adds a dimension to the outline and sets the member's attributes. The call also specifies a member of the new dimension to associate data with when the outline is restructured.

Syntax

ESS_FUNC_M EssOtlAddDimension (hOutline, pMemberInfo, hPrevSibling, pssDataMbr, phMember);
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline;</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline context handle.</td>
</tr>
<tr>
<td>pMemberInfo;</td>
<td>“ESS_MBRINFO_T” on page 699</td>
<td>Member information structure defining the member and its attributes.</td>
</tr>
<tr>
<td>hPrevSibling</td>
<td>ESS_HMEMBER_T</td>
<td>Handle of previous sibling. If this field is ESS_NULL, the dimension becomes the first dimension in the outline. Otherwise, the dimension is placed after the dimension specified in hPrevSibling.</td>
</tr>
<tr>
<td>pszDataMbr;</td>
<td>ESS_STR_T</td>
<td>Member name of a member in the new dimension that will receive the data values when the outline is restructured. If this field is ESS_NULL, the dimension member itself is used.</td>
</tr>
<tr>
<td>phMember</td>
<td>ESS_PHMEMBER_T</td>
<td>Handle of new member returned from the API.</td>
</tr>
</tbody>
</table>

**Notes**

- The ESS_MBRINFO_T structure must be created and filled before calling this function.
- To add an attribute dimension, you must call this function.
- To add a dimension that is not an attribute dimension, you can call this function or EssOtlAddMember().
  - EssOtlAddDimension() gives you the benefit of selecting any member in the added dimension to be assigned the data values associated with the existing dimensions.
  - If EssOtlAddMember() is used, the top member (dimension) of the added dimension is used.
- In order for the pszDataMbr field to take effect, the outline must have been opened using EssOtlOpenOutline() with the fKeepTrans flag set to ESS_TRUE.
- The member referred to in the pszDataMbr field is added to the new dimension using EssOtlAddMember() after the dimension is created. If the referred to member doesn’t exist when restructuring takes place, the dimension member is used instead.
- For an attribute dimension, you must set the fields of ESS_MBRINFO_T as follows:

<table>
<thead>
<tr>
<th>Field</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>usConsolidation</td>
<td>ESS_UCALC_NOOP</td>
</tr>
<tr>
<td>fTwoPass</td>
<td>ESS_FALSE</td>
</tr>
<tr>
<td>fExpense</td>
<td>ESS_FALSE</td>
</tr>
<tr>
<td>usConversion</td>
<td>ESS_CONV_NONE</td>
</tr>
<tr>
<td>usTimeBalance</td>
<td>ESS_TIMEBAL_NONE</td>
</tr>
<tr>
<td>usSkip</td>
<td>ESS_SKIP_NONE</td>
</tr>
<tr>
<td>usShare</td>
<td>ESS_SHARE_DYNCALCNOSTORE</td>
</tr>
<tr>
<td>usStorage</td>
<td>ESS_DIMTYPE_SPARSE</td>
</tr>
<tr>
<td>Field</td>
<td>Setting</td>
</tr>
<tr>
<td>------------------------</td>
<td>--------------------------------</td>
</tr>
<tr>
<td>usCategory</td>
<td>ESS_CAT_ATTRIBUTE</td>
</tr>
<tr>
<td>usStorageCategory</td>
<td>ESS_STORECAT_ATTRIBUTE</td>
</tr>
<tr>
<td>Attribute.usDataType</td>
<td>One of the following attribute member data types:</td>
</tr>
<tr>
<td></td>
<td>ESS_ATTRMBRDT_BOOL</td>
</tr>
<tr>
<td></td>
<td>ESS_ATTRMBRDT_DATETIME</td>
</tr>
<tr>
<td></td>
<td>ESS_ATTRMBRDT_DOUBLE</td>
</tr>
<tr>
<td></td>
<td>ESS_ATTRMBRDT_STRING</td>
</tr>
</tbody>
</table>

- An attribute dimension must be associated with a base dimension.
- Attribute dimensions must be placed after base dimensions and standard dimensions.

**Return Value**

Returns 0 if successful; otherwise one of the following:

- OTLAPI_BAD_CONSOL
- OTLAPI_BAD_MBRNAME
- OTLAPI_ERR_ADDNAMEUSED
- OTLAPI_ERR_ADDDELETEDIM_DYNAMICCALC
- OTLAPI_ERR_BADSHARE
- OTLAPI_ERR_BADSKIP
- OTLAPI_ERR_BADSTORAGE
- OTLAPI_ERR_BADSTORAGECATEGORY
- OTLAPI_ERR_BADTIMEBAL
- OTLAPI_ERR_CURTOOMANYDIMS
- OTLAPI_ERR_ILLEGALBOOLEAN
- OTLAPI_ERR_ILLEGALCURRENCY
- OTLAPI_ERR_ILLEGALDATE
- OTLAPI_ERR_ILLEGALNUMERIC
- OTLAPI_ERR_ILLEGALTAG
- OTLAPI_ERR_LEAFLABEL
- OTLAPI_ERR_NONATTRDIMFOLLOWED
- OTLAPI_ERR_NOSHAREPROTO
- OTLAPI_ERR_NOTIMEDIM

**Example**

```c
#include <essapi.h>
#include <essotl.h>

ESS_STS_T     sts = 0;
ESS_OUTLINEINFO_T NewInfo;
```
memset (&NewInfo, '\0', sizeof(NewInfo));
sts = EssOtlNewOutline(hCtx, &NewInfo, &hOutline);
if (!sts)
{
    memset(&MbrInfo, '\0', sizeof(MbrInfo));
    strcpy(MbrInfo.szMember, "Measures");
    MbrInfo.usStorage = ESS_DIMTYPE_SPARSE;
    MbrInfo.usCategory = ESS_CAT_ACCOUNTS;
    sts = EssOtlAddDimension(hOutline, &MbrInfo, ESS_NULL, "Profit", &hDimMeasures);
}

See Also

- EssOtlAddMember
- EssOtlDeleteDimension
- EssOtlDeleteMember
- EssOtlGetMemberInfo

**EssOtlAddQueryHint**

Adds a query hint to the outline to aid in view selection.

Hints are numbered from 1 to $n$. The first query hint has a hint number of 1. Each new query hint is added to the end of the list, with its number increased by 1.

**Syntax**

```c
ESS_FUNC_M EssOtlAddQueryHint (hOutline, numMembers, pMemberArray);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline context handle (input).</td>
</tr>
<tr>
<td>numMembers</td>
<td>ESS_SHORT_T</td>
<td>Number of members in the array provided - usually the number of real dimensions in the outline. (input)</td>
</tr>
<tr>
<td>pMemberArray</td>
<td>ESS_PHMEMBER_T</td>
<td>An array of members for the hint. Usually the array has one member per real dimension, with NULL used for dimensions that are not part of the hint. This array needs to be allocated.</td>
</tr>
</tbody>
</table>

**Notes**

- Level usage constraints override query hints whenever a conflict occurs (see SetAggLevelUsage).
- Hints may not contain dynamic, label-only, or shared members.
- Hints may become invalid when the outline changes. Invalid hints result in a warning message.
Query hints enable you to influence normal view selection by informing Essbase about the profile of common queries.

This function is applicable only to Release 9.3 or higher aggregate storage databases.

Query hints are written as MDX tuples, with no more than one member from each dimension specified.

Each member used in the query hint is considered a representative member. Essbase Server interprets representative members as "this member or any member at the similar level of aggregation." For example, using a query hint of (Qtr1, Sales, 100, East, Actual) on Sample Basic means that quarterly, actual profit margin measures for level 1 products at level 1 markets is a common type of query.

For any given dimension, Essbase Server interprets the omission of representative members to mean that any member from the dimension may be used in a query. For example, using a query hint of (Sales, 100, East) on Sample Basic means that profit margin measures for level 1 products at level 1 markets is a common type of query, regardless of Year and Scenario dimensions, which were omitted. The hint (Sales, 100, East) is treated as identical to (NULL, Sales, 100, East, NULL).

Return Value

Returns 0 if successful.

Example

```c
ESS_STS_T     sts = ESS_STS_NOERR;
ESS_HOUTLINE_T hOutline = ESS_NULL;
ESS_HMEMBER_T hMember1 = ESS_NULL;
ESS_HMEMBER_T hMember2 = ESS_NULL;
ESS_HMEMBER_T hMember3 = ESS_NULL;
ESS_HMEMBER_T hMember[3];
ESS_SHORT_T   nmMembers = 3;

/* code to assign hOutline variable omitted */
/* code to assign hMember1 variable to member "Sales" omitted */
/* code to assign hMember2 variable to member "100" omitted */
/* code to assign hMember3 variable to member "East" omitted */
hMember[0] = hMember1;
hMember[1] = hMember2;
hMember[2] = hMember3;

if (hOutline)
{
    sts = EssOtlAddQueryHint(hOutline, nmMembers, hMember);
    if (sts)
        printf("Error (%ld) adding QueryHint\n", sts);
} else
{
    if (!hOutline)
        printf("Outline not provided\n");
}
```
### EssOTIAddMember

Adds a member to the outline and sets the member's attributes.

#### Syntax

```c
ESS_FUNC_M EssOTIAddMember (hOutline, pMemberInfo, hParent, hPrevSibling, phMember);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline;</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline context handle.</td>
</tr>
<tr>
<td>pMemberInfo;</td>
<td>“ESS_MBRINFO_T” on page 699</td>
<td>Member information structure defining the member and its attributes.</td>
</tr>
<tr>
<td>hParent;</td>
<td>ESS_HMEMBER_T</td>
<td>Handle of parent. This field is used only if the hPrevSibling field is ESS_NULL.</td>
</tr>
<tr>
<td>hPrevSibling;</td>
<td>ESS_HMEMBER_T</td>
<td>Handle of previous sibling.</td>
</tr>
<tr>
<td>phMember;</td>
<td>ESS_PHMEMBER_T</td>
<td>Handle of new member returned from the API.</td>
</tr>
</tbody>
</table>

#### Notes

- The ESS_MBRINFO_T structure must be created and filled before calling this function.
- The member name must be unique unless you are creating a shared member.
- Position of the added member:
  - The new member is inserted following the hPrevSibling member.
  - If the hPrevSibling field is ESS_NULL, the new member becomes the first child of the parent specified by hParent.
  - If both hParent and hPrevSibling are ESS_NULL, the new member becomes the first dimension in the outline.
- To add a shared member:
  - The shared member must be a zero-level (leaf node) member. (Shared members cannot have children.)
  - The actual member must already exist in the dimension.
  - Set the usShare field of the ESS_MBRINFO_T structure to ESS_SHARE_SHARE.
- To add a LABEL member:
  - You must first add the member without the label attribute set.
• Next, add its children. (A label member must have children.)
• Then, use \texttt{EssOtlSetMemberInfo()} to set the label tag of the label member.

To add an attribute member, set the fields of \texttt{ESS\_MBRINFO\_T} as follows:

<table>
<thead>
<tr>
<th>Field</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>usConsolidation</td>
<td>ESS_UCALC_NOOP</td>
</tr>
<tr>
<td>fTwoPass</td>
<td>ESS_FALSE</td>
</tr>
<tr>
<td>fExpense</td>
<td>ESS_FALSE</td>
</tr>
<tr>
<td>usConversion</td>
<td>ESS_CONV_NONE</td>
</tr>
<tr>
<td>usTimeBalance</td>
<td>ESS_TIMEBAL_NONE</td>
</tr>
<tr>
<td>usSkip</td>
<td>ESS_SKIP_NONE</td>
</tr>
<tr>
<td>usShare</td>
<td>ESS_SHARE_DYNCALCNOSTORE</td>
</tr>
<tr>
<td>usStorage</td>
<td>ESS_DIMTYPE_SPARSE</td>
</tr>
<tr>
<td>usCategory</td>
<td>ESS_CAT_ATTRIBUTE</td>
</tr>
<tr>
<td>usStorageCategory</td>
<td>ESS_STORECAT_ATTRIBUTE</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Attribute.\texttt{usDataType}</th>
<th>For an attribute dimension or zero-level (leaf node) attribute member, set one of the following data types:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• ESS_ATTRMBRDT_BOOL</td>
</tr>
<tr>
<td></td>
<td>• ESS_ATTRMBRDT_DATETIME</td>
</tr>
<tr>
<td></td>
<td>• ESS_ATTRMBRDT_DOUBLE</td>
</tr>
<tr>
<td></td>
<td>• ESS_ATTRMBRDT_STRING</td>
</tr>
</tbody>
</table>

You may instead set a zero-level (leaf node) attribute member to ESS\_ATTRMBRDT\_AUTO.

You may set attribute members that are not zero level to ESS\_ATTRMBRDT\_NONE or ESS\_ATTRMBRDT\_AUTO.

• Notes on Adding an Attribute Member:
  • Adding a zero-level attribute member that is not of type ESS\_ATTRMBRDT\_STRING also sets the \texttt{szMember} field of the \texttt{ESS\_MBRINFO\_T} structure to the attribute member's long name, using the specifications for the outline in the “ESS\_ATTRSPECS\_T” on page 119 structure.
  • You must set \texttt{usCategory} and \texttt{usStorageCategory} for an attribute member, as well as an attribute dimension. (You need not set \texttt{usCategory} and \texttt{usStorageCategory} for a base member. You must set them for a base dimension only.)
  • Do not set the \texttt{szDimName} field of the \texttt{ESS\_MBRINFO\_T} structure.
  • For a zero-level attribute member that is not of type ESS\_ATTRMBRDT\_STRING, do not set the \texttt{Attribute.value} field of the “ESS\_ATTRIBUTEVALUE\_T” on page 118 structure. The attribute value is derived internally by converting the attribute member long name.
If you set an attribute member’s data type to ESS_ATTRMBRDT_AUTO, Essbase does the following:

- Sets the member’s data type to the data type of its dimension, if the member name can be converted to a value of that type.
- If the member name cannot be converted to a value of the dimension’s data type, sets the member’s data type to ESS_ATTRMBRDT_NONE.
- For the first child member converted from ESS_ATTRMBRDT_AUTO to a data type other than ESS_ATTRMBRDT_NONE, converts the parent’s long name to a short name.

To add a dimension:

- To add an attribute dimension, call EssOtIAddDimension(). Do not call EssOtIAddMember().
- To add a dimension that is not an attribute dimension, call either EssOtIAddDimension() or EssOtIAddMember().
  - EssOtIAddDimension() gives you the benefit of selecting any member in the added dimension to be assigned the data values associated with the existing dimensions.
  - If EssOtIAddMember() is used, the top member (dimension) of the added dimension is assigned the data values associated with the existing dimensions.

Return Value

Returns 0 if successful; otherwise one of the following:

- OTLAPI_BAD_CONSOL
- OTLAPI_BAD_MBRNAME
- OTLAPI_ERR_ADDNAMEUSED
- OTLAPI_ERR_BADSHARE
- OTLAPI_ERR_BADSKIP
- OTLAPI_ERR_BADSTORAGE
- OTLAPI_ERR_BADSTORAGECATEGORY
- OTLAPI_ERR_BADTIMEBAL
- OTLAPI_ERR_CURTOOMANYDIMS
- OTLAPI_ERR_ILLEGALBOOLEAN
- OTLAPI_ERR_ILLEGALCURRENCY
- OTLAPI_ERR_ILLEGALDATE
- OTLAPI_ERR_ILLEGALNUMERIC
- OTLAPI_ERR_ILLEGALTAG
- OTLAPI_ERR_LEAFLABEL
- OTLAPI_ERR_NOSHAKEPROTO
- OTLAPI_ERR_NOTIMEDIM
Example

```c
#include <essapi.h>
#include <essotl.h>

ESS_STS_T         sts = 0;
ESS_OBJDEF_T       Object;
ESS_HOUTLINE_T     hOutline;
ESS_MBRINFO_T      MbrInfo;
ESS_HMEMBER_T      hMemberProfit;
ESS_HMEMBER_T      hNewMember;
ESS_APPNAME_T      szAppName;
ESS_DBNAME_T       szDbName;
ESS_OBJNAME_T      szFileName;
memezt(&Object, '\0', sizeof(Object));
Object.hCtx = hCtx;
Object.ObjType = ESS_OBJTYPE_OUTLINE;
strcpy(szAppName, "Sample");
strcpy(szDbName, "Basic");
strcpy(szFileName, "Basic");
Object.AppName = szAppName;
Object.DbName = szDbName;
Object.FileName = szFileName;

sts = EssOtlOpenOutline(hCtx, &Object,
                        ESS_TRUE, ESS_TRUE, &hOutline);

if (!sts)
{
    sts = EssOtlFindMember(hOutline, "Profit",
                            &hMemberProfit);
}
if (!sts && hMemberProfit)
{
    memset(&MbrInfo, '\0', sizeof(MbrInfo));
    strcpy(MbrInfo.szMember, "Inventory");
    sts = EssOtlAddMember(hOutline, &MbrInfo,
                          ESS_NULL, hMemberProfit, &hNewMember);
}
```

See Also

- `EssOtlAddDimension`
- `EssOtlDeleteMember`
- `EssOtlDeleteDimension`
- `EssOtlSetMemberInfo`
- `EssOtlFindMember`

**EssOtlAssociateAttributeDimension**

Associates an attribute dimension with a standard or base dimension.

**Syntax**
### Parameter | Data Type | Description
--- | --- | ---
hOutline; | ESS_HOUTLINE_T | Handle to the outline
hStandardDimension; | ESS_HMEMBER_T | Handle to the standard or base dimension
hAttributeDimension; | ESS_HMEMBER_T | Handle to the attribute dimension

**Notes**

- The attribute dimension must be sparse.
- The standard or base dimension must be sparse.
- You must associate an attribute dimension with a standard or base dimension.
- You can associate more than one attribute dimension with a base dimension.
- You cannot associate an attribute dimension with more than one base dimension.

**Example**

```c
void ESS_OtlAssociateAttributeDimension()
{
    ESS_STS_T        sts = ESS_STS_NOERR;
    ESS_HOUTLINE_T   hOutline;
    ESS_HMEMBER_T    hBaseMbr;
    ESS_HMEMBER_T    hAttrMbr;
    ESS_OBJDEF_T     Object;
    ESS_APPNAME_T    szAppName;
    ESS_DBNAME_T     szDbName;
    ESS_OBJNAME_T    szFileName;
    ESS_PROCSTATE_T  pState;

    memset(&Object, '\0', sizeof(Object));
    Object.hCtx = hCtx;
    Object.ObjType = ESS_OBJTYPE_OUTLINE;
    strcpy(szAppName, "Sample");
    strcpy(szDbName, "Basic");
    strcpy(szFileName, "Basic");
    Object.AppName = szAppName;
    Object.DbName = szDbName;
    Object.FileName = szFileName;

    sts = EssOtlOpenOutline(hCtx, &Object, ESS_TRUE, ESS_TRUE, &hOutline);
    printf("EssOtlOpenOutline() sts: %ld\n",sts);

    sts = EssOtlFindMember(hOutline, "Product", &hBaseMbr);
    printf("EssOtlFindMember() sts: %ld\n",sts);

    sts = EssOtlFindMember(hOutline, "Color", &hAttrMbr);
    printf("EssOtlFindMember() sts: %ld\n",sts);

    sts = EssOtlAssociateAttributeDimension(hOutline,hBaseMbr,hAttrMbr);
    printf("EssOtlAssociateAttributeDimension() sts: %ld\n",sts);

    sts = EssOtlWriteOutline(hOutline, &Object);
}```
printf("EssOtlWriteOutline() sts: %ld\n",sts);

sts = EssOtlRestructure(hCtx, ESS_DOR_ALLDATA);
printf("EssOtlRestructure() sts: %ld\n",sts);

if (!sts)
{
    sts = EssGetProcessState (hCtx, &pState);
    while (!sts || (pState.State != ESS_STATE_DONE))
    {
        sts = EssGetProcessState (hCtx, &pState);
    }
}
sts = EssOtlCloseOutline(hOutline);
printf("EssOtlCloseOutline() sts: %ld\n",sts);

See Also

- `EssCheckAttributes`
- `EssFreeStructure`
- `EssGetAssociatedAttributesInfo`
- `EssGetAttributeInfo`
- `EssGetAttributeSpecifications`
- `EssOtlAssociateAttributeMember`
- `EssOtlDisassociateAttributeDimension`
- `EssOtlDisassociateAttributeMember`
- `EssOtlFindAttributeMembers`
- `EssOtlFreeStructure`
- `EssOtlGetAssociatedAttributes`
- `EssOtlGetAttributeInfo`
- `EssOtlGetAttributeSpecifications`
- `EssOtlQueryAttributes`
- `EssOtlSetAttributeSpecifications`

### `EssOtlAssociateAttributeMember`

Associates an attribute member with a standard or base member.

**Syntax**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline;</td>
<td>ESS_HOUTLINE_T</td>
<td>Handle to the outline</td>
</tr>
<tr>
<td>hStandardMember;</td>
<td>ESS_HMEMBER_T</td>
<td>Handle to the standard or base member</td>
</tr>
<tr>
<td>hAttributeMember;</td>
<td>ESS_HMEMBER_T</td>
<td>Handle to the attribute member</td>
</tr>
</tbody>
</table>

**Notes**
Before you associate an attribute member with a standard or base member using this function, associate the dimension of the attribute member with the dimension of the standard or base member using `EssOtlAssociateAttributeDimension()`.

- You cannot associate an attribute member with a base dimension.
- Only a zero-level attribute member can associate with a standard or base member.
- You cannot associate members of a given attribute dimension with base members that are at different levels from each other.
- You cannot associate more than one member of an attribute dimension with a base member.
- You can associate members of more than one attribute dimension with a base member.

Example

```c
void ESS_OtlAssociateAttributeMember()
{
    ESS_STS_T       sts = ESS_STS_NOERR;
    ESS_HOUTLINE_T  hOutline;
    ESS_HMEMBER_T   hBaseMbr;
    ESS_HMEMBER_T   hAttrMbr;
    ESS_OBJDEF_T    Object;
    ESS_APPNAME_T   szAppName;
    ESS_DBNAME_T    szDbName;
    ESS_OBJNAME_T   szFileName;
    ESS_PROCSTATE_T pState;

    memset(&Object, '\0', sizeof(Object));
    Object.hCtx = hCtx;
    Object.ObjType = ESS_OBJTYPE_OUTLINE;
    strcpy(szAppName, "Sample");
    strcpy(szDbName, "Basic");
    strcpy(szFileName, "Basic");
    Object.AppName = szAppName;
    Object.DbName = szDbName;
    Object.FileName = szFileName;

    sts = EssOtlOpenOutline(hCtx, &Object, ESS_TRUE, ESS_TRUE, &hOutline);
    printf("EssOtlOpenOutline() sts: %ld\n",sts);

    sts = EssOtlFindMember(hOutline, "Product", &hBaseMbr);
    printf("EssOtlFindMember() sts: %ld\n",sts);

    sts = EssOtlFindMember(hOutline, "Color", &hAttrMbr);
    printf("EssOtlFindMember() sts: %ld\n",sts);

    sts = EssOtlAssociateAttributeMember(hOutline,hBaseMbr,hAttrMbr);
    printf("EssOtlAssociateAttributeMember() sts: %ld\n",sts);

    sts = EssOtlWriteOutline(hOutline, &Object);
    printf("EssOtlWriteOutline() sts: %ld\n",sts);

    sts = EssOtlRestructure(hCtx, ESS_DOR_ALLDATA);
    printf("EssOtlRestructure() sts: %ld\n",sts);

    if (!sts)
    {
        //
    }
}```
sts = EssGetProcessState (hCtx, &pState);
while (!sts || (pState.State != ESS_STATE_DONE))
    sts = EssGetProcessState (hCtx, &pState);
}
sts = EssOtlCloseOutline(hOutline);
printf("EssOtlCloseOutline() sts: %ld\n",sts);
}

See Also

- EssCheckAttributes
- EssFreeStructure
- EssGetAssociatedAttributesInfo
- EssGetAttributeInfo
- EssGetAttributeSpecifications
- EssOtlAssociateAttributeDimension
- EssOtlDisassociateAttributeDimension
- EssOtlDisassociateAttributeMember
- EssOtlFindAttributeMembers
- EssOtlFreeStructure
- EssOtlGetAssociatedAttributes
- EssOtlGetAttributeInfo
- EssOtlGetAttributeSpecifications
- EssOtlQueryAttributes
- EssOtlSetAttributeSpecifications

EssOtlClearAliasTable

Clears all entries from an existing alias table. The alias table is not deleted.

Syntax

ESS_FUNC_M EssOtlClearAliasTable (hOutline, pszAliasTable);

Parameter | Data Type  | Description
----------|------------|-------------
hOutline;  | ESS_HOUTLINE_T | Outline context handle.
pszAliasTable; | ESS_STR_T | Name of alias table to clear. Use ESS_NULL or "Default" for the default table.

Notes

When clearing aliases from an alias table, language codes associated with the alias table are removed.

Return Value

Returns 0 if successful; otherwise:

OTLAPI_BAD_ALIASTABLE
Example

```c
#include <essapi.h>
#include <essotl.h>

ESS_STS_T          sts = 0;
ESS_OBJDEF_T       Object;
ESS_HOUTLINE_T     hOutline;
ESS_APPNAME_T      szAppName;
ESS_DBNAME_T       szDbName;
ESS_OBJNAME_T      szFileName;

memset(&Object, '\0', sizeof(Object));
Object.hCtx = hCtx;
Object.ObjType = ESS_OBJTYPE_OUTLINE;
strcpy(szAppName, "Sample");
strcpy(szDbName, "Basic");
strcpy(szFileName, "Basic");
Object.AppName = szAppName;
Object.DbName = szDbName;
Object.FileName = szFileName;

sts = EssOtlOpenOutline(hCtx, &Object, ESS_TRUE,
                         ESS_TRUE, &hOutline);
if (!sts)
{
    sts = EssOtlClearAliasTable(hOutline,
                                 "Default");
}
```

See Also

- EssOtlCreateAliasTable
- EssOtlCopyAliasTable
- EssOtlRenameAliasTable
- EssOtlDeleteAliasTable
- EssOtlSetAliasTableLanguage

**EssOtlClearAliasTableLanguages**

Clears the set of language codes associated with the specified alias table.

**Syntax**

```c
ESS_FUNC_M EssOtlClearAliasTableLanguages (hOutline, pszAliasTable);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline context handle.</td>
</tr>
<tr>
<td>pszAliasTable</td>
<td>ESS_STR_T</td>
<td>Name of the alias table from which to remove all associated language codes.</td>
</tr>
</tbody>
</table>

**Return Value**

- If successful, returns 0.
If unsuccessful, returns the error OTLAPI_BAD_ALIAS_TABLE (invalid alias table).

Access

This function does not require special privileges.

Example

```c
#include <essapi.h>
#include <essotl.h>

ESS_STS_T           sts = 0;
ESS_OUTLINEINFO_T   NewInfo;
ESS_HOUTLINE_T      hOutline;
ESS_PALIASLANG_T    pLangs=ESS_NULL;
ESS_ULONG_T         nLangs = 0, i=0;

memset(&NewInfo, '\0', sizeof(NewInfo));
sts = EssOtlNewOutline(hCtx, &NewInfo, &hOutline);

if (!sts)
{
    sts = EssOtlCreateAliasTable(hOutline,
        "French Alias Table");
}

if (!sts)
{
    sts = EssOtlSetAliasTableLanguage (hOutline,
        "French Alias Table", "fr");
}

if (!sts)
{
    sts = EssOtlSetAliasTableLanguage (hOutline,
        "French Alias Table", "fr-CA");
}

if (!sts)
{
    sts = EssOtlGetAliasTableLanguages(hOutline, "French Alias Table", &nLangs,
        &pLangs);

    if ( !sts == ESS_STS_NOERR && ( pLangs ) )
    {
        for (i=0;i<nLangs ;++i)
        {
            if (pLangs[i])
            {
                printf("Language Code:  %s\n", pLangs[i]);
            }
        }
    }

    EssFree(hInst, pLangs);
}
if (!sts)
{
    sts = EssOtlClearAliasTableLanguages (hOutline,
        "French Alias Table");
}
```
See Also

- EssOtlGetAliasTableLanguages
- EssOtlSetAliasTableLanguage

**EssOtlCloseOutline**

Frees all information associated with the outline.

**Syntax**

```c
ESS_FUNC_M EssOtlCloseOutline (hOutline);
```

**Parameter**  **Data Type**  **Description**

- hOutline  ESS_HOUTLINE_T  Outline context handle.

**Notes**

- This function should always be called if EssOtlNewOutline() or EssOtlOpenOutline() is called.
- If the object was locked when it was opened, you should call EssUnlockObject() before making this call.

**Return Value**

Returns 0 if successful.

**Example**

```c
#include <essapi.h>
#include <essotl.h>

ESS_STS_T          sts = 0;
ESS_OBJDEF_T       Object;
ESS_HOUTLINE_T     hOutline;
ESS_APPNAME_T      szAppName;
ESS_DBNAME_T       szDbName;
ESS_OBJNAME_T      szFileName;
memset(&Object, '\0', sizeof(Object));
Object.hCtx = hCtx;
Object.ObjType = ESS_OBJTYPE_OUTLINE;
strcpy(szAppName, "Sample");
strcpy(szDbName, "Basic");
strcpy(szFileName, "Basic");
Object.AppName = szAppName;
Object.DbName = szDbName;
Object.FileName = szFileName;

sts = EssOtlOpenOutline(hCtx, &Object, ESS_TRUE, ESS_TRUE, &hOutline);
```
/* body of code */
if (!sts)
{
    sts = EssOtlWriteOutline(hOutline, &Object);
}

/* restructure db using EssOtlRestructure() */
if (!sts)
{
    sts = EssOtlCloseOutline(hOutline);
}

See Also

- EssOtlOpenOutline
- EssOtlWriteOutline
- EssOtlRestructure

**EssOtlCompactOutline**

Compacts an outline file that requires compacting at the client side.

**Syntax**

```c
ESS_FUNC_M EssOtlCompactOutline (hCtx, filename);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context acquired during login</td>
</tr>
<tr>
<td>filename</td>
<td>ESS_STR_T</td>
<td>Path and outline file to be compacted</td>
</tr>
</tbody>
</table>

**Return Value**

Returns 0 if successful. The compacted file is named the same with extension .otn and is available in the path specified.

**Example**

```c
#include <windows.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>

#pragma pack(push, api, 1)
#include <essapi.h>
#include <essotl.h>
#pragma pack(pop, api)

/* default names */
ESS_SVRNAME_T srvrName = "localhost";
ESS_USERNAME_T userName = "essexer";
ESS_PASSWORD_T pswd = "password";
ESS_APPNAME_T app = "ASOSamp";
ESS_DBNAME_T db = "Sample";
```
```c
int main(int argc, char *argv[]) {
    ESS_STS_T sts = ESS_STS_NOERR;
    ESS_HINST_T hInst = NULL;
    ESS_HOUTLINE_T hOutlineQuery = NULL, hOutline = NULL;
    ESS_HCTX_T hCtx = NULL;
    ESS_USHORT_T Items;
    ESS_PAPPDB_T pAppsDbs = NULL;
    ESS_ACCESS_T Access;
    ESS_INIT_T InitStruct = /* Define init */
        /* structure */
    {
        ESS_API_VERSION, /* Version of API */
        (ESS_PVOID_T)0,    /* user-defined message context */
        0,               /* max handles */
        0L,              /* max buffer size */
        NULL, //((ESS_STR_T)"C:\Hyperion\AnalyticServices", )/* local
        path */
        /* The following parameters use defaults */
        NULL, /* message db path */
        NULL, /* allocation function pointer */
        NULL, /* reallocation function pointer */
        NULL, /* free function pointer */
        NULL, //((ESS_PFUNC_T)MessageFunc, /* error handling function
        pointer */
        NULL, /* path name of user-defined */
        /* Application help file */
        0L     /* Reserved for internal use. */
    }/* Set to NULL */
    #ifdef AD_UTF8
        , ESS_API_UTF8
    #endif
    if (argc < 6) {
        puts(" Usage: EssCompactOtl ServerName Userid Password AppName DbName
\n");
        exit (0);
    }
    strcpy(srvrName, argv[1]);
    strcpy(userName, argv[2]);
    strcpy(pswd, argv[3]);
    strcpy(app, argv[4]);
    strcpy(db, argv[5]);
    /* Initialize the Essbase API */
    if ((sts = EssInit(&InitStruct, &hInst)) != ESS_STS_NOERR)
        {printf("EssInit failure: %ld\n", sts);
            exit ((int) sts);
    }
    /* Login to Essbase */
```

if ((sts = EssLogin (hInst, srvrName, userName, pswd, &Items, &pAppsDbs, &hCtx)) != ESS_STS_NOERR) {
    printf("EssLogin failure: %ld\n", sts);
    exit ((int) sts);
}

if(pAppsDbs)    
    EssFree(hInst, pAppsDbs);

/* Select the application */
if (((sts = EssSetActive(hCtx, app, db, &Access)) != ESS_STS_NOERR)) {
    printf("EssSetActive failure: %ld\n", sts);
    exit ((int) sts);
}

/* compact the outline and restructure */
if (((sts = EssCompactOutline(hCtx)) != ESS_STS_NOERR)) {
    printf("EssCompactOutline failure: %ld\n", sts);
    exit ((int) sts);
}

/* done, logout and terminate the api */
if (((sts = EssLogout (hCtx)) != ESS_STS_NOERR)) {
    printf("EssLogout failure: %ld\n", sts);
    exit ((int) sts);
}

if (((sts = EssTerm(hInst)) != ESS_STS_NOERR)) {
    /* error terminating API */
    exit((int) sts);
}

return(0);


---

**EssOtlCopyAliasTable**

Copies an alias table to another alias table.

**Syntax**

```c
ESS_FUNC_M EssOtlCopyAliasTable (hOutline, pszSourceAliasTable, pszDestAliasTable, fMerge);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline context handle.</td>
</tr>
<tr>
<td>pszSourceAliasTable</td>
<td>ESS_STR_T</td>
<td>Name of alias table to copy from. If this parameter is ESS_NULL, the default alias table is used.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>pszDestAliasTable</td>
<td>ESS_STR_T</td>
<td>Name of alias table to copy to. Cannot be the same as pszSourceAliasTable.</td>
</tr>
<tr>
<td>fMerge</td>
<td>ESS_BOOL_T</td>
<td>Set to ESS_TRUE to merge the source file into the existing destination alias table. Set to ESS_FALSE to clear the destination alias table before copying.</td>
</tr>
</tbody>
</table>

**Notes**

- If the destination alias table does not exist, it is created. If the destination alias table exists, it is cleared first, unless the fMerge flag is set to ESS_TRUE.
- The maximum number of alias tables in a single block storage or aggregate storage database outline (including the default table) is 32.
- When copying an alias table, language codes associated with the alias table are removed from the copied alias table.

**Return Value**

Returns 0 if successful; otherwise one of the following:

- OTLAPI_BAD_ALIASTABLE
- OTLAPI_ERR_MAXALIASTABLES
- OTLAPI_ERR_ALIASTABLENAME
- OTLAPI_ERR_COPYALIASTABLE: Source and destination tables are the same.

**Example**

```c
#include <essapi.h>
#include <essotl.h>

ESS_STS_T          sts = 0;
ESS_HOUTLINE_T     hOutline;
ESS_OBJDEF_T       Object;
ESS_APPNAME_T      szAppName;
ESS_DBNAME_T       szDbName;
ESS_OBJNAME_T      szFileName;
memset(&Object, '\0', sizeof(Object));
Object.hCtx = hCtx;
Object.ObjType = ESS_OBJTYPE_OUTLINE;
strcpy(szAppName, "Sample");
strcpy(szDbName, "Basic");
strcpy(szFileName, "Basic");
Object.AppName = szAppName;
Object.DbName = szDbName;
Object.FileName = szFileName;

sts = EssOtlOpenOutline(hCtx, &Object, ESS_TRUE, ESS_TRUE, &hOutline);
if (!sts)
{
    sts = EssOtlCopyAliasTable(hOutline, ESS_NULL, "Alias Table 2", ESS_TRUE);
}
```
EssOtlCreateAliasTable

Creates an empty alias table in the outline.

Syntax

```
ESS_FUNC_M EssOtlCreateAliasTable (hOutline, pszAliasTable);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline context handle.</td>
</tr>
<tr>
<td>pszAliasTable</td>
<td>ESS_STR_T</td>
<td>Name of alias table to create.</td>
</tr>
</tbody>
</table>

Notes

- An alias table named “Default” cannot be created, since the default alias table always exists.
- The maximum number of alias tables in a single block storage or aggregate storage database outline (including the default table) is 32.
- You can specify multiple language codes for an alias table, using the `EssOtlSetAliasTableLanguage` API. When you create an alias table, a language code is not specified.

Return Value

Returns 0 if successful; otherwise one of the following:

- OTLAPI_ERR_ALIASTABLEEXISTS
- OTLAPI_ERR_MAXALIASTABLES
- OTLAPI_ERR_ALIASTABLENAME

Example

```
#include <essapi.h>
#include <essotl.h>

ESS_STS_T          sts = 0;
ESS_OUTLINEINFO_T  NewInfo;
ESS_HOUTLINE_T     hOutline;

memset(&NewInfo, '\0', sizeof(NewInfo));
sts = EssOtlNewOutline(hCtx, &NewInfo, &hOutline);
if (!sts)
{
    sts = EssOtlCreateAliasTable(hOutline,
```


"Alias Table 1");
}

See Also

- `EssOtlCopyAliasTable`
- `EssOtlRenameAliasTable`
- `EssOtlDeleteAliasTable`
- `EssOtlSetAliasTableLanguage`

## EssOtlCreateObject

Creates an object of the specified object type and name and returns the object handle.

### Syntax

```
ESS_FUNC_M EssOtlCreateObject (hOutline, objType, name, phObjHandle)
```

### Parameter Data Type Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline handle (Edit mode only)</td>
</tr>
<tr>
<td>objType</td>
<td>ESS_OBJECT_TYPES</td>
<td>Object type with one of the following values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• OBJECT_SMARTLIST</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Object type is Text List (SmartList)/</td>
</tr>
<tr>
<td>name</td>
<td>ESS_STR_T</td>
<td>String identifying the object</td>
</tr>
<tr>
<td>phObjHandle</td>
<td>ESS_PHOBJECT_T</td>
<td>Returns the created object handle.</td>
</tr>
</tbody>
</table>

### Return Value

Returns:

- 0—If successful
  
  Object created with handle in `phObjHandle`

- Error number—If unsuccessful
  
  No object created, and `phObjHandle` is NULL.

- OTLAPI_ERR_OBJTYPE_NOTSUPPORTED
  
  If invalid object type is passed.

### Example

```c
void TestCreateObject()
{
    ESS_STS_T            sts = ESS_STS_NOERR;
    ESS_HOUTLINE_T       hOutline = ESS_NULL;
    ESS_OBJDEF_T         Object;
    ESS_OBJECT_TYPES     objType;
    ESS_STR_T            smartListName;
    ESS_HOBJECT_T        ObjHandle;
    ESS_ULONG_T          Count, i;
```
See Also

- EssOtlGetMemberSmartList
EssOtlDeleteAliasTable

Deletes the specified alias table from the outline, clearing all of its entries.

Syntax

```c
ESS_FUNC_M EssOtlDeleteAliasTable (hOutline, pszAliasTable);
```

Parameter | Data Type | Description |
--- | --- | --- |
hOutline | ESS_HOUTLINE_T | Outline context handle. |
pszAliasTable | ESS_STR_T | Name of alias table to delete. |

Notes

You cannot delete the default alias table.

Return Value

Returns 0 if successful; otherwise one of the following:

- OTLAPI_BAD_ALIASTABLE
- OTLAPI_ERR_DELETEDEFALIAS

Example

```c
#include <essapi.h>
#include <essotl.h>

ESS_STS_T          sts = 0;
ESS_HOUTLINE_T     hOutline;
ESS_OBJDEF_T       Object;
ESS_APPNAME_T      szAppName;
ESS_DBNAME_T       szDbName;
ESS_OBJNAME_T      szFileName;
memset(&Object, '\0', sizeof(Object));
```
Object.hCtx = hCtx;
Object.ObjType = ESS_OBJTYPE_OUTLINE;
strcpy(szAppName, "Sample");
strcpy(szDbName, "Basic");
strcpy(szFileName, "Basic");
Object.AppName = szAppName;
Object.DbName = szDbName;
Object.FileName = szFileName;

sts = EssOtlOpenOutline(hCtx, &Object, ESS_TRUE, ESS_TRUE, &hOutline);

if (!sts)
{
    sts = EssOtlDeleteAliasTable(hOutline, " Alias Table 1");
}

See Also

- EssOtlCreateAliasTable
- EssOtlCopyAliasTable
- EssOtlRenameAliasTable
- EssOtlClearAliasTable

EssOtlDeleteDimension

Deletes a dimension from the outline. The call also specifies a member of the dimension being deleted from which to keep data when the outline is restructured.

Syntax

ESS_FUNC_M EssOtlDeleteDimension (hOutline, hMember, pszDataMbr);

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline context handle.</td>
</tr>
<tr>
<td>hMember</td>
<td>ESS_HMEMBER_T</td>
<td>Handle of member to delete.</td>
</tr>
<tr>
<td>pszDataMbr</td>
<td>ESS_STR_T</td>
<td>Member name in the dimension to be deleted from which data will be saved when the outline is restructured. If this field is ESS_NULL, the dimension is used.</td>
</tr>
</tbody>
</table>

Notes

- All shared members of the dimension and its descendants are deleted.
- All members of the dimension are deleted.
- To delete a dimension, you can use this call or EssOtlDeleteMember(). EssOtlDeleteDimension() gives you the benefit of selecting a member of the deleted dimension whose data values will be used as the data values for the other dimensions when the database is restructured. If EssOtlDeleteMember() is used, the data values of the top member (dimension) of the deleted dimension are used.
In order for the `pszDataMbr` field to take effect, the outline must have been opened with `EssOtlOpenOutline()` with the `fKeepTrans` flag set to ESS_TRUE.

**Return Value**

Returns 0 if successful; otherwise:

- OTLAPI_ERR_ADDDELETEDIMDYNAMICCALC
- OTLAPI_ERR_NOTIMEDIM

**Example**

```c
#include <essapi.h>
#include <essotl.h>

ESS_STS_T          sts = 0;
ESS_OBJDEF_T       Object;
ESS_HOUTLINE_T     hOutline;
ESS_HMEMBER_T      hMemberScenario;
ESS_APPNAME_T      szAppName;
ESS_DBNAME_T       szDbName;
ESS_OBJNAME_T      szFileName;

memset(&Object, '\0', sizeof(Object));
Object.hCtx = hCtx;
Object.ObjType = ESS_OBJTYPE_OUTLINE;
strcpy(szAppName, "Sample");
strcpy(szDbName, "Basic");
strcpy(szFileName, "Basic");
Object.AppName = szAppName;
Object.DbName = szDbName;
Object.FileName = szFileName;

sts = EssOtlOpenOutline(hCtx, &Object, ESS_TRUE, ESS_TRUE, &hOutline);
if (!sts)
{
    sts = EssOtlFindMember(hOutline, "Scenario", &hMemberScenario);
}
if (!sts && hMemberScenario)
{
    sts = EssOtlDeleteDimension(hOutline, hMemberScenario, "Actual");
}
```

**See Also**

- `EssOtlDeleteMember`
- `EssOtlAddDimension`
- `EssOtlAddMember`
- `EssOtlFindMember`
- `EssOtlGetMemberInfo`
EssOtlDeleteDTSMemberAlias

Deletes an alias name for a Dynamic Time Series (DTS) member.

Syntax

```
ESS_STS_T EssOtlDeleteDTSMemberAlias (hOutline, pszDTSMember, pszAliasTable);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>The Essbase outline handle returned from the EssOtlOpenOutline call.</td>
</tr>
<tr>
<td>pszDTSMember</td>
<td>ESS_STR_T</td>
<td>Name of the DTS member which provides the alias.</td>
</tr>
<tr>
<td>pszAliasTable</td>
<td>ESS_STR_T</td>
<td>Name of the alias table which provides the alias. If NULL, use the default alias table.</td>
</tr>
</tbody>
</table>

**Return Value**

If successful the return value is zero. Otherwise, one of the following is returned:

- OTLAPI_ERR_DTSMBRNOTDEFINED
- OTLAPI_BAD_ALIASTABLE
- OTLAPI_ERR_NOALIAS

**Example**

```c
#include "essapi.h"
#include "essotl.h"
#include "esserror.h"

ESS_STS_T ESS_OtlDeleteDTSMemberAlias(ESS_HCTX_T hCtx)
{
    ESS_STS_T               sts = ESS_STS_NOERR;
    ESS_OBJDEF_T    Object;
    ESS_HOUTLINE_T  hOutline;
    ESS_APPNAME_T   szAppName;
    ESS_DBNAME_T    szDbName;
    ESS_OBJNAME_T   szFileName;
    ESS_CHAR_T      pszAliasTable[ESS_ALIASNAMELEN];
    ESS_CHAR_T      pszDTSMember[ESS_MBRNAMELEN];
    ESS_CHAR_T      pszDTSMember[ESS_MBRNAMELEN];
    ESS_PROCSTATE_T pState;
    ESS ULONG_T     ulErrors;
    ESS ULONG_T     ulCount;
    ESS POUTERROR_T pMbrErrors = NULL;

    strcpy(szAppName, "sample");
    strcpy(szDbName, "Basic");
    strcpy(szFileName, "Basic");
    strcpy(pszAliasTable, "Default");
    strcpy(pszDTSMember, "Q-T-D");
    strcpy(pszDTSMember, "Q-T-D");

    Object.hCtx = hCtx;
    Object.ObjType = ESS_OBJTYPE_OUTLINE;
    Object.AppName = szAppName;
    Object.DbName = szDbName;
    Object.FileName = szFileName;
```
sts = EssOtlOpenOutline(hCtx, &Object, ESS_TRUE, ESS_TRUE, &hOutline);

if(sts)
{
    printf("Could not open outline\n");
    return sts;
}

sts = EssOtlDeleteDTSMemberAlias(hOutline, pszDTSMember, pszAliasTable);
if(sts)
{
    printf("Could not get DTS member alias\n");
    return sts;
}

sts = EssOtlWriteOutline(hOutline, &Object);
if(sts)
{
    printf("Could not write outline\n");
    return sts;
}

sts = EssOtlRestructure(hCtx, ESS_DOR_ALLDATA);
if(sts)
{
    printf("Could not restructure outline\n");
    return sts;
}

memset (&pState, 0, sizeof(ESS_PROCSTATE_T));
sts = EssGetProcessState(hCtx, &pState);
{
    while ((sts == ESS_STS_NOERR ) && (pState.State != ESS_STATE_DONE))
    {
        memset (&pState, 0, sizeof(ESS_PROCSTATE_T));
        sts = EssGetProcessState(hCtx, &pState);
    }
}

sts = EssUnlockObject(hCtx, ESS_OBJTYPE_OUTLINE, szAppName, szDbName, szFileName);
if (sts)
{
    printf("Could not unlock outline\n");
    return sts;
}

EssOtlCloseOutline(hOutline);
return sts;

See Also
● EssOtlEnableDTSMember
● EssOtlGetEnabledDTSMembers
● EssOtlGetDTSMemberAlias
EssOtlDeleteGenName

Deletes the name of a specific generation within a dimension.

Syntax

```c
ESS_FUNC_M EssOtlDeleteGenName (hOutline, pszDimension, usGen);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline context handle.</td>
</tr>
<tr>
<td>pszDimension</td>
<td>ESS_STR_T</td>
<td>Name of the dimension that contains the generation.</td>
</tr>
<tr>
<td>usGen</td>
<td>ESS_USHORT_T</td>
<td>Number of generation for which to delete name. Leaf members are level 0.</td>
</tr>
</tbody>
</table>

**Return Value**

Returns 0 if successful; otherwise one of the following:

- OTLAPI_NO_GENLEVELNAME
- OTLAPI_ERR_NOTADIM

**Example**

```c
#include <essapi.h>
#include <essotl.h>

ESS_STS_T          sts = 0;
ESS_HOUTLINE_T     hOutline;
ESS_OBJDEF_T       Object;
ESS_APPNAME_T      szAppName;
ESS_DBNAME_T       szDbName;
ESS_OBJNAME_T      szFileName;
ESS_STR_T          Dimension;
ESS_USHORT_T       GenNum;
memset(&Object, '\0', sizeof(Object));
Object.hCtx = hCtx;
Object.ObjType = ESS_OBJTYPE_OUTLINE;
strcpy(szAppName, "Sample");
strcpy(szDbName, "Basic");
strcpy(szFileName, "Basic");
Object.AppName = szAppName;
Object.DbName = szDbName;
Object.FileName = szFileName;
sts = EssOtlOpenOutline(hCtx, &Object, ESS_TRUE,
                        ESS_TRUE, &hOutline);
/******************** Delete Generation Name ********************/
Dimension = "Year";
GenNum = 2;
if (!sts)
{
    sts = EssOtlDeleteGenName(hOutline, Dimension,
```
EssOtlDeleteLevelName

Deletes the name for a specific level within a dimension.

**Syntax**

```c
ESS_FUNC_M EssOtlDeleteLevelName (hOutline, pszDimension, usLevel);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline context handle.</td>
</tr>
<tr>
<td>pszDimension</td>
<td>ESS_STR_T</td>
<td>Name of the dimension that contains the level name.</td>
</tr>
<tr>
<td>usLevel</td>
<td>ESS_USHORT_T</td>
<td>Number of level for which to delete name. Leaf members are level 0.</td>
</tr>
</tbody>
</table>

**Notes**

In C programs, call EssFree() to free the returned buffer.

**Return Value**

Returns 0 if successful; otherwise one of the following:

- OTLAPI_NO_GENLEVELNAME
- OTLAPI_ERR_NOTADIM

**Example**

```c
#include <essapi.h>
#include <essotl.h>

ESS_STS_T          sts = 0;
ESS_HOUTLINE_T     hOutline;
ESS_OBJDEF_T       Object;
ESS_APPNAME_T      szAppName;
ESS_DBNAME_T       szDbName;
ESS_OBJNAME_T      szFileName;
ESS_STR_T          Dimension;
ESS_USHORT_T       LevelNum;

memset(&Object, '\0', sizeof(Object));
Object.hCtx = hCtx;
Object.ObjType = ESS_OBJTYPE_OUTLINE;
strcpy(szAppName, "Sample");
strcpy(szDbName, "Basic");
strcpy(szFileName, "Basic");
Object.AppName = szAppName;
Object.DbName = szDbName;
Object.FileName = szFileName;
```
sts = EssOtlOpenOutline(hCtx, &Object, ESS_TRUE, ESS_TRUE, &hOutline);

/MMM**********  **Delete Level Name** **********/
Dimension = "Year";
LevelNum = 2;
if (!sts)
{
    sts = EssOtlDeleteLevelName(hOutline,
                                Dimension, LevelNum);
}

See Also

- EssOtlGetLevelName
- EssOtlSetLevelName

**EssOtlDeleteObject**

Deletes the object passed.

**Syntax**

ESS_FUNC_M EssOtlDeleteObject (hOutline, objHandle)

**Parameter** | **Data Type** | **Description**
--- | --- | ---
| hOutline | ESS_HOUTLINE_T | Outline handle (Edit mode only)
| objHandle | ESS_HOBJECT_T | Object to be deleted

**Notes**

You cannot delete objects with existing associations. With Text List objects (SmartList objects), you cannot delete the SmartList object without removing references—use the Get Object References API to do so.

**Return Value**

Returns:

- 0—if successful
- Error number—if unsuccessful

**Example**

void TestCreateObject()
{
    ESS_STS_T         sts = ESS_STS_NOERR;
    ESS_HOUTLINE_T    hOutline = ESS_NULL;
    ESS_OBJDEF_T      Object;
    ESS_OBJECT_TYPES  objType;
    ESS_STR_T         smartListName;
    ESS_HOBJECT_T     ObjHandle;
    ESS_ULONG_T       Count, i;
memset(&Object, '\0', sizeof(Object));
Object.hCtx = hCtx;
Object.ObjType = ESS_OBJTYPE_OUTLINE;
Object.AppName = szAppName;
Object.DbName = szDbName;
Object.FileName = szFileName;

/* Open outline */
sts = EssOtlOpenOutline(hCtx, &Object,
                   ESS_TRUE, ESS_TRUE, &hOutline);

/* Create a static SmartList */
objType = OBJECT_SMARTLIST;
smartListName = "SList1";
sts = EssOtlCreateObject(hOutline, objType,
                          smartListName, &ObjHandle);

/* List all SmartList objects */
objType = OBJECT_SMARTLIST;
sts = EssOtlListObjects(hOutline, objType,
                         &Count, &ObjHandles);

/* Save */
SaveOutline(hOutline);

/* Find objects */
objName = "SList1";
sts = EssOtlFindObject(hOutline, objType, objName,
                       &hObjHandle);

/* Delete objects */
hSmartList = (ESS_HSMARTLIST_T)hObjHandle;
sts = EssOtlDeleteObject(hOutline, hSmartList);
SaveOutline(hOutline);

if(ObjHandles)
   EssFree (hInst, ObjHandles);

/* Unlock objects */
sts = EssUnlockObject(hCtx, Object.ObjType,
                      Object.AppName, Object.DbName, Object.FileName);

/* Close outline */
sts = EssOtlCloseOutline(hOutline);
}

See Also
- EssOtlGetMemberSmartList
- EssOtlCreateObject
- EssOtlDeleteObject
- EssOtlGetSmartListInfo
EssOtlFindObject
EssOtlFreeObjectArray
EssOtlFreeSmartListInfo
EssOtlGetMemberSmartList
EssOtlGetMemberType
EssOtlGetObjectReferenceCount
EssOtlGetObjectReferences
EssOtlImportExportObject
EssOtlListObjects
EssOtlQueryObjects
EssOtlSetMemberType
EssOtlSetMemberTypeToSmartList

EssOtlDeleteMember

Deletes a member from the outline.

Syntax

```c
ESS_FUNC_M EssOtlDeleteMember (hOutline, hMember);
```

Parameter | Data Type          | Description
----------|-------------------|-------------
`hOutline` | ESS_HOUTLINE_T    | Outline context handle.
`hMember`  | ESS_HMEMBER_T     | Handle of member to delete.

Notes

- All descendants of the member are deleted.
- All shared members of this member and its descendants are deleted.
- If a shared member, only the specified member is deleted.
- To delete a dimension, you can use this call or `EssOtlDeleteDimension()`.
  `EssOtlDeleteDimension()` gives you the benefit of selecting a member of the deleted dimension whose data values will be used as the data values for the other dimensions when the database is restructured. If `EssOtlDeleteMember()` is used, the data values of the top member (dimension) of the deleted dimension are used.

Return Value

Returns 0 if successful; otherwise one of the following:

- OTLAPI_ERR_LEAFLABEL
- OTLAPI_ERR_NOTIMEDIM

Example

```c
#include <essapi.h>
#include <essotl.h>

ESS_STS_T          sts = 0;
ESS_OBJDEF_T       Object;
```
memset(&Object, '\0', sizeof(Object));
Object.hCtx = hCtx;
Object.ObjType = ESS_OBJTYPE_OUTLINE;
strcpy(szAppName, "Sample");
strcpy(szDbName, "Basic");
strcpy(szFileName, "Basic");
Object.AppName = szAppName;
Object.DbName = szDbName;
Object.FileName = szFileName;

sts = EssOtlOpenOutline(hCtx, &Object, ESS_TRUE, ESS_TRUE, &hOutline);
if (!sts)
{
    sts = EssOtlFindMember(hOutline, "COGS", &hCOGS);
}
if (!sts && hCOGS)
{
    sts = EssOtlDeleteMember(hOutline, hCOGS);
}

See Also

- EssOtlDeleteDimension
- EssOtlAddMember
- EssOtlAddDimension
- EssOtlFindMember
- EssOtlGetMemberInfo

**EssOtlDeleteMemberAlias**

Deletes the default member alias for a specified member in a specified alias table.

**Syntax**

```c
ESS_FUNC_M EssOtlDeleteMemberAlias(hOutline, hMember, pszAliasTable);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline context handle.</td>
</tr>
<tr>
<td>hMember</td>
<td>ESS_HMEMBER_T</td>
<td>Handle of member to delete the alias from.</td>
</tr>
<tr>
<td>pszAliasTable</td>
<td>ESS_STR_T</td>
<td>Alias table to delete the alias from. If this parameter is ESS_NULL, the default table is used.</td>
</tr>
</tbody>
</table>
Return Value

Returns 0 if successful; otherwise:

OTLAPI_ERR_NOALIAS

Example

#include <essapi.h>
#include <essotl.h>

ESS_STS_T         sts = 0;
ESS_HOUTLINE_T    hOutline;
ESS_HMEMBER_T     hMemberJan;
ESS_OBJDEF_T       Object;
ESS_APPNAME_T     szAppName;
ESS_DBNAME_T      szDbName;
ESS_OBJNAME_T     szFileName;

memset(&Object, '\0', sizeof(Object));
Object.hCtx = hCtx;
Object.ObjType = ESS_OBJTYPE_OUTLINE;
strncpy(szAppName, "Sample");
strncpy(szDbName, "Basic");
strncpy(szFileName, "Basic");
Object.AppName = szAppName;
Object.DbName = szDbName;
Object.FileName = szFileName;
sts = EssOtlOpenOutline(hCtx, &Object, ESS_TRUE,
                        ESS_TRUE, &hOutline);
if (!sts)
{
    sts = EssOtlFindMember(hOutline, "Jan",
                          &hMemberJan);
}
if (!sts && hMemberJan)
{
    sts = EssOtlDeleteMemberAlias(hOutline,
                                 hMemberJan, ESS_NULL);
}

See Also

- EssOtlGetMemberAlias
- EssOtlSetMemberAlias

EssOtlDeleteMemberFormula

Deletes the formula for the specified member.

Syntax

ESS_FUNC_M EssOtlDeleteMemberFormula (hOutline, hMember);

Parameter  Data Type      Description

hOutline   ESS_HOUTLINE_T  Outline context handle.
Parameter | Data Type | Description
--- | --- | ---
hMember | ESS_HMEMBER_T | Member handle.

**Return Value**

Returns 0 if successful; otherwise:

OTLAPI_ERR_NOFORMULA

**Example**

```c
#include <essapi.h>
#include <essotl.h>

ESS_STS_T         sts = 0;
ESS_HOUTLINE_T     hOutline;
ESS_HMEMBER_T      hMember;
ESS_OBJDEF_T       Object;
ESS_APPNAME_T      szAppName;
ESS_DBNAME_T       szDbName;
ESS_OBJNAME_T      szFileName;
memset(&Object, '\0', sizeof(Object));
Object.hCtx = hCtx;
Object.ObjType = ESS_OBJTYPE_OUTLINE;
strcpy(szAppName, "Sample");
strcpy(szDbName, "Basic");
strcpy(szFileName, "Basic");
Object.AppName = szAppName;
Object.DbName = szDbName;
Object.FileName = szFileName;

sts = EssOtlOpenOutline(hCtx, &Object, ESS_TRUE,
                         ESS_TRUE, &hOutline);

if (!sts)
{
    sts = EssOtlFindMember(hOutline,
                           "Variance", &hMember);
}

if (!sts && hMember)
{
    sts = EssOtlDeleteMemberFormula(hOutline,
                                     hMember);
}
```

**See Also**

- EssOtlSetMemberFormula
- EssOtlGetMemberFormula

**EssOtlDeleteQueryHint**

Deletes the query hint indicated by the input outline and hint number.
Hints are numbered from 1 to \( n \). This function deletes the specified query hint and decreases the number of hints with one. All hints with a \( \text{hintNum} \) greater than the deleted hint are renumbered to \( \text{hintNum} - 1 \).

**Syntax**

```c
ESS_FUNC_M EssOtlDeleteQueryHint (hOutline, hintNum);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline context handle (input).</td>
</tr>
<tr>
<td>hintNum</td>
<td>ESS_SHORT_T</td>
<td>Query hint number (input).</td>
</tr>
</tbody>
</table>

**Notes**

- Query hints enable you to influence normal view selection by informing Essbase about the profile of common queries.
- This function is applicable only to Release 9.3 or higher aggregate storage databases.

**Return Value**

Returns 0 if successful.

**Example**

```c
ESS_STS_T         sts = ESS_STS_NOERR;
ESS_HOUTLINE_T    hOutline = ESS_NULL;
ESS_PMBRINFO_T    pMemberInfo = ESS_NULL;
ESS_SHORT_T       nmHints = 0;
ESS_SHORT_T       i, j, hintNum;
ESS_HMEMBER_T     hMember[10]; /* (nm real dimensions) < 10 */

/* Code to assign hOutline variable omitted */
/* Code to assign hintNum variable omitted */

sts = EssOtlGetNumQueryHints(hOutline, &nmHints);
if (sts) return sts; /* error out */

if (hintNum <= nmHints)
{
    sts = EssOtlDeleteQueryHint(hOutline, hintNum);
    if (sts)
        printf("Error [%s] deleting query hint (%d)\n", sts, hintNum);
    else
        printf("Query-Hint number: (%d) deleted\n", hintNum);
}
else
{
    printf("Query-Hint number: (%d) does not exist\n", hintNum);
}
```

**See Also**

- EssOtlAddQueryHint
- EssOtlSetQueryHint
- EssOtlSetQueryHint
EssOtlDeleteUserAttribute

Deletes a user-defined attribute of a member.

Syntax

\[
\text{ESS\_FUNC\_M} \quad \text{EssOtlDeleteUserAttribute} \quad (\text{hOutline, hMember, pszString});
\]

Parameter | Data Type | Description
--- | --- | ---
| hOutline | ESS\_HOUTLINE\_T | Outline context handle |
| hMember | ESS\_HMEMBER\_T | Handle of member that contains the attribute you are deleting |
| pszString | ESS\_STR\_T | User attribute string. |

Notes

The caller passes in a string to identify the attribute.

Return Value

Returns 0 if successful; otherwise:

OTLAPI\_NO\_USERATTR.

Example

```
#include <essapi.h>
#include <essotl.h>

ESS\_STS\_T \hspace{1em} \text{sts = 0};
ESS\_HOUTLINE\_T \hspace{1em} \text{hOutline};
ESS\_OBJDEF\_T \hspace{1em} \text{Object};
ESS\_APPNAME\_T \hspace{1em} \text{szAppName};
ESS\_DBNAME\_T \hspace{1em} \text{szDbName};
ESS\_HMEMBER\_T \hspace{1em} \text{szFileName};
ESS\_STR\_T \hspace{1em} \text{AttributeList};

memset(&Object, '\0', sizeof(Object));
Object.hCtx = hCtx;
Object.ObjType = ESS\_OBJTYPE\_OUTLINE;
strcpy(szAppname, "Sample");
strcpy(szDbName, "Basic");
strcpy(szFileName, "Basic");
Object.AppName = szAppName;
Object.DbName = szDbName;
Object.FileName = szFileName;

sts = EssOtlOpenOutline(hCtx, &Object, ESS\_TRUE, ESS\_TRUE, \&hOutline);

/******* Delete User Attributes *******/
```
AttributeList = "Read Write";

if (!sts)
{
    sts = EssOtlFindMember(hOutline, "Jan",
                         &hMember);
}

if (!sts && hMember)
{
    sts = EssOtlDeleteUserAttribute(hOutline,
                                     hMember, AttributeList);
}

See Also

- EssOtlGetUserAttributes
- EssOtlSetUserAttribute

EssOtlDetailQueryAttributes

Not specific to varying attributes, but similar to EssOtlQueryVaryingAttributes, except that the results provide specific associated attributes in pphDetailMemberArray.

Syntax

ESS_FUNC_M EssOtlDetailQueryAttributes (  
    ESS_HOUTLINE_T             hOutline,  
    ESS_PATTRIBUTEQUERY_T    pAttrQuery, 
    ESS_PMBRCOUNTS_T             pCount,  
    ESS_PPHMEMBER_T             pphReturnedMemberArray,  
    ESS_PPHMEMBER_T             pphDetailMemberArray)

Parameter | Data Type | Description
----------|-----------|---------------------

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline context handle (input)</td>
</tr>
<tr>
<td>pAttrQuery</td>
<td>ESS_PATTRIBUTEQUERY_T</td>
<td>Pointer to the structure that defines the query</td>
</tr>
<tr>
<td>pCount</td>
<td>ESS_PMBRCOUNTS_T</td>
<td>Pointer to the number of base members returned</td>
</tr>
<tr>
<td>pphReturnedMemberArray</td>
<td>ESS_PPHMEMBER_T</td>
<td>Pointer to the array of returned member handles</td>
</tr>
<tr>
<td>pphDetailMemberArray</td>
<td>ESS_PPHMEMBER_T</td>
<td>Pointer to the array of member details</td>
</tr>
</tbody>
</table>

Return Value

Returns:

- 0—If successful
- Error number—If unsuccessful

See Also

- EssOtlQueryVaryingAttributes
EssOtlDetailQueryVaryingAttributes

Similar to EssOtlQueryVaryingAttributes, except that the results provide the specific associated attributes in `pphDetailMemberArray`.

**Syntax**

```c
ESS_FUNC_M EssOtlDetailQueryVaryingAttributes(
    ESS_HOUTLINE_T                             hOutline,
    ESS_PVARYING_ATTRIBUTEQUERY_T     pAttrQuery,
    ESS_PPERSPECTIVE_T                  pPerspective,
    ESS_PMBRCOUNTS_T                    pCount,
    ESS_PPHMEMBER_T                     pphMembers,
    ESS_PPHMEMBER_T                     pphDetailMemberArray,
    ESS_USHORT_T                        usValiditySetType,
    ESS_PVALIDITYSET_T                 **pppValiditySets);
```

**Parameter**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline context handle</td>
</tr>
<tr>
<td>pAttrQuery</td>
<td>ESS_PVARYING_ATTRIBUTEQUERY_T</td>
<td>Pointer to the structure that defines the query</td>
</tr>
<tr>
<td>pPerspective</td>
<td>ESS_PPERSPECTIVE_T</td>
<td>Pointer to a collection of independent members used when querying the client or server for associations</td>
</tr>
<tr>
<td>pCount</td>
<td>ESS_PMBRCOUNTS_T</td>
<td>Pointer to the number of base members returned</td>
</tr>
<tr>
<td>pphMembers</td>
<td>ESS_PPHMEMBER_T</td>
<td>Pointer to the array of attribute member handles</td>
</tr>
<tr>
<td>pphDetailMemberArray</td>
<td>ESS_PPHMEMBER_T</td>
<td>Pointer to the array of attribute member detail handles</td>
</tr>
<tr>
<td>usValiditySetType</td>
<td>ESS_USHORT_T</td>
<td>See “ESS_VALIDITYSET_T” on page 710.</td>
</tr>
<tr>
<td><strong>pppValiditySets</strong></td>
<td>ESS_PVALIDITYSET_T</td>
<td>Pointer to the validity set array</td>
</tr>
</tbody>
</table>

**Return Value**

Returns:

- 0—If successful
- Error number—If unsuccessful

**Example**

```c
void TestEssOtlDetailQueryVaryingAttributes()
{
    ESS_STS_T    sts = ESS_STS_NOERR;
    ESS_HOUTLINE_T    hOutline = ESS_NULL;
    ESS_OBJDEF_T    Object;
    ESS_USHORT_T    i = 0;
    ESS_PMBRINFO_T    pMbrInfo = ESS_NULL;
    ESS_VARYING_ATTRIBUTEQUERY_T    pAttrQuery;
    ESS_PMBRCOUNTS_T        Counts;
    ESS_USHORT_T    usValiditySetType;
    ESS_HMEMBER_T    hIndepMbrHandlesArray[4];
```
ESS_PERSPECTIVE_T Perspective;
ESS_PHMEMBER_T phMbrHandles = ESS_NULL;
ESS_PHMEMBER_T phDetailedMembers = ESS_NULL;
ESS_PVALIDITYSET_T *pValiditySets = ESS_NULL;
ESS_HMEMBER_T hAttrMbr, hBaseMbr;
ESS_HMEMBER_T hAttrDim;
ESS_PREDICATE_T Predicate;

memset(&Object, '\0', sizeof(ESS_OBJDEF_T));
memset(&Counts, '\0', sizeof(ESS_MBRCOUNTS_T));
memset(&pAttrQuery, 0x00, sizeof(ESS_ATTRIBUTEQUERY_T));
memset(&Predicate, '\0', sizeof(ESS_PERSPECTIVE_T));

Object.hCtx = hCtx;
Object.ObjType = ESS_OBJTYPE_OUTLINE;
Object.AppName = szAppName;
Object.DbName = szDbName;
Object.FileName = szDbName;
sts = EssOtlOpenOutlineQuery(hCtx, &Object, &hOutline);
printf("EssOtlOpenOutlineQuery sts: %ld\n",sts);

Counts.ulStart = 0;
Counts.ulMaxCount = 10;

Predicate.ulQuery = ESS_SEARCH;
Predicate.ulOptions = ESS_MEMBERSONLY;
Predicate.pszDimension = "";
Predicate.pszString2 = "";

/* Get handles for attribute member and dimension */
Predicate.pszString1 = "Type";
hs = EssOtlQueryMembersByName(hOutline, ESS_NULL, &Predicate, &Counts,
&phMbrHandles);
  hAttrDim = phMbrHandles[0];
  Predicate.pszString1 = "Contractor";
  hs = EssOtlQueryMembersByName(hOutline, ESS_NULL, &Predicate, &Counts,
&phMbrHandles);
  hAttrMbr = phMbrHandles[0];

Predicate.pszString1 = "Doe, Jane";
sts = EssOtlQueryMembersByName(hOutline, ESS_NULL, &Predicate, &Counts,
&phMbrHandles);
  hBaseMbr = phMbrHandles[0];

/* Get handles for independent members */
Predicate.pszString1 = "Jan";
sts = EssOtlQueryMembersByName(hOutline, ESS_NULL, &Predicate, &Counts,
&phMbrHandles);
  hIndepMbrHandlesArray[0] = phMbrHandles[0];
  hIndepMbrHandlesArray[2] = phMbrHandles[0];

Predicate.pszString1 = "FY03";
sts = EssOtlQueryMembersByName(hOutline, ESS_NULL, &Predicate, &Counts,
&phMbrHandles);
  hIndepMbrHandlesArray[1] = phMbrHandles[0];
Predicate.pszString1 = "FY04";
sts = EssOtlQueryMembersByName(hOutline, ESS_NULL, &Predicate, &Counts,
&phMbrHandles);
    hIndepMbrHandlesArray[3] = phMbrHandles[0];

    memset(&Perspective, '\0', sizeof(ESS_PERSPECTIVE_T));
    Perspective.usValiditySetType = ESS_VALIDITYSET_TYPE_MBRHDLS;
    Perspective.countOfIndepDims = 2;
    Perspective.countOfIndepRanges = 1;
    Perspective.pIndepMbrs = hIndepMbrHandlesArray;

    /* Query by handle with InputMemberType of ESS_ATTRIBUTE_MEMBER and OutputMemberType of ESS_BASE_MEMBER*/
    printf("\n*** Query by handle with InputMemberType of ESS_ATTRIBUTE_MEMBER and
    OutputMemberType of ESS_BASE_MEMBER:\n");
    pAttrQuery.bInputMemberIsHandle = ESS_TRUE;
    pAttrQuery.uInputMember.hMember = hAttrMbr;
    pAttrQuery.usInputMemberType = ESS_ATTRIBUTE_MEMBER;
    pAttrQuery.usOutputMemberType = ESS_BASE_MEMBER;
    pAttrQuery.Attribute.usDataType = ESS_ATTRMBRDT_NONE;
    pAttrQuery.usOperation = ESS_ALL;

    usValiditySetType = ESS_VALIDITYSET_TYPE_MBRHDLS;
    sts = EssOtlDetailQueryVaryingAttributes(hOutline, &pAttrQuery, &Perspective,
                &Counts, &phMbrHandles, &phDetailedMembers, usValiditySetType,
                &pValiditySets);
    printf("EssOtlDetailQueryVaryingAttributes sts: %d\n", sts);
    if (!sts)
    {
        if (phMbrHandles)
        {
            printf("tReturned member:\n");
            GetMemberInfo(hOutline, Counts, phMbrHandles);
            if(Counts.ulReturnCount && phMbrHandles)
                sts = EssOtlFreeMembers(hOutline, Counts.ulReturnCount, phMbrHandles);
        }

        if(phDetailedMembers)
        {
            printf("tAssociated attribute member:\n");
            GetMemberInfo(hOutline, Counts, phDetailedMembers);
            if (Counts.ulReturnCount && phDetailedMembers)
                sts = EssOtlFreeMembers(hOutline, Counts.ulReturnCount, phDetailedMembers);
        }
    }

    sts = EssUnlockObject(hCtx, ESS_OBJTYPE_OUTLINE, Object.AppName, Object.DbName,
                Object.FileName);
    printf("\nEssUnlockObject sts: %d\n", sts);
    sts = EssOtlCloseOutline(hOutline);
    printf("EssOtlCloseOutline sts: %d\n",sts);
}

See Also

- EssOtlQueryVaryingAttributes
EssOtlDisassociateAttributeDimension

Disassociates an attribute dimension from a base dimension.

Syntax

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline;</td>
<td>ESS_HOUTLINE_T</td>
<td>Handle to the outline</td>
</tr>
<tr>
<td>hBaseDimension;</td>
<td>ESS_HMEMBER_T</td>
<td>Handle to the base dimension</td>
</tr>
<tr>
<td>hAttributeDimension;</td>
<td>ESS_HMEMBER_T</td>
<td>Handle to the attribute dimension</td>
</tr>
</tbody>
</table>

Notes

When you disassociate an attribute dimension from a base dimension, you disassociate all members of the attribute dimension from members of the base dimension.

Example

```c
void ESS_OtlDisassociateAttributeDimension()
{
    ESS_STS_T        sts = ESS_STS_NOERR;
    ESS_HOUTLINE_T   hOutline;
    ESS_HMEMBER_T    hBaseMbr;
    ESS_HMEMBER_T    hAttrMbr;
    ESS_OBJDEF_T     Object;
    ESS_APPNAME_T    szAppName;
    ESS_DBNAME_T     szDbName;
    ESS_OBJNAME_T    szFileName;
    ESS_PROCSTATE_T  pState;

    memset(&Object, '\0', sizeof(Object));
    Object.hCtx = hCtx;
    Object.ObjType = ESS_OBJTYPE_OUTLINE;
    strcpy(szAppName, "Sample");
    strcpy(szDbName, "Basic");
    strcpy(szFileName, "Basic");
    Object.AppName = szAppName;
    Object.DbName = szDbName;
    Object.FileName = szFileName;
    ESS_STS_T        sts = EssOtlOpenOutline(hCtx, &Object, ESS_TRUE, ESS_TRUE, &hOutline);
    printf("EssOtlOpenOutline() sts: %ld\n",sts);

    sts = EssOtlFindMember(hOutline, "Product", &hBaseMbr);
    printf("EssOtlFindMember() sts: %ld\n",sts);

    sts = EssOtlFindMember(hOutline, "Color", &hAttrMbr);
    printf("EssOtlFindMember() sts: %ld\n",sts);

    sts = EssOtlDisassociateAttributeDimension(hOutline,hBaseMbr,hAttrMbr);
    printf("EssOtlDisassociateAttributeDimension() sts: %ld\n",sts);

    sts = EssOtlWriteOutline(hOutline, &Object);
    printf("EssOtlWriteOutline() sts: %ld\n",sts);
}```
sts = EssOtlRestructure(hCtx, ESS_DOR_ALLDATA);
printf("EssOtlRestructure() sts: %ld\n",sts);

if (!sts)
{
    sts = EssGetProcessState (hCtx, &pState);
    while (!sts || (pState.State != ESS_STATE_DONE))
    {
        sts = EssGetProcessState (hCtx, &pState);
    }
}
sts = EssOtlCloseOutline(hOutline);
printf("EssOtlCloseOutline() sts: %ld\n",sts);

See Also

- EssCheckAttributes
- EssFreeStructure
- EssGetAssociatedAttributesInfo
- EssGetAttributeInfo
- EssGetAttributeSpecifications
- EssOtlAssociateAttributeDimension
- EssOtlAssociateAttributeMember
- EssOtlDisassociateAttributeMember
- EssOtlFindAttributeMembers
- EssOtlFreeStructure
- EssOtlGetAssociatedAttributes
- EssOtlGetAttributeInfo
- EssOtlGetAttributeSpecifications
- EssOtlQueryAttributes
- EssOtlSetAttributeSpecifications

**EssOtlDisassociateAttributeMember**

Disassociates an attribute member from a base member.

**Syntax**

```
ESS_FUNC_M EssOtlDisassociateAttributeMember (hOutline, hBaseMember, hAttributeMember);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Handle to the outline</td>
</tr>
<tr>
<td>hBaseMember</td>
<td>ESS_HMEMBER_T</td>
<td>Handle to the base member</td>
</tr>
<tr>
<td>hAttributeMember</td>
<td>ESS_HMEMBER_T</td>
<td>Handle to the attribute member</td>
</tr>
</tbody>
</table>

**Notes**

When you disassociate an attribute dimension from a base dimension, you disassociate all members of the attribute dimension from members of the base dimension.
Example

```c
void ESS_OtlDisassociateAttributeMember()
{
    ESS_STS_T        sts = ESS_STS_NOERR;
    ESS_HOUTLINE_T   hOutline;
    ESS_HMEMBER_T    hBaseMbr;
    ESS_HMEMBER_T    hAttrMbr;
    ESS_OBJDEF_T     Object;
    ESS_APPNAME_T    szAppName;
    ESS_DBNAME_T     szDbName;
    ESS_OBJNAME_T    szFileName;
    ESS_PROCSTATE_T  pState;

    memset(&Object, '\0', sizeof(Object));
    Object.hCtx = hCtx;
    Object.ObjType = ESS_OBJTYPE_OUTLINE;
    strcpy(szAppName, "Sample");
    strcpy(szDbName, "Basic");
    strcpy(szFileName, "Basic");
    Object.AppName = szAppName;
    Object.DbName = szDbName;
    Object.FileName = szFileName;

    sts = EssOtlOpenOutline(hCtx, &Object, ESS_TRUE, ESS_TRUE, &hOutline);
    printf("EssOtlOpenOutline() sts: %ld\n",sts);

    sts = EssOtlFindMember(hOutline, "Product", &hBaseMbr);
    printf("EssOtlFindMember() sts: %ld\n",sts);

    sts = EssOtlFindMember(hOutline, "Color", &hAttrMbr);
    printf("EssOtlFindMember() sts: %ld\n",sts);

    sts = EssOtlDisassociateAttributeMember(hOutline,hBaseMbr,hAttrMbr);
    printf("EssOtlDisassociateAttributeMember() sts: %ld\n",sts);

    sts = EssOtlWriteOutline(hOutline, &Object);
    printf("EssOtlWriteOutline() sts: %ld\n",sts);

    sts = EssOtlRestructure(hCtx, ESS_DOR_ALLDATA);
    printf("EssOtlRestructure() sts: %ld\n",sts);

    if (!sts)
    {
        sts = EssGetProcessState (hCtx, &pState);
        while (!sts || (pState.State != ESS_STATE_DONE))
            sts = EssGetProcessState (hCtx, &pState);
    }
    sts = EssOtlCloseOutline(hOutline);
    printf("EssOtlCloseOutline() sts: %ld\n",sts);
}
```

See Also

- EssCheckAttributes
- EssFreeStructure
- EssGetAssociatedAttributesInfo
EssOtlEnableDTSMember

 Enables a new DTS member for the outline.

**Syntax**

```
ESS_FUNC_M EssOtlEnableDTSMember (hOutline, pszDTSMember, usGen, bEnable);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline;</td>
<td>ESS_HOUTLINE_T</td>
<td>The Essbase outline handle returned from the EssOtlOpenOutline call.</td>
</tr>
<tr>
<td>pszDTSMember;</td>
<td>ESS_STR_T</td>
<td>Name of the DTS member</td>
</tr>
<tr>
<td>usGen;</td>
<td>ESS_USHORT_T</td>
<td>Generation to assign to the DTS member</td>
</tr>
<tr>
<td>bEnable;</td>
<td>ESS_BOOL_T</td>
<td>Flag to enable the DTS member</td>
</tr>
</tbody>
</table>

**Notes**

This function also fills in the ESS_DTSMBRNAME_T structure passed to it.

**Return Value**

Returns zero if successful.

**Example**

```c
#include "essapi.h"
#include "essotl.h"
#include "esserror.h"

ESS_STS_T ESS_OtlEnableDTSMember(ESS_HCTX_T hCtx, ESS_HINST_T hInst)
{
    ESS_STS_T       sts =ESS_STS_NOERR;
    ESS_HOUTLINE_T  hOutline;
    ESS_OBJDEF_T    Object;
    ESS_APPNAME_T   szAppName;
    ESS_DBNAME_T    szDbName;
    ESS_OBJNAME_T   szFileName;
    ESS_PROCSTATE_T pState;
```
// Define variables
ESS_ULONG_T ulErrors;
ESS_ULONG_T ulCount;
ESS_POUTERROR_T pMbrErrors = NULL;

// Initialize strings
strcpy(szAppName, "1Sample");
strcpy(szDbName, "Basic");
strcpy(szFileName, "Basic");

// Initialize object
memset(&Object, '\0', sizeof(ESS_OBJDEF_T));
Object.hCtx = hCtx;
Object.ObjType = ESS_OBJTYPE_OUTLINE;
Object.AppName = szAppName;
Object.DbName = szDbName;
Object.FileName = szFileName;

// Open outline
sts = EssOtlOpenOutline(hCtx, &Object, ESS_TRUE, ESS_TRUE, &hOutline);
if(sts)
{
    printf("Could not open outline\n");
    return sts;
}

// Enable DTS member alias
sts = EssOtlEnableDTSMember(hOutline, "H-T-D", 1, ESS_TRUE);
if(sts)
{
    printf("Could not enable DTS member alias\n");
}

// Verify outline
sts = EssOtlVerifyOutline(hOutline, &ulErrors, &ulCount, &pMbrErrors);
if(sts)
{
    printf("Could not verify outline\n");
    return sts;
}

// Write outline
sts = EssOtlWriteOutline(hOutline, &Object);
if(sts)
{
    printf("Could not write outline\n");
    return sts;
}

// Restructure outline
sts = EssOtlRestructure(hCtx, ESS_DOR_ALLDATA);
if(sts)
{
    printf("Could not restructure outline\n");
    return sts;
}

// Initialize process state
memset(&pState, 0, sizeof(ESS_PROCSTATE_T));
sts = EssGetProcessState(hCtx, &pState);
{
    printf("sts from Proc State is %d and ProcState is %d\n", sts, pState.State);
    while ((sts == ESS_STS_NOERR) && (pState.State != ESS_STATE_DONE))
    {
        memset(&pState, 0, sizeof(ESS_PROCSTATE_T));
    }
}
sts = EssGetProcessState(hCtx, &pState);
printf("sts from Proc State is %d and ProcState is %d\n", sts,
pState.State);
}
}

EssUnlockObject(hCtx, Object.ObjType, Object.AppName, Object.DbName,
Object.FileName);
EssOtlCloseOutline(hOutline);
return sts;

See Also

- EssOtlDeleteDTSMemberAlias
- EssOtlGetEnabledDTSMembers
- EssOtlGetDTSMemberAlias
- EssOtlSetDTSMemberAlias

**EssOtlFindAlias**

Finds a member with the specified alias name and returns a handle to the member.

**Syntax**

```
ESS_FUNC_M EssOtlFindAlias (hOutline, pszAlias, pszAliasTable, phMember);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline context handle.</td>
</tr>
<tr>
<td>pszAlias</td>
<td>ESS_STR_T</td>
<td>Alias name to find. Can be a simple alias name or a qualified alias name</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(distinguishing this member from another member having the same name). For</td>
</tr>
<tr>
<td></td>
<td></td>
<td>information about syntax used to specify a qualified alias name, see the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Oracle Essbase Database Administrator's Guide section entitled &quot;Creating and</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Working With Duplicate Member Outlines.&quot;</td>
</tr>
<tr>
<td>pszAliasTable</td>
<td>ESS_STR_T</td>
<td>Alias table to search in. Use ESS_NULL to search all alias tables. Use &quot;Default&quot; to</td>
</tr>
<tr>
<td></td>
<td></td>
<td>search the default alias table.</td>
</tr>
<tr>
<td>phMember</td>
<td>ESS_PHMEMBER_T</td>
<td>Variable for the return of the member handle. ESS_NULL if the member is not</td>
</tr>
<tr>
<td></td>
<td></td>
<td>found.</td>
</tr>
</tbody>
</table>

**Notes**

Aliases used in alias combinations are also searched.

**Return Value**

Returns 0 if successful. If no member is found, *phMember is set to ESS_NULL and the call returns 0.

**Example**

```c
#include <essapi.h>
#include <essotl.h>
ESS_STS_T   sts = 0;
```
ESS_OBJDEF_T Object;
ESS_HOUTLINE_T hOutline;
ESS_HMEMBER_T hMemberAlias;
ESS_APPNAME_T szAppName;
ESS_DBNAME_T szDbName;
ESS_OBJNAME_T szFileName;
memset(&Object, '\0', sizeof(Object));
Object.hCtx = hCtx;
Object.ObjType = ESS_OBJTYPE_OUTLINE;
strcpy(szAppName, "Sample");
strcpy(szDbName, "Basic");
strcpy(szFileName, "Basic");
Object.AppName = szAppName;
Object.DbName = szDbName;
Object.FileName = szFileName;
sts = EssOtlOpenOutline(hCtx, &Object, ESS_TRUE, ESS_TRUE, &hOutline);
if (!sts)
{
    /* search all alias tables */
    sts = EssOtlFindAlias(hOutline, "Colas", ESS_NULL, &hMemberAlias);
}

See Also

- EssOtlGetOutlineInfo
- EssOtlGetMemberAlias

**EssOtlFindAttributeMembers**

Returns all attribute members having the specified short name.

**Syntax**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline;</td>
<td>ESS_HOUTLINE_T</td>
<td>Handle to the outline</td>
</tr>
<tr>
<td>pszMember;</td>
<td>ESS_STR_T</td>
<td>Attribute short name</td>
</tr>
<tr>
<td>pszDimName;</td>
<td>ESS_STR_T</td>
<td>Attribute dimension name (optional)</td>
</tr>
<tr>
<td>pusCount;</td>
<td>ESS_PUSHORT_T</td>
<td>Number of base members returned</td>
</tr>
<tr>
<td>pphMembers;</td>
<td>ESS_PPHMEMBER_T</td>
<td>Pointer to an array of base member handles</td>
</tr>
</tbody>
</table>

**Notes**

- *pszMember* must be a short name.
- *pszDimName* is optional. You may enter NULL.
Example

```c
void ESS_OtlFindAttributeMembers()
{
    ESS_STS_T        sts = ESS_STS_NOERR;
    ESS_SHORT_T      index;
    ESS_USHORT_T     count;
    ESS_OBJDEF_T     Object;
    ESS_APPNAME_T    szAppName;
    ESS_DBNAME_T     szDbName;
    ESS_OBJNAME_T    szFileName;
    ESS_HOUTLINE_T   hOutline;
    ESS_PPHMEMBER_T  phMember;
    ESS_PPMBRINFO_T  phMemberInfo;
    ESS_MBRNAME_T    mbrName;
    memset(&Object, '\0', sizeof(Object));
    Object.hCtx = hCtx;
    Object.ObjType = ESS_OBJTYPE_OUTLINE;
    strcpy(szAppName, "Sample");
    strcpy(szDbName, "Basic");
    strcpy(szFileName, "Basic");
    Object.AppName = szAppName;
    Object.DbName = szDbName;
    Object.FileName = szFileName;
    sts = EssOtlOpenOutline(hCtx, &Object, ESS_TRUE, ESS_TRUE, &hOutline);
    printf("EssOtlOpenOutline() sts: %ld\n",sts);
    /* Returning an array of member handles? */
    sts = EssOtlFindAttributeMembers(hOutline,"12", "", &count, &phMember);
    /* sts = EssOtlFindAttributeMembers(hOutline,"10-01-1996", "", &count, &phMember); */
    printf("EssOtlFindAttributeMembers() sts: %ld\n",sts);
    /* Allocate memory for an array of memberinfo struct handles */
    sts = EssAlloc(hInst,count * (sizeof(ESS_HMEMBER_T)), (ESS_PPVOID_T)&phMemberInfo);
    if (!sts)
    {
        for(index = 0; index < count; index++)
        {
            /* Step through array of member handles, and assign member */
            sts = EssOtlGetMemberInfo(hOutline,phMember[index],&phMemberInfo[index]);
            printf("EssOtlGetMemberInfo() sts: %ld\n",sts);
            strcpy(mbrName,phMemberInfo[index]->szMember);
            printf("Attribute member name #%d is: %s\n",(index + 1),mbrName);
        }
        EssFree(hInst, phMember);
        EssFree(hInst, phMemberInfo);
    }
}

See Also
- EssCheckAttributes
- EssFreeStructure
- EssGetAssociatedAttributesInfo
- EssGetAttributeInfo
- EssGetAttributeSpecifications
```
**EssOtlFindMember**

Finds a member with the specified name and returns a handle to the member.

**Syntax**

```c
ESS_FUNC_M EssOtlFindMember (hOutline, pszMember, phMember);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline context handle.</td>
</tr>
<tr>
<td>pszMember</td>
<td>ESS_STR_T</td>
<td>Member name to find. Can be a simple member name or a qualified member name (distinguishing this member from another member having the same name). For information about syntax used to specify a qualified member name, see the Oracle Essbase Database Administrator’s Guide section entitled &quot;Creating and Working With Duplicate Member Outlines.”</td>
</tr>
<tr>
<td>phMember</td>
<td>ESS_PHMEMBER_T</td>
<td>Variable for the return of the member handle. ESS_NULL if the member is not found.</td>
</tr>
</tbody>
</table>

**Notes**

- If the target member has shared members, only the handle to the actual member is returned.
- Once you have the member handle to the actual member, use `EssOtlGetNextSharedMember()` to get shared member information.
- If no member is found, `*phMember` is set to ESS_NULL and the call returns 0.
- Whenever you use `EssOtlFindMember()`, always perform two checks:
  1. Check the return status.
  2. Check whether the handle was returned.

**Return Value**

Returns 0 if successful.

**Example**

```c
#include <essapi.h>
#include <essotl.h>

ESS_STS_T sts = 0;
```
ESS_OBJDEF_T Object;
ESS_HOUTLINE_T hOutline;
ESS_HMEMBER_T hDimProduct;
ESS_APPNAME_T szAppName;
ESS_DBNAME_T szDbName;
ESS_OBJNAME_T szFileName;

memset(&Object, '\0', sizeof(Object));
Object.hCtx = hCtx;
Object.ObjType = ESS_OBJTYPE_OUTLINE;
strcpy(szAppName, "Sample");
strcpy(szDbName, "Basic");
strcpy(szFileName, "Basic");
Object.AppName = szAppName;
Object.DbName = szDbName;
Object.FileName = szFileName;

sts = EssOtlOpenOutline(hCtx, &Object, ESS_TRUE,
                        ESS_TRUE, &hOutline);

if (!sts)
{
    sts = EssOtlFindMember(hOutline, "Product",
                          &hDimProduct);
}

See Also

- EssOtlMoveMember
- EssOtlRenameMember
- EssOtlAddMember
- EssOtlDeleteMember
- EssOtlGetNextSharedMember

**EssOtlFindObject**

Returns the object handle of the specified type and name.

**Syntax**

```c
ESS_FUNC_M EssOtlFindObject(hOutline, objType, objName, pObjHandle)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline handle (Edit mode only)</td>
</tr>
<tr>
<td>objType</td>
<td>ESS_OBJECT_TYPES</td>
<td>Object type with one of the following values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- OBJECT_SMARTLIST</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Object type is Text List (SmartList)</td>
</tr>
<tr>
<td>objName</td>
<td>ESS_STR_T</td>
<td>String identifying the object</td>
</tr>
<tr>
<td>phObjHandle</td>
<td>ESS_PHOBJECT_T</td>
<td>Returns the found object handle.</td>
</tr>
</tbody>
</table>
**Return Value**

Returns:

- **0**—If successful
  
  *phObjHandle* contains the object handle.

- **Error number**—If unsuccessful
  
  *phObjHandle* in NULL.

**Example**

```c
void TestCreateObject()
{
    ESS_STS_T         sts = ESS_STS_NOERR;
    ESS_HOUTLINE_T    hOutline = ESS_NULL;
    ESS_OBJDEF_T      Object;
    ESS_OBJECT_TYPES   objType;
    ESS_STR_T         smartListName;
    ESS_HOBJECT_T     ObjHandle;
    ESS_UULONG_T      Count, i;
    ESS_PHOBJECT_T    ObjHandles;
    ESS_HOBJECT_T     hObjHandle;
    ESS_HSMARTLIST_T  hSmartList;
    ESS_STR_T         objName;

    memset(&Object, '\0', sizeof(Object));
    Object.hCtx = hCtx;
    Object.ObjType = ESS_OBJTYPE_OUTLINE;
    Object.AppName = szAppName;
    Object.DbName = szDbName;
    Object.FileName = szFileName;

    /* Open outline */
    sts = EssOtlOpenOutline(hCtx, &Object,
                            ESS_TRUE, ESS_TRUE, &hOutline);

    /* Create a static SmartList */
    objType = OBJECT_SMARTLIST;
    smartListName = "SList1";
    sts = EssOtlCreateObject(hOutline, objType,
                             smartListName, &ObjHandle);

    /* List all SmartList objects */
    objType = OBJECT_SMARTLIST;
    sts = EssOtlListObjects(hOutline, objType,
                            &Count, &ObjHandles);

    /* Save */
    SaveOutline(hOutline);

    /* Find objects */
    objName = "SList1";
    sts = EssOtlFindObject(hOutline, objType, objName,
                           &hObjHandle);

    /* Delete objects */
    hSmartList = (ESS_HSMARTLIST_T)hObjHandle;
}``
sts = EssOtlDeleteObject(hOutline, hSmartList);
SaveOutline(hOutline);

if(ObjHandles)
    EssFree(hInst, ObjHandles);

/* Unlock objects */
sts = EssUnlockObject(hCtx, Object.ObjType,
        Object.AppName, Object.DbName, Object.FileName);

/* Close outline */
sts = EssOtlCloseOutline(hOutline);
}

See Also

- EssOtlGetMemberSmartList
- EssOtlCreateObject
- EssOtlDeleteObject
- EssOtlGetSmartListInfo
- EssOtlFindObject
- EssOtlFreeObjectArray
- EssOtlFreeSmartListInfo
- EssOtlGetMemberSmartList
- EssOtlGetMemberType
- EssOtlGetObjectReferenceCount
- EssOtlGetObjectReferences
- EssOtlImportExportObject
- EssOtlListObjects
- EssOtlQueryObjects
- EssOtlSetMemberType
- EssOtlSetMemberTypeToSmartList

**EssOtlFreeMembers**

Frees the member array returned from EssOtlQueryMembers().

**Syntax**

ESS_FUNC_M EssOtlFreeMembers (hOutline, ulCount, phMembers);

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Essbase outline handle. This must have been returned from EssOtlOpenOutlineQuery().</td>
</tr>
<tr>
<td>ulCount</td>
<td>ESS_UULONG_T</td>
<td>The number of elements in the phMember array.</td>
</tr>
<tr>
<td>phMembers</td>
<td>ESS_PHMEMBER_T</td>
<td>An array of member handles to be freed.</td>
</tr>
</tbody>
</table>

**Return Value**

The return value is zero if the function was successful.
Example
See the example for EssOt1QueryMembers.

See Also
- EssOt1OpenOutlineQuery
- EssOt1QueryMembers
- EssOt1QueryMembersByName

EssOt1FreeSmartListInfo
Frees the Text List (SmartList) object obtained by EssOt1GetSmartListInfo.

Syntax

```c
ESS_FUNC_M EssOt1FreeSmartListInfo(hOutline, pSmartListInfo);
```

**Parameter** | **Data Type** | **Description**
--- | --- | ---
 hOutline | ESS_HOUTLINE_T | The source Essbase outline for the Text List (SmartList).
 pSmartListInfo | ESS_PSMARTLISTINFO_T | Text List (SmartList) information.

**Return Value**

Returns:
- 0—If successful
- Error number—If unsuccessful

Example

```c
DisplaySmartListInfo(ESS_HOUTLINE_T hOutline, ESS_PHOBJECT_T ObjHandles)
{
    ESS_STS_T                        sts = ESS_STS_NOERR;
    ESS_PSMARTLISTINFO_T    SmartListInfo;
    ESS_ULONG_T                    i;

    sts = EssOt1GetSmartListInfo(hOutline, ObjHandles, &SmartListInfo);
    if(!sts)
    {
        printf("\n");
        printf("\tName: %s\n",    SmartListInfo->szName);
        printf("\tMissing Name: %s\n",    SmartListInfo->szMissingName);
        printf("\tOut of Range Name: %s\n",    SmartListInfo->szOutOfRangeName);
        printf("\tusLen: %d\n",    SmartListInfo->usLen);
        for (i = 0; i < SmartListInfo->usLen; i++)
        {
            printf("\tpIDs: %d, \tpszText[%d]: %s\n",    SmartListInfo->pIDs[i], i,    SmartListInfo->ppszText[i]);
        }
    }
}
```
printf("\n");
}
else
    printf("\t\tEssOtlGetSmartListInfo   sts: %d\n",sts);

if(SmartListInfo)
    sts = EssOtlFreeSmartListInfo(hOutline, SmartListInfo);
}

See Also

- EssOtlGetMemberSmartList
- EssOtlCreateObject
- EssOtlDeleteObject
- EssOtlGetSmartListInfo
- EssOtlFindObject
- EssOtlFreeObjectArray
- EssOtlFreeSmartListInfo
- EssOtlGetMemberSmartList
- EssOtlGetMemberType
- EssOtlGetObjectReferenceCount
- EssOtlGetObjectReferences
- EssOtlImportExportObject
- EssOtlListObjects
- EssOtlQueryObjects
- EssOtlSetMemberType
- EssOtlSetMemberTypeToSmartList

**EssOtlFreeObjectArray**

Deallocates the object handle array.

**Syntax**

`ESS_FUNC_M EssOtlFreeObjectArray(hOutline, count, objHandles)`

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline handle (Query mode only)</td>
</tr>
<tr>
<td>count</td>
<td>ESS_ULONG_T</td>
<td>Count of object handles</td>
</tr>
<tr>
<td>objHandles</td>
<td>ESS_PHOBJECT_T</td>
<td>Array of object handles to be de allocated</td>
</tr>
</tbody>
</table>

**Return Value**

Returns:

- 0—If successful
- Error number—If unsuccessful
Example

void TestFreeObjectArray()
{
    ESS_STS_T                      sts = ESS_STS_NOERR;
    ESS_HOUTLINE_T                hOutline = ESS_NULL;
    ESS_OBJDEF_T                   Object;
    ESS_STR_T                     objNames[1];
    ESS_OBJECT_TYPES              objType;
    ESS_ULONG_T                    count;
    ESS_PHOBJECT_T                hObjHandles = ESS_NULL;

    memset(&Object, '\0', sizeof(Object));
    Object.hCtx = hCtx;
    Object.ObjType = ESS_OBJTYPE_OUTLINE;
    Object.AppName = szAppName;
    Object.DbName = szDbName;
    Object.FileName = szFileName;

    /* Set up */
    sts = EssOtlOpenOutlineQuery(hCtx, &Object, &hOutline);
    count = 2;
    objType = OBJECT_SMARTLIST;
    objNames[0] = "Smartlist1";
    objNames[1] = "Smartlist2";

    /* Query objects */
    sts = EssOtlQueryObjects(hOutline, objType,
                             objNames, &Count, &hObjHandles);

    /* Free object array */
    if(hObjHandles)
    {
        sts = EssOtlFreeObjectArray(hOutline, count,
                                   hObjHandles);
    }

    /* Close outline */
    sts = EssOtlCloseOutline(hOutline);
}

See Also

- EssOtlGetMemberSmartList
- EssOtlCreateObject
- EssOtlDeleteObject
- EssOtlGetSmartListInfo
- EssOtlFindObject
- EssOtlFreeObjectArray
- EssOtlFreeSmartListInfo
- EssOtlGetMemberSmartList
- EssOtlGetMemberType
- EssOtlGetObjectReferenceCount
- EssOtlGetObjectReferences
- EssOtlImportExportObject
EssOtlFreeStructure

Frees memory dynamically allocated by EssOtlGetAttributeInfo() and EssOtlGetMemberInfo() for string type attribute information.

Syntax

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline;</td>
<td>ESS_HOUTLINE_T</td>
<td>Handle to the outline</td>
</tr>
<tr>
<td>structId;</td>
<td>ESS_ULONG_T</td>
<td>One of the following constant identifiers for the structure:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESS_DT_STRUCT_ATTRIBUTEINFO</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESS_DT_STRUCT_ATTRSPECS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESS_DT_STRUCT_MBRINFO</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESS_DT_STRUCT_TIGENINFO</td>
</tr>
<tr>
<td>count;</td>
<td>ESS_ULONG_T</td>
<td>Number of structures</td>
</tr>
<tr>
<td>structPtr;</td>
<td>ESS_PVOID_T</td>
<td>Pointer to memory</td>
</tr>
</tbody>
</table>

Notes

Always call this function, EssOtlFreeStructure(), after you call EssOtlGetMemberInfo().

Example

```c
void ESS_OtlGetAssociatedAttributes()
{
    ESS_STS_T        sts = ESS_STS_NOERR;
    ESS_SHORT_T      index;
    ESS_USHORT_T     count;
    ESS_OBJDEF_T     Object;
    ESS_APPNAME_T    szAppName;
    ESS_DBNAME_T     szDbName;
    ESS_OBJNAME_T    szFileName;
    ESS_HOUTLINE_T   hOutline;
    ESS_PPHMEMBER_T  hMember;
    ESS_PPHMEMBER_T  phMember;
    ESS_PPMBRINFO_T  phMemberInfo;
    ESS_MBRNAME_T    mbrName;

    memset(&Object, '\0', sizeof(Object));
    Object.hCtx = hCtx;
    Object.ObjType = ESS_OBJTYPE_OUTLINE;
    strcpy(szAppName, "Sample");
    strcpy(szDbName, "Basic");
    strcpy(szFileName, "Basic");
```
Object.AppName = szAppName;
Object.DbName = szDbName;
Object.FileName = szFileName;

sts = EssOtlOpenOutline(hCtx, &Object, ESS_TRUE, ESS_TRUE, &hOutline);
printf("EssOtlOpenOutline() sts: %ld\n", sts);

sts = EssOtlFindMember(hOutline, "100-10", &hMember);
printf("EssOtlFindMember() sts: %ld\n", sts);

sts = EssOtlGetAssociatedAttributes(hOutline, hMember, &count, &phMember);
printf("EssOtlGetAssociatedAttributes() sts: %ld\n", sts);

/* Allocate memory for an array of memberinfo structs */
sts = EssAlloc(hInst, count * (sizeof(ESS_MBRINFO_T)), (ESS_PPVOID_T)&phMemberInfo);
if (!sts)
{
    for(index = 0; index < count; index++)
    {
        /* Step through array of member handles, and assign member */
        sts = EssOtlGetMemberInfo(hOutline,phMember[index],&phMemberInfo[index]);
        printf("EssOtlGetMemberInfo() sts: %ld\n", sts);
        strcpy(mbrName,phMemberInfo[index]->szMember);
        printf("Associated attribute member name #%d is: %s\n",(index + 1),mbrName);
    }
    EssFree(hInst, phMember);
    EssOtlFreeStructure(hOutline, ESS_DT_STRUCT_MBRINFO, 1, phMemberInfo);
}

printf("\n Attributes associated :%ld\n\n", count);

See Also

- EssCheckAttributes
- EssFreeStructure
- EssGetAssociatedAttributesInfo
- EssGetAttributeInfo
- EssGetAttributeSpecifications
- EssOtlAssociateAttributeDimension
- EssOtlAssociateAttributeMember
- EssOtlDisassociateAttributeDimension
- EssOtlDisassociateAttributeMember
- EssOtlFindAttributeMembers
- EssOtlGetAssociatedAttributes
- EssOtlGetAttributeInfo
- EssOtlGetAttributeSpecifications
- EssOtlQueryAttributes
- EssOtlSetAttributeSpecifications

**EssOtlGenerateCurrencyOutline**

Generates a currency outline based on the existing outline.
Syntax

```c
ESS_FUNC_M EssOtlGenerateCurrencyOutline (hOutline, phCurOutline);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline context handle.</td>
</tr>
<tr>
<td>phCurOutline</td>
<td>ESS_PHOUTLINE_T</td>
<td>Pointer to an outline context handle for the return of the currency outline.</td>
</tr>
</tbody>
</table>

Notes

- The source outline must have a Time, Accounts, and Country dimension.
- Time dimension and all descendants are copied directly from the source outline to a Time dimension in the new outline.
- A dimension named CurCategory (Dense, Category = Accounts) is created in the new outline. All currency categories in the source Accounts dimension become children of the CurCategory dimension in the new outline.
- A dimension named CurName (Dense, Category = Country) is created in the new outline. All currency names from the source Country dimension become children of the CurName dimension in the new outline.
- A dimension named CurType (Sparse, Category = Type) is created with no children in the new outline.
- The currency outline must be saved by calling `EssOtlWriteOutline()` followed by `EssOtlRestructure()` and closed by calling `EssOtlCloseOutline()`.
- The new outline has the following attributes:
  - Auto-configure is set to ESS_TRUE
  - Case-sensitivity is the same as the original outline

Return Value

Returns 0 if successful; otherwise one of the following:

- OTLAPI_ERR_ALREADYCURRENCY
- OTLAPI_CUR_NOACCOUNTS
- OTLAPI_CUR_NOTIME
- OTLAPI_CUR_NOCOUNTRY

Example

```c
#include <essapi.h>
#include <essotl.h>

ESS_STS_T          sts = 0;
ESS_OBJDEF_T       Object;
ESS_HOUTLINE_T     hOutline;
ESS_HOUTLINE_T     hCurOutline;
ESS_APPNAME_T      szAppName;
ESS_DBNAME_T       szDbName;
ESS_OBJNAME_T      szFileName;
```
memset(&Object, '\0', sizeof(Object));
Object.hCtx = hCtx;
Object.ObjType = ESS_OBJTYPE_OUTLINE;
strcpy(szAppName, "Sample");
strcpy(szDbName, "Interntl");
strcpy(szFileName, "Interntl");
Object.AppName = szAppName;
Object.DbName = szDbName;
Object.FileName = szFileName;

sts = EssOtlOpenOutline(hCtx, &Object, ESS_TRUE,
                        ESS_TRUE, &hOutline);

if (!sts)
{
    sts = EssOtlGenerateCurrencyOutline(hOutline,
                                       &hCurOutline);
}

See Also

- EssOtlOpenOutline
- EssOtlWriteOutline
- EssOtlRestructure

EssOtlGetAggLevelUsage

Returns the applied view selection properties on stored hierarchies.

Syntax

```
ESS_FUNC_M EssOtlGetAggLevelUsage (hOutline, hMember, pAgglevelUsage);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline context handle (input).</td>
</tr>
<tr>
<td>hMember</td>
<td>ESS_HMEMBER_T</td>
<td>Member handle (input).</td>
</tr>
<tr>
<td>pAgglevelUsage</td>
<td>ESS_PSHORT_T</td>
<td>One of the Level Usage Constants listed in EssOtlSetAggLevelUsage documentation (output).</td>
</tr>
</tbody>
</table>

Notes

This function is applicable only to Release 9.3 or higher aggregate storage databases.

Return Value

Returns 0 if successful.

Example

```
ESS_STS_T         sts = ESS_STS_NOERR;
ESS_HOUTLINE_T    hOutline = ESS_NULL;
ESS_HMEMBER_T     hMember = ESS_NULL;
ESS_SHORT_T       sAggLevelUsage = 0;
```
if (hOutline && hMember)
{
    sts = EssOtlGetAggLevelUsage(hOutline, hMember, &sAggLevelUsage);
    if (sts)
        printf("Error (%ld) getting AggLevelUsage\n", sts);
    else
        printf("AggLevelUsage is: %d ", sAggLevelUsage);
        switch (sAggLevelUsage)
        {
        case ESS_AGGLEVELUSAGE_NOTSET :
            printf("(not set)\n");
        break;
        case ESS_AGGLEVELUSAGE_DEFAULT :
            printf("(Default)\n");
        break;
        case ESS_AGGLEVELUSAGE_ALL :
            printf("(All levels considered)\n");
        break;
        case ESS_AGGLEVELUSAGE_NOAGGREGATION :
            printf("(Do not aggregate)\n");
        break;
        case ESS_AGGLEVELUSAGE_BOTTOMONLY :
            printf("(Bottom level only considered)\n");
        break;
        case ESS_AGGLEVELUSAGE_TOPONLY :
            printf("(Top level only considered)\n");
        break;
        case ESS_AGGLEVELUSAGE_BOTTOMTOP :
            printf("(Never aggregate intermediate levels)\n");
        break;
        case ESS_MULTIPLE_HIERARCHY_IS_ENABLED :
            printf("(Error: Multiple hierarchies - hierarchy members are gen=2)\n");
        break;
        case ESS_MULTIPLE_HIERARCHIES_NOT_ENABLED :
            printf("(Error: Single hierarchy - hierarchy member is gen=1)\n");
        break;
        case ESS_NOT_HIERARCHY_MEMBER :
            printf("(Error: This member does not carry agglevel information)\n");
        break;
        default: printf("(Unrecognized response)\n");
        }
    }
else
{
    if (!hOutline)
        printf("Outline not provided\n");
    if (!hMember)
        printf("Member not provided\n");
}

See Also

- EssOtlSetAggLevelUsage
EssOtlGetAliasTableLanguages

Returns an array of language codes, and the number of language codes in the array, that are associated with the specified alias table.

Syntax

ESS_FUNC_M EssOtlGetAliasTableLanguages (hOutline, pszAliasTable, pulCount, ppLangArray);

Parameter | Data Type    | Description
----------|--------------|-------------
 hOutline   | ESS_HOUTLINE_T | Handle to the outline.
 pszAliasTable | ESS_STR_T     | Name of the alias table for which to get the associated language codes.
 pulCount   | ESS_PULONG_T  | Address of a variable in which to return the number of language codes associated with the alias table.
 ppLangArray | ESS_PPALIASLANG_T | An array of the language codes associated with the alias table specified in pszAliasTable.

The memory allocated for ppLangArray should be freed using EssFree().

Return Value

- If successful, returns 0.
- If unsuccessful, returns the error OTLAPI_BAD_ALIASTABLE (invalid alias table).

Access

This function does not require special privileges.

Example

```
#include <essapi.h>
#include <essotl.h>

ESS_STS_T          sts = 0;
ESS_OUTLINEINFO_T  NewInfo;
ESS_HOUTLINE_T     hOutline;
ESS_PPALIASLANG_T  pLangs=ESS_NULL;
ESS_ULONG_T        nLangs = 0, i=0;

memset(&NewInfo, '\0', sizeof(NewInfo));
sts = EssOtlNewOutline(hCtx, &NewInfo, &hOutline);

if (!sts)
{
    sts = EssOtlCreateAliasTable(hOutline, "French Alias Table");
}

if (!sts)
{
    sts = EssOtlSetAliasTableLanguage (hOutline, "French Alias Table", "fr");
}
```
if (!sts)
{
    sts = EssOtlSetAliasTableLanguage (hOutline,
            "French Alias Table", "fr-CA");
}

if (!sts)
{
    sts = EssOtlGetAliasTableLanguages(hOutline, "French Alias Table", &nLangs, &pLangs);

    if ( !sts == ESS_STS_NOERR && ( pLangs) )
    {
        for (i=0;i<nLangs ;++i)
        {
            if (pLangs[i])
                { printf("Language Code: %s\n", pLangs[i]);
            }
            EssFree(hInst, pLangs);
        }
    }
}
if (!sts)
{
    sts = EssOtlClearAliasTableLanguages (hOutline,
            "French Alias Table");
}

See Also

- EssOtlClearAliasTableLanguages
- EssOtlSetAliasTableLanguage

**EssOtlGetAltHierarchyEnabled**

Returns the dimension's multiple-hierarchy-enabled setting.

**Syntax**

```c
ESS_FUNC_M EssOtlGetAltHierarchyEnabled(hOutline, hDimMember, pEnabled);
```

**Parameter** | **Data Type** | **Description**
---|---|---
`hOutline` | ESS_HOUTLINE_T | Outline context handle (input).
`hDimMember` | ESS_HMEMBER_T | A dimension member (input).
`pEnabled` | ESS_BOOL_T | Returns TRUE if the dimension is multiple hierarchy enabled, and FALSE otherwise.

**Return Value**

- 0—If successful
- Returns error OTLAPI_ERR_BADDIM if `hDimMember` is not a dimension member.
EssOtlGetASOCompressionDimension

Returns the handle of the aggregate storage dimension tagged as Compression.

Syntax

ESS_FUNC_M EssOtlGetASOCompressionDimension (hOutline, phDim);

Parameter | Data Type | Description
---|---|---
| hOutline | ESS_HOUTLINE_T | Outline context handle (input).
| phDim | ESS_PHMEMBER_T | Pointer to a dimension handle (output).

Notes

By default, the compression dimension in aggregate storage databases is the Accounts dimension. To change the compression dimension, use EssOtlSetASOCompressionDimension. Changing the compression dimension triggers a full restructure of the database.

Return Value

Returns 0 if successful.

Example

```c
ESS_STS_T         sts = ESS_STS_NOERR;
ESS_HOUTLINE_T    hOutline = ESS_NULL;
ESS_PMBRINFO_T    pMemberInfo = ESS_NULL;
ESS_HMEMBER_T     hMember = ESS_NULL;

/* code to assign hOutline variable omitted */

if (hOutline)
{
    sts = EssOtlGetASOCompressionDimension(hOutline, &hMember);
    if (!sts)
    {
        if (hMember)
        {
            sts = EssOtlGetMemberInfo(hOutline, hMember, &pMemberInfo);
            printf("The ASO compression dimension is: %s\n", pMemberInfo->szMember);
        }
        else
        {
            printf("Outline has no dimension selected for compression\n");
        }
    }
    else
    {
```

printf("Error returned\n");
}
else
{
    printf("NULL outline selected");
}

See Also

● EssOt1SetASOCompressionDimension

EssOt1GetAssociatedAttributes

Returns all attribute members that are associated with a base member or dimension.

Syntax

```c
ESS_FUNC_M EssOt1GetAssociatedAttributes(hOutline, hMember, pusCount, pphMemberArray);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline;</td>
<td>ESS_HOUTLINE_T</td>
<td>Handle to the outline</td>
</tr>
<tr>
<td>hMember;</td>
<td>ESS_HMEMBER_T</td>
<td>Handle to the base member or base dimension</td>
</tr>
<tr>
<td>pusCount;</td>
<td>ESS_PUSHORT_T</td>
<td>Number of attribute members returned</td>
</tr>
<tr>
<td>pphMemberArray;</td>
<td>ESS_PPHMEMBER_T</td>
<td>Pointer to an array of attribute member handles</td>
</tr>
</tbody>
</table>

Example

```c
void ESS_OtlGetAssociatedAttributes()
{
    ESS_STS_T        sts = ESS_STS_NOERR;
    ESS_SHORT_T      index;
    ESS_USHORT_T     count;
    ESS_OBJDEF_T     Object;
    ESS_APPNAME_T    szAppName;
    ESS_DBNAME_T     szDbName;
    ESS_OBJNAME_T    szFileName;
    ESS_HOUTLINE_T   hOutline;
    ESS_PPHMEMBER_T  hMember;
    ESS_PPHMEMBER_T  phMember;
    ESS_PPMBRINFO_T  phMemberInfo;
    ESS_MBRNAME_T    mbrName;

    memset(&Object, '\0', sizeof(Object));
    Object.hCtx = hCtx;
    Object.ObjType = ESS_OBJTYPE_OUTLINE;
    strcpy(szAppName, "Sample");
    strcpy(szDbName, "Basic");
    strcpy(szFileName, "Basic");
    Object.AppName = szAppName;
    Object.DbName = szDbName;
```
Object.FileName = szFileName;

sts = EssOtlOpenOutline(hCtx, &Object, ESS_TRUE, ESS_TRUE, &hOutline);
printf("EssOtlOpenOutline() sts: %ld\n",sts);

sts = EssOtlFindMember(hOutline, "100-10", &hMember);
printf("EssOtlFindMember() sts: %ld\n",sts);

sts = EssOtlGetAssociatedAttributes(hOutline, hMember, &count, &phMember);
printf("EssOtlGetAssociatedAttributes() sts: %ld\n",sts);

/* Allocate memory for an array of memberinfo struct handles */
sts = EssAlloc(hInst,count * (sizeof(ESS_HMEMBER_T)), (ESS_PPVOID_T)&phMemberInfo);
if (!sts)
{
  for(index = 0; index < count; index++)
  {
    /* Step through array of member handles, and assign member */
    sts = EssOtlGetMemberInfo(hOutline,phMember[index],&phMemberInfo[index]);
    printf("EssOtlGetMemberInfo() sts: %ld\n",sts);
    strcpy(mbrName,phMemberInfo[index]->szMember);
    printf("Associated attribute member name #%d is: %s\n",(index + 1),mbrName);
  }
  EssFree(hInst, phMember);
  EssFree(hInst, phMemberInfo);
}

printf("\n Attributes associated :%ld\n\n", count);

See Also

- EssCheckAttributes
- EssFreeStructure
- EssGetAssociatedAttributesInfo
- EssGetAttributeInfo
- EssGetAttributeSpecifications
- EssOtlAssociateAttributeDimension
- EssOtlAssociateAttributeMember
- EssOtlDisassociateAttributeDimension
- EssOtlDisassociateAttributeMember
- EssOtlFindAttributeMembers
- EssOtlFreeStructure
- EssOtlGetAttributeInfo
- EssOtlGetAttributeSpecifications
- EssOtlQueryAttributes
- EssOtlSetAttributeSpecifications

**EssOtlGetAttributeAssocLevel**

Gets the association level for an attribute or linked attribute dimension.
Every attribute has an association level and an attachment level associated with the attribute dimension definition. For a linked attribute dimension, the association level is always the shorter of the two periods in the periodic comparison represented by the linked attribute dimension. For example, in the linked attribute dimension Quarter by Year, Quarter is the association level, and Year is the attachment level.

Syntax

```
ESS_FUNC_M EssOtlGetAttributeAssocLevel (hOutline, hDimMember, psLevel);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline context handle (input).</td>
</tr>
<tr>
<td>hDimMember</td>
<td>ESS_HMEMBER_T</td>
<td>Attribute or linked-attribute dimension member handle (input).</td>
</tr>
<tr>
<td>psLevel</td>
<td>ESS_PUSHORT_T</td>
<td>The attribute association level (output).</td>
</tr>
</tbody>
</table>

Notes

- Before you call this function, open the outline in edit or query mode using either `EssOtlOpenOutline` or `EssOtlOpenOutlineQuery`.
- This function is applicable when `hDimMember` is any type of attribute dimension.

Return Value

Returns 0 if successful; otherwise, returns an error.

Example

```c
#include <essapi.h>
#include <essotl.h>

ESS_STS_T          sts = 0;
ESS_OBJDEF_T       Object;
ESS_HOUTLINE_T     hOutline;
ESS_HMEMBER        hDimMember;
ESS_USHORT_T       usAssocLevel;
ESS_APPNAME_T      szAppName;
ESS_DBNAME_T       szDbName;
ESS_OBJNAME_T      szFileName;
memset(&Object, '\0', sizeof(Object));
Object.hCtx = hCtx;
Object.ObjType = ESS_OBJTYPE_OUTLINE;
strcpy(szAppName, "Sample");
strcpy(szDbName, "Basic");
strcpy(szFileName, "Basic");
Object.AppName = szAppName;
Object.DbName = szDbName;
Object.FileName = szFileName;

sts = EssOtlOpenOutline(hCtx, &Object, ESS_TRUE,
                         ESS_TRUE, &hOutline);
if (!sts)
{
```

C Outline API Functions
sts = EssOtlFindMember(hOutline, "Quarter By Year", &hDimMember);
}

if (!sts && hMemberJan)
{
    sts = EssOtlGetAttributeAssocLevel(hOutline, hDimMember, &usAssocLevel);
}

See Also

- EssOtlGetLinkedAttributeAttachLevel
- EssOtlQueryGenerationInfo

EssOtlGetAttributeInfo

Returns attribute information for a given attribute member or dimension.

Syntax

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Handle to the outline</td>
</tr>
<tr>
<td>hAttribute</td>
<td>ESS_HMEMBER_T</td>
<td>Handle to the attribute member or dimension</td>
</tr>
<tr>
<td>pAttributeInfo</td>
<td>“ESS_ATTRIBUTEINFO_T”</td>
<td>Attribute information</td>
</tr>
</tbody>
</table>

Notes

- This function is similar to EssGetAttributeInfo().
- After you call this function, call EssOtlFreeStructure() to free memory dynamically allocated by EssOtlGetAttributeInfo() for string type attribute information.

Example

```c
void ESS_GetAttributeInfo()
{
    ESS_PPATTRIBUTEINFO_T    pAttributeInfo = ESS_NULL;
    ESS_STS_T        sts = ESS_STS_NOERR;
    ESS_OBJDEF_T      Object;
    ESS_APPNAME_T      szAppName;
    ESS_DBNAME_T      szDbName;
    ESS_OBJNAME_T      szFileName;
    ESS_HOUTLINE_T       hOutline;
    ESS_PPHMEMBER_T      phMember;
    ESS_PPMBRINFO_T      phMemberInfo;
    ESS_MBRNAME_T      mbrName;
    ESS_HMEMBER_T      hMember;

    memset(&Object, '\0', sizeof(Object));
    Object.hCtx = hCtx;
    Object.ObjType = ESS_OBJTYPE_OUTLINE;
```
strcpy(szAppName, "Sample");
strcpy(szDbName, "Basic");
strcpy(szFileName, "Basic");
Object.AppName = szAppName;
Object.DbName = szDbName;
Object.FileName = szFileName;
sts = EssOtlOpenOutline(hCtx, &Object, ESS_TRUE, ESS_TRUE, &hOutline);
printf("EssOtlOpenOutline() sts: %ld\n", sts);

sts = EssOtlFindMember(hOutline, "100-10", &hmember);
printf("EssOtlFindMember() sts: %ld\n", sts);

sts = EssOtlGetAttributeInfo(hOutline, hMember, &pAttributeInfo);
if (sts == ESS_STS_NOERR && pAttributeInfo)
{
    printf("\n------Attribute Information------\n");
    printf("Member name:                %s\n", pAttributeInfo->MbrName);
    printf("Dim name:                   %s\n", pAttributeInfo->DimName);

    switch(pAttributeInfo->Attribute.usDataType)
    {
    case (ESS_ATTRMBRDT_STRING):
        printf("Attribute data type:        Text\n");
        if(pAttributeInfo->Attribute.value.strData)
            printf("Attribute value:            %s\n",pAttributeInfo->Attribute.value.strData);
        break;

    case (ESS_ATTRMBRDT_BOOL):
        printf("Attribute data type:        Boolean\n");
        printf("Attribute value:            %d\n",pAttributeInfo->Attribute.value.bData);
        break;

    case (ESS_ATTRMBRDT_DOUBLE):
        printf("Attribute data type:        Numeric\n");
        printf("Attribute value:            %f\n",pAttributeInfo->Attribute.value.dblData);
        break;

    case (ESS_ATTRMBRDT_DATETIME):
        printf("Attribute data type:        Date\n");
        printf("Attribute value:            %s\n",ctime(&pAttributeInfo->Attribute.value.dtData));
        break;

    case (ESS_ATTRMBRDT_NONE):
        printf("Attribute data type:        None\n");
        break;

    default:
        printf("Attribute data type:        \n");
        break;
    }
}
}
EssOtlGetAttributeSpecifications

Retrieves attribute specifications for the outline.

Syntax

Parameter | Data Type | Description
---|---|---
hOutline; | ESS_HOUTLINE_T | Handle to the outline
pAttrSpecs; | “ESS_ATTRSPECS_T” on page 119 | Attribute specifications

Notes

- This function is similar to EssGetAttributeSpecifications(), except that it returns information from the opened outline.
- Set attribute specifications for the outline using EssOtlSetAttributeSpecifications().
- Attribute specifications are used to do the following:
  - Generate a long name
  - Indicate the format of a datetime attribute
  - Indicate a numeric attribute’s bucketing type
  - Provide the name of the attribute calculations dimension and the names for the values used with it

Example

```c
void  ESS_OtlGetAttributeSpecifications()
{
    ESS_STS_T         sts = ESS_STS_NOERR;
```
ESS_PATTRSPECS_T AttrSpecs;
ESS_OBJDEF_T Object;
ESS_HOUTLINE_T hOutline;
ESS_APPNAME_T szAppName;
ESS_DBNAME_T szDbName;
ESS_OBJNAME_T szFileName;
ESS_PROCSTATE_T pState;

memset(&Object, '\0', sizeof(Object));
Object.hCtx = hCtx;
Object.ObjType = ESS_OBJTYPE_OUTLINE;
strcpy(szAppName, "Sample");
strcpy(szDbName, "Basic");
strcpy(szFileName, "Basic");
Object.AppName = szAppName;
Object.DbName = szDbName;
Object.FileName = szFileName;

sts = EssOtlOpenOutline(hCtx, &Object, ESS_TRUE, ESS_TRUE, &hOutline);
printf("EssOtlOpenOutline() sts: %ld\n", sts);

sts = EssOtlGetAttributeSpecifications(hOutline, &AttrSpecs);
printf("EssOtlGetAttributeSpecifications() sts: %ld\n", sts);

switch(AttrSpecs->usGenNameBy)
{
    case ESS_GENNAMEBY_PREFIX:
        printf("\n Prefix/Suffix   : Prefix\n");
        break;
    case ESS_GENNAMEBY_SUFFIX:
        printf("\n Prefix/Suffix   : Suffix\n");
        break;
    default:
        printf("\n Prefix/Suffix   : None\n");
        break;
}

switch(AttrSpecs->usUseNameOf)
{
    case ESS_USENAMEOF_PARENT:
        printf("\n Use Name of     : Parent\n");
        break;
    case ESS_USENAMEOF_GRANDPARENTANDPARENT:
        printf("\n Use Name of     : Grand Parent and Parent\n");
        break;
    case ESS_USENAMEOF_ALLANCESTORS:
        printf("\n Use Name of     : All Ancestors\n");
        break;
    case ESS_USENAMEOF_DIMENSION:
        printf("\n Use Name of     : Dimension\n");
        break;
    case ESS_USENAMEOF_NONE:
        printf("\n Use Name of     : None\n");
        break;
    default:
        printf("\n Use Name of     : Invalid setting\n");
        break;
}
switch(AttrSpecs->cDelimiter)
{
    case ESS_DELIMITER_PIPE:
        printf("\n Delimiter       : '|'");
        break;
    case ESS_DELIMITER_UNDERSCORE:
        printf("\n Delimiter       : '_'");
        break;
    case ESS_DELIMITER_CARET:
        printf("\n Delimiter       : '^'");
        break;
    default:
        printf("\n Delimiter       : Invalid setting");
        break;
}

switch(AttrSpecs->usDateFormat)
{
    case ESS_DATEFORMAT_DDMMYYYY :
        printf("\n Date Format     : DD-MM-YYYY");
        break;
    case ESS_DATEFORMAT_MMDDYYYY :
        printf("\n Date Format     : MM-DD-YYYY");
        break;
    default:
        printf("\n Date Format     : Invalid setting");
        break;
}

switch(AttrSpecs->usBucketingType)
{
    case ESS_UPPERBOUNDINCLUSIVE :
        printf("\n Bucketing Type  : Upper Bound inclusive");
        break;
    case ESS_UPPERBOUNDNONINCLUSIVE :
        printf("\n Bucketing Type  : Upper Bound non-inclusive");
        break;
    case ESS_LOWERBOUNDINCLUSIVE :
        printf("\n Bucketing Type  : Lower Bound inclusive");
        break;
    case ESS_LOWERBOUNDNONINCLUSIVE :
        printf("\n Bucketing Type  : Lower Bound non-inclusive");
        break;
    default:
        printf("\n Bucketing Type   : Invalid setting");
        break;
}

printf("\n Default for TRUE       : %s",
        AttrSpecs->pszDefaultTrueString);

printf("\n Default for FALSE      : %s",
        AttrSpecs->pszDefaultFalseString);

printf("\n Default for Attr Calc  : %s",
        AttrSpecs->pszDefaultAttrCalcDimName);
printf("\n Default for Sum        : %s", 
    AttrSpecs->pszDefaultSumMbrName);

printf("\n Default for Count      : %s", 
    AttrSpecs->pszDefaultCountMbrName);

printf("\n Default for Average    : %s", 
    AttrSpecs->pszDefaultAverageMbrName);

printf("\n Default for Min        : %s", 
    AttrSpecs->pszDefaultMinMbrName);

printf("\n Default for Max        : %s", 
    AttrSpecs->pszDefaultMaxMbrName);

printf("\n");

sts = EssOtlWriteOutline(hOutline, &Object);
printf("EssOtlWriteOutline() sts: %ld\n",sts);

sts = EssOtlRestructure(hCtx, ESS_DOR_ALLDATA);
printf("EssOtlRestructure() sts: %ld\n",sts);

if (!sts)
{ 
    sts = EssGetProcessState (hCtx, &pState);
    while (!sts || (pState.State != ESS_STATE_DONE))
    {
        sts = EssGetProcessState (hCtx, &pState);
    }
sts = EssOtlCloseOutline(hOutline);
printf("EssOtlCloseOutline() sts: %ld\n",sts);

    EssOtlFreeStructure(hInst, ESS_DT_STRUCT_ATTRSPECS, 1,&AttrSpecs);
}

See Also

- EssCheckAttributes
- EssFreeStructure
- EssGetAssociatedAttributesInfo
- EssGetAttributeInfo
- EssGetAttributeSpecifications
- EssOtlAssociateAttributeDimension
- EssOtlAssociateAttributeMember
- EssOtlDisassociateAttributeDimension
- EssOtlDisassociateAttributeMember
- EssOtlFindAttributeMembers
- EssOtlFreeStructure
- EssOtlGetAssociatedAttributes
- EssOtlGetAttributeInfo
- EssOtlQueryAttributes
- EssOtlSetAttributeSpecifications
EssOtlGetChild

Returns the child of a member.

Syntax

ESS_FUNC_M EssOtlGetChild (hOutline, hMember, phMember);

Parameter | Data Type | Description
--- | --- | ---
hOutline | ESS_HOUTLINE_T | Outline context handle.
hMember | ESS_HMEMBER_T | Handle of member to retrieve the child of.
phMember | ESS_PHMEMBER_T | Pointer for return of a member handle of the child of the hMember parameter.

Notes

- If there is no child, *phMember is set to ESS_NULL and the call returns 0.

Return Value

Returns 0 if successful.

Example

```c
#include <essapi.h>
#include <essotl.h>

ESS_STS_T          sts = 0;
ESS_HOUTLINE_T     hOutline;
ESS_HMEMBER_T      hMemberParent;
ESS_HMEMBER_T      hMemberChild;
ESS_OBJDEF_T       Object;
ESS_APPNAME_T      szAppName;
ESS_DBNAME_T       szDbName;
ESS_OBJNAME_T      szFileName;
memset(&Object, '\0', sizeof(Object));
Object.hCtx = hCtx;
Object.ObjType = ESS_OBJTYPE_OUTLINE;
strcpy(szAppName, "Sample");
strcpy(szDbName, "Basic");
strcpy(szFileName, "Basic");
Object.AppName = szAppName;
Object.DbName = szDbName;
Object.FileName = szFileName;
sts = EssOtlOpenOutline(hCtx, &Object, ESS_TRUE,
                           ESS_TRUE, &hOutline);
if (!sts)
{
    sts = EssOtlFindMember(hOutline, "Year",
                           &hMemberParent);
}
if (!sts && hMemberParent)
{
    sts = EssOtlGetChild(hOutline, hMemberParent,
```
See Also

- EssOtlGetParent
- EssOtlGetNextSibling
- EssOtlGetPrevSibling
- EssOtlGetFirstMember

EssOtlGetCountOfDupMemberNameInDim

Returns the number of members in a dimension whose names are duplicate in the outline opened in query mode.

Syntax

```c
ESS_FUNC_M EssOtlGetDimensionNameUniqueness (hOutline, hDim, *pulDupCount);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline context handle (input).</td>
</tr>
<tr>
<td>hDim</td>
<td>ESS_HMEMBER_T</td>
<td>Input dimension, returned by EssOtlQueryGetFirstDimension() or EssOtlQueryGetNextDimension().</td>
</tr>
<tr>
<td>*pulDupCount</td>
<td>ESS_ULONG_T</td>
<td>The number of members with duplicate names (output).</td>
</tr>
</tbody>
</table>

Notes

- A shared member in the dimension will not influence the count.
- Before you call this function, call EssOtlOpenOutlineQuery to open the outline in query mode.

Return Value

Returns 0 if successful; otherwise, returns an error.

Example

```c
ESS_FUNC_M ESS_GetCount()
{
    ESS_STS_T    sts = 0;
    ESS_HOUTLINE_T   hOutline;
    ESS_OBJDEF_T   Object;
    ESS_APPNAME_T   szAppName;
    ESS_DBNAME_T   szDbName;
    ESS_OBJNAME_T   szFileName;
    ESS_HMEMBER_T   hDim;
    ESS_LONG_T    pulDupCount;

    memset(&Object, '\0', sizeof(Object));
    Object.hCtx =   hCtx;
    Object.ObjType =  ESS_OBJTYPE_OUTLINE;
    strcpy(szAppName, "Demo");
```
strcpy(szDbName, "Test");
strcpy(szFileName, "Test");
Object.AppName = szAppName;
Object.DbName = szDbName;
Object.FileName = szFileName;

sts = EssOtlOpenOutlineQuery (hCtx, &Object, &hOutline);

if (!sts)
{
    sts = EssOtlQueryGetFirstDimension(hOutline, &hDim);

    if (sts)
        printf("EssOtlQueryGetFirstDimension failed sts %ld\n",sts);
}

if (!sts)
{
// returns pulDupCount which gives the number of members in a dimension
// whose names are duplicate
    sts = EssOtlGetCountOfDupMemberNameInDim (hOutline, hDim, &pulDupCount);

    if (sts)
        printf("EssOtlGetCountOfDupMemberNameInDim failed sts %ld\n",sts);
}

return sts;

See Also

* EssOtlQueryGetFirstDimension
  * EssOtlQueryGetNextDimension

**EssOtlGetDateFormatString**

This function gets the outline property date format String.

**Syntax**

```c
ESS_FUNC_M EssOtlGetDateFormatString(
    ESS_HOUTLINE_T hOutline,
    ESS_PSTR_T   formatString)
```

**Parameter** | **Data Type**   | **Description**
---|---|---
  hOutline      | ESS_HOUTLINE_T | Outline handle
  formatString  | ESS_PSTR_T     | Returns the outline date format string to this argument.

**Return Value**

Returns:

* 0—If successful
**formatString** contains the outline date format.

- *Error number—If unsuccessful*

**Example**

```c
void TestGetSetDateFormatString()
{
    ESS_STS_T st = ESS_STS_NOERR;
    ESS_HOUTLINE_T hOutline = ESS_NULL;
    ESS_OBJDEF_T Object;
    ESS_SHORT_T length = 80;
    ESS_STR_T dateFormatString = "\n";
    ESS_STR_T localeStr;
    ESS_USHORT_T count, i;
    ESS_STR_T* pdateStrings;
    ESS_STR_T* pformatStrings;

    memset(&Object, '\0', sizeof(Object));
    Object.hCtx = hCtx;
    Object.ObjType = ESS_OBJTYPE_OUTLINE;
    Object.AppName = szAppName;
    Object.DbName = szDbName;
    Object.FileName = szFileName;

    st = EssOtlOpenOutline(hCtx, &Object,
                           ESS_TRUE, ESS_TRUE, &hOutline);
    /* Get current value */
    st = EssOtlGetDateFormatString(hOutline, &dateFormatString);
    printf("EssOtlGetSMDateFormatString sts: %d \n", st);
    printf("\tDate format string: %s\n", dateFormatString);

    printf("\n");
    localeStr = "English_UnitedStates.Latin1@Binary";
    st = EssOtlGetServerDateFormats(hCtx, localeStr,
                                     &Count, &pdateStrings, &pformatStrings);
    printf("EssOtlGetServerDateFormats sts: %d \n", st);

    for (i = 0; i < count; i++)
    {
        printf("\nCase with %s: \n", pformatStrings[i]);
        st = EssOtlSetDateFormatString(hOutline,
                                        pformatStrings[i]);
        printf("EssOtlSetSMDateFormatString sts: %d \n", st);
        SaveOutline(hOutline);

        st = EssOtlGetDateFormatString(hOutline,
                                        &dateFormatString);
        printf("EssOtlGetSMDateFormatString sts: %d \n", st);
        printf("\tDate format string: %s\n", dateFormatString);
    }

    st = EssUnlockObject(hCtx, Object.ObjType,
                         Object.AppName, Object.DbName, Object.FileName);
    st = EssOtlCloseOutline(hOutline);
}
```

798  C Outline API Functions
printf("EssOtlCloseOutline   sts: %d\n",sts);
}

See Also

- EssOtlGetServerDateFormats
- EssOtlSetDateFormatString

EssOtlGetDimensionNameUniqueness

Returns the dimension's member-name uniqueness setting.

Syntax

ESS_FUNC_M EssOtlGetDimensionNameUniqueness (hOutline, hDim, pbNameUnique);

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline context handle (input).</td>
</tr>
<tr>
<td>hDim</td>
<td>ESS_HMEMBER_T</td>
<td>Member handle of the dimension root member (input).</td>
</tr>
<tr>
<td>pbNameUnique</td>
<td>ESS_PBOOL_T</td>
<td>The dimension member name uniqueness setting (output). If TRUE, the dimension cannot have duplicate member names.</td>
</tr>
</tbody>
</table>

Notes

Call EssOtlFindMember to set up the ESS_HMEMBER_T (hDim) variable.

Return Value

Returns 0 if successful; otherwise, returns an error.

Example

ESS_FUNC_M ESS_GetSetDimNameUniq()
{
    ESS_STS_T    sts = 0;
    ESS_POUTLINEINFO_T pInfo = ESS_NULL;
    ESS_HOUTLINE_T   hOutline;
    ESS_OBJDEF_T   Object;
    ESS_APPNAME_T   szAppName;
    ESS_DBNAME_T    szDbName;
    ESS_OBJNAME_T   szFileName;
    ESS_BOOL_T     pbNameUnique;
    ESS_HMEMBER_T   hDim = ESS_NULL;

    memset(&Object, '\0', sizeof(Object));
    Object.hCtx =   hCtx;
    Object.ObjType =  ESS_OBJTYPE_OUTLINE;
    strcpy(szAppName, "Demo");
    strcpy(szDbName, "Test");
    strcpy(szFileName, "Test");
    Object.AppName =  szAppName;
    Object.DbName =   szDbName;
    Object.FileName =  szFileName;
sts = EssOtlOpenOutline(hCtx, &Object, ESS_TRUE, ESS_TRUE, &hOutline);

if (!sts)
{
    sts = EssOtlFindMember(hOutline, "Year", &hDim);

    if (sts)
        printf("EssOtlFindMember failed sts %ld\n", sts);
}

/*Get the dimension's, Year, member-name uniqueness setting */
if (!sts)
{
    sts = EssOtlGetDimensionNameUniqueness (hOutline, hDim, &pbNameUnique);

    if (sts)
        printf("EssOtlGetDimensionNameUniqueness failed sts %ld\n", sts);
    else
        printf("Dimension Year has Member Name Uniqueness value: %ld\n", pbNameUnique);
}

if (!sts)
{
    sts = EssOtlFindMember(hOutline, "Product", &hDim);

    if (sts)
        printf("EssOtlFindMember failed sts %ld\n", sts);
}

if (!sts)
{
    /*set Product to prohibit duplicate (non-unique) member names*/
    pbNameUnique = ESS_TRUE;
    sts = EssOtlSetDimensionNameUniqueness (hOutline, hDim, pbNameUnique);

    if (sts)
        printf("EssOtlSetDimensionNameUniqueness failed sts %ld\n", sts);
    else
        printf("Dimension Product has Member Name Uniqueness value: %ld\n", pbNameUnique);
}

return sts;


See Also

- EssOtlSetDimensionNameUniqueness

**EssOtlGetDimensionSolveOrder**

Returns the solve order of a dimension.
Syntax

```c
ESS_FUNC_M EssOtlGetDimensionSolveOrder (hOutline, hMember, pOrder);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline context handle (input).</td>
</tr>
<tr>
<td>hMember</td>
<td>ESS_PHMEMBER_T</td>
<td>Dimension handle (input).</td>
</tr>
<tr>
<td>pOrder</td>
<td>ESS_PUCHAR_T</td>
<td>Solve order (output).</td>
</tr>
</tbody>
</table>

Notes

Solve order is applicable only to aggregate storage databases.

Return Value

Returns 0 if successful.

Example

```c
ESS_STS_T         sts = ESS_STS_NOERR;
ESS_HOUTLINE_T    hOutline = ESS_NULL;
ESS_HMEMBER_T     hMember = ESS_NULL;
ESS_UCHAR_T       ucOrder = 0;
/* code to assign hOutline variable omitted */
/* code to assign hMember variable omitted */

if (hOutline && hMember)
{
    sts = EssOtlGetDimensionSolveOrder(hOutline, hMember, &ucOrder);

    if (sts)
        printf("Error [%ld] returned\n", sts);
    else
        printf("Solve Order: %d\n", ucOrder);
} else
    printf("Both hOutline and hMember must have values\n");
```

See Also

- EssOtlSetDimensionSolveOrder
- EssOtlGetMemberSolveOrder
- EssOtlSetMemberSolveOrder

EssOtlGetDimensionUserAttributes

Returns the user defined attributes used in the specified dimension.

Syntax

```c
ESS_FUNC_M EssOtlGetDimensionUserAttributes (hOutline, pPredicate, pCounts, ppAttributeNames);
```
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Essbase outline handle. This must have been returned from EssOtlOpenOutlineQuery().</td>
</tr>
<tr>
<td>pPredicate</td>
<td>“ESS_PREDICATE_T” on page 709</td>
<td>Structure defining the query. The fields of this structure are used as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ulQuery—Value defining the operation to perform. The only valid value is ESS_DIMUSERATTRIBUTES.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- pszDimension—Dimension to limit the scope of the query. Specify a valid dimension name.</td>
</tr>
<tr>
<td>pCounts</td>
<td>“ESS_MBRCOUNTS_T” on page 699</td>
<td>Structure defining information about counts It contains the following fields:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ulStart—Starting number to return</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ulMaxCount—Maximum number of member names to return</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ulTotalCount—Total number of members that are defined in the results of the query</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- pulReturnCount—Number of member names returned in this query</td>
</tr>
<tr>
<td>ppAttributeNames</td>
<td>ESS_PPMBRNAME_T</td>
<td>An array of attribute names returned from the query.</td>
</tr>
</tbody>
</table>

**Notes**

- This function is used only to get the user's defined attributes on a specific dimension. Therefore, the only valid value for Predicate is ESS_DIMUSERATTRIBUTES_T.
- Solve order property on a member or dimension specifies its calculation order.
- Member solve order takes precedence over dimension solve order. Solve order can be between 0 and 127. The default is 0.
- Members without formulas that do not have a specified solve order inherit the solve order of their dimension. Members with formulas that do not have a specified solve order have a solve order of zero.

**Return Value**

The return value is zero if the function was successful.

**Example**

```c
#include <essapi.h>
#include <essotl.h>

ESS_STS_T          sts = ESS_STS_NOERR;
ESS_HOUTLINE_T     hOutline;
ESS_OBJDEF_T       Object;
ESS_PREDICATE_T    Predicate;
ESS_MBRCOUNTS_T    Counts;
ESS_MBRNAME_T      pAttribNames;
ESS_ULONG_T        i;
ESS_ACCESS_T       Access;
ESS_STR_T         AppName;
ESS_STR_T          DbName;
```
AppName = "Sample";
DbName = "Basic";

sts = EssSetActive(hCtx, AppName, DbName, &Access);

if ( sts == 0)
{
    memset(&Object, '\0', sizeof(Object));
    sts = EssOtlOpenOutlineQuery(hCtx, &Object, &hOutline);

    memset(&Predicate, '\0', sizeof(Predicate));
    Predicate.ulQuery       = ESS_DIMUSERATTRIBUTES;
    Predicate.pszDimension  = "Market";

    memset(&Counts, '\0', sizeof(Counts));
    Counts.ulStart    = 0;
    Counts.ulMaxCount = 10;

    if(!sts)
    {
        sts = EssOtlGetDimensionUserAttributes(hOutline,
                                         &Predicate, &Counts, &pAttribNames);

        if (!sts && Counts.ulReturnCount)
        {
            sts = EssFree(hInstance, pAttribNames);
        }
    }
}

See Also

- EssFree
- EssOtlOpenOutlineQuery
- EssOtlQueryMembers
- EssOtlQueryMembersByName

EssOtlGetDTSMemberAlias

Gets an alias name for a Dynamic Time Series (DTS) member.

Syntax

ESS_STS_T EssOtlGetDTSMemberAlias (hOutline, pszDTSMember, pszAliasTable, ppszAlias);

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>The Essbase outline handle returned from the EssOtlOpenOutline call.</td>
</tr>
<tr>
<td>pszDTSMember</td>
<td>ESS_STR_T</td>
<td>Name of the DTS member which provides the alias.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>------------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>pszAliasTable</td>
<td>ESS_STR_T</td>
<td>Name of the alias table which provides the alias. If NULL, the default alias table is used.</td>
</tr>
<tr>
<td>ppszAlias</td>
<td>ESS_PSTR_T</td>
<td>Pointer to a pointer to a C string containing the alias name for the DTS member.</td>
</tr>
</tbody>
</table>

**Return Value**

If successful the return value is zero. Otherwise, one of the following is returned:

- OTLAPI_ERR_DTSMBRNOTDEFINED
- OTLAPI_BAD_ALIASTABLE

**Example**

```c
#include "essapi.h"
#include "essotl.h"
#include "esserror.h"

ESS_STS_T ESS_OtlGetDTSMemberAlias(ESS_HCTX_T hCtx)
{
    ESS_STS_T               sts = ESS_STS_NOERR;
    ESS_OBJDEF_T            Object;
    ESS_HOUTLINE_T          hOutline;
    ESS_APPNAME_T           szAppName;
    ESS_DBNAME_T            szDbName;
    ESS_OBJNAME_T           szFileName;
    ESS_CHAR_T              pszAliasTable[ESS_ALIASNAMELEN];
    ESS_STR_T               pszAlias;
    ESS_CHAR_T              pszDTSMember[ESS_MBRNAMELEN];

    strcpy(szAppName, "sample");
    strcpy(szDbName, "Basic");
    strcpy(szFileName, "Basic");
    strcpy(pszDTSMember, "Q-T-D");
    strcpy(pszAliasTable, "Default");

    Object.hCtx = hCtx;
    Object.ObjType = ESS_OBJTYPE_OUTLINE;
    Object.AppName = szAppName;
    Object.DbName = szDbName;
    Object.FileName = szFileName;

    sts = EssOtlOpenOutline(hCtx, &Object, ESS_FALSE, ESS_TRUE, &hOutline);

    if(sts)
    {
        printf("Could not open outline\n");
        return sts;
    }

    sts = ESS_OtlGetDTSMemberAlias(hOutline, pszDTSMember, pszAliasTable, &pszAlias);
    if(sts)
    {
        printf("Could not get DTS member alias\n");
        return sts;
    }
}
```
 EssOtlGetEnabledDTSMembers

Retrieves the member information structures for the enabled Dynamic Time Series (DTS) members in the specified outline.

Syntax

ESS_STS_T EssOtlGetEnabledDTSMembers (hOutline, pusCount, ppEnabledDTSMemberList);

Parameter | Data Type | Description
--- | --- | ---
 hOutline | ESS_HOUTLINE_T | The Essbase outline handle returned from the EssOtlOpenOutline call.
 pusCount | ESS_PUSHORT_T | The number of enabled DTS Members.
 ppEnabledDTSMemberList | ESS_PPDTSMBRINFO_T | Pointer to an array of DTS member info structures (for the enabled DTS members for the outline).

Notes

This function also fills in the ESS_DTSMBRNAME_T structure passed to it.

Return Value

If successful the return value is zero. Otherwise, returns the status of the EssOtlQueryMembers() call.

Example

```c
#include "essapi.h"
#include "essotl.h"
#include "esserror.h"

ESS_STS_T ESS_OtlGetEnabledDTSMembers(ESS_HCTX_T hCtx) {
    ESS_STS_T       sts =ESS_STS_NOERR;
    ESS_HOUTLINE_T  hOutline;
    ESS_OBJDEF_T    Object;
    ESS_APPNAME_T   szAppName;
    ESS_DBNAME_T    szDbName;
    ESS_OBJNAME_T   szFileName;
    ESS_USHORT_T    usCount, i;
```
ESS_PDTSMBRNAME_T pEnabledDTSMbrList;

strcpy(szAppName, "Sample");
strcpy(szDbName, "Basic");
strcpy(szFileName, "Basic");

memset(&Object, '\0', sizeof(ESS_OBJDEF_T));
Object.hCtx = hCtx;
Object.ObjType = ESS_OBJTYPE_OUTLINE;
Object.AppName = szAppName;
Object.DbName = szDbName;
Object.FileName = szFileName;

sts = EssOtlOpenOutline(hCtx, &Object, ESS_FALSE, ESS_TRUE, &hOutline);
if(sts)
{
    printf("Could not open outline\n");
    return sts;
}

sts = EssOtlGetEnabledDTSMembers(hOutline, &usCount, &pEnabledDTSMbrList);
if(sts)
{
    printf("Could not get enabled DTS member alias\n");
}
else
{
    printf("No of enabled DTS members is %u\n", usCount);
    for (i = 0; i < usCount; i++)
    {
        printf("%s\n", pEnabledDTSMbrList[i]);
    }
}
EssOtlCloseOutline(hOutline);
return sts;

See Also

- EssOtlDeleteDTSMemberAlias
- EssOtlEnableDTSMember
- EssOtlGetDTSMemberAlias
- EssOtlSetDTSMemberAlias

**EssOtlGetFirstMember**

Returns a member handle to the first member in the outline. The first member is the first dimension defined in the outline.

**Syntax**

ESS_FUNC_M EssOtlGetFirstMember (hOutline, phMember);
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline context handle.</td>
</tr>
<tr>
<td>phMember</td>
<td>ESS_PHMEMBER_T</td>
<td>Pointer for return of a member handle of the first member in the outline. This parameter is passed to subsequent calls for traversing the outline.</td>
</tr>
</tbody>
</table>

**Return Value**

Returns 0 if successful.

**Example**

```c
#include <essapi.h>
#include <essotl.h>

ESS_STS_T          sts = 0;
ESS_HOUTLINE_T     hOutline;
ESS_MEMBER_T       hMemberFirst
ESS_OBJDEF_T       Object;
ESS_APPNAME_T      szAppName;
ESS_DBNAME_T       szDbName;
ESS_OBJNAME_T      szFileName;
memset(&Object, '\0', sizeof(Object));
Object.hCtx = hCtx;
Object.ObjType = ESS_OBJTYPE_OUTLINE;
strcpy(szAppName, "Sample");
strcpy(szDbName, "Basic");
strcpy(szFileName, "Basic");
Object.AppName = szAppName;
Object.DbName = szDbName;
Object.FileName = szFileName;

sts = EssOtlOpenOutline(hCtx, &Object, ESS_TRUE,
                        ESS_TRUE, &hOutline);
if (!sts)
{
    sts = EssOtlGetFirstMember(hOutline,
                                &hMemberFirst);
}
```

**See Also**

- EssOtlGetParent
- EssOtlGetNextSibling
- EssOtlGetPrevSibling
- EssOtlGetChild

**EssOtlGetGenName**

Retrieves the name for a specific generation within a dimension.

**Syntax**

```c
ESS_FUNC_M EssOtlGetGenName (hOutline, pszDimension, usGen, ppszName);
```
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline context handle.</td>
</tr>
<tr>
<td>pszDimension</td>
<td>ESS_STR_T</td>
<td>Name of dimension that contains the generation.</td>
</tr>
<tr>
<td>usGen</td>
<td>ESS_USHORT_T</td>
<td>Number of generation for which to get a name. The dimension is generation 1.</td>
</tr>
<tr>
<td>ppszName</td>
<td>ESS_PSTR_T</td>
<td>Buffer for return of generation name, allocated by the API.</td>
</tr>
</tbody>
</table>

**Notes**

- The generation name follows the same rules as a member name and must be unique across the entire member name space. It cannot duplicate any other generation, level, member name, or alias. Attempting to add a duplicate name generates an error.
- Generation names are not automatically assigned. For this function to return the name, a name must have been assigned. The name can be assigned with `EssOtlSetGenName`.
- Call `EssFree()` to free the returned buffer.

**Return Value**

Returns 0 if successful; otherwise one of the following:

- OTLAPI_NO_GENLEVELNAME
- OTLAPI_ERR_NOTADIM
- OTLAPI_ERR_GENLEVELNAMEMBR

**Example**

```c
#include <essapi.h>
#include <essotl.h>

ESS_STS_T          sts = 0;
ESS_HOUTLINE_T     hOutline;
ESS_OBJDEF_T       Object;
ESS_APPNAME_T      szAppName;
ESS_DBNAME_T       szDbName;
ESS_OBJNAME_T      szFileName;
ESS_STR_T          Dimension;
ESS_USHORT_T       GenNum;
ESS_STR_T          GenName;
memset(&Object, '\0', sizeof(Object));
Object.hCtx = hCtx;
Object.ObjType = ESS_OBJTYPE_OUTLINE;
strcpy(szAppName, "Sample");
strcpy(szDbName, "Basic");
strcpy(szFileName, "Basic");
Object.AppName = szAppName;
Object.DbName = szDbName;
Object.FileName = szFileName;

sts = EssOtlOpenOutline(hCtx, &Object, ESS_TRUE, ESS_TRUE, &hOutline);
```
Dimension = "Year";
GenNum = 3;

if (!sts)
{
    sts = EssOtlGetGenName(hOutline, Dimension, GenNum, &GenName);
}

if (!sts && GenName)
{
    printf("Gen Name: %s\n", GenName);
    EssFree(hInst, GenName);
}

See Also

- EssFree
- EssOtlDeleteGenName
- EssOtlSetGenName

**EssOtlGetGenNameEx**

Retrieves the name and member uniqueness setting for a specific generation within a dimension.

**Syntax**

ESS_FUNC_M EssOtlGetGenName (hOutline, pszDimension, usGen, ppszName, pbNameUnique);

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline context handle.</td>
</tr>
<tr>
<td>pszDimension</td>
<td>ESS_STR_T</td>
<td>Name of dimension that contains the generation.</td>
</tr>
<tr>
<td>usGen</td>
<td>ESS_USHORT_T</td>
<td>Number of generation for which to get a name.</td>
</tr>
<tr>
<td>ppszName</td>
<td>ESS_PSTR_T</td>
<td>Buffer for return of generation name, allocated</td>
</tr>
<tr>
<td>pbNameUnique</td>
<td>ESS_PBOOL_T</td>
<td>Member name uniqueness setting.</td>
</tr>
</tbody>
</table>

**Notes**

- The generation name must be unique across the entire member name space. It cannot
duplicate any other generation, level, member name, or alias. Attempting to add a duplicate
generation name generates an error.
- Generation names are not automatically assigned. For this function to return the name, a
name must have been assigned. The name can be assigned with EssOtlSetGenName
- Call EssFree() to free the returned buffer.

**Return Value**

Returns 0 if successful; otherwise, returns an error code.
Example

void ESS_GetGenNameEx()
{

    ESS_STS_T             sts = 0;
    ESS_HOUTLINE_T        hOutline;
    ESS_OBJDEF_T          Object;
    ESS_APPNAME_T         szAppName;
    ESS_DBNAME_T          szDbName;
    ESS_OBJNAME_T         szFileName;
    ESS_STR_T             Dimension;
    ESS_USHORT_T          GenNum;
    ESS_STR_T             GenName;
    ESS_BOOL_T            bUnique = ESS_FALSE;
    memset(&Object, '\0', sizeof(Object));
    Object.hCtx = hCtx;
    Object.ObjType = ESS_OBJTYPE_OUTLINE;
    strcpy(szAppName, "Demo");
    strcpy(szDbName, "Test");
    strcpy(szFileName, "Test");
    Object.AppName = szAppName;
    Object.DbName = szDbName;
    Object.FileName = szFileName;

    sts = EssOtlOpenOutline(hCtx, &Object, ESS_TRUE,
                ESS_TRUE, &hOutline);
    printf("EssOtlOpenOutline sts: %ld\n",sts);

    /***************  Set and Get GenName **************/
    Dimension = "Year";
    GenNum = 1;
    GenName = "Gen 1 Year"

    //SetGenNameEx() so that Gen 1 members of Year cannot be non-unique
    if (!sts)
    {
        sts = EssOtlSetGenNameEx(hOutline, Dimension,
                GenNum, GenName, ESS_TRUE);
    }

    // GetGenNameEx() to see if the gen is able to be non-unique
    if (!sts)
    {
        sts = EssOtlGetGenNameEx(hOutline, Dimension,
                GenNum, &GenName, &bUnique);
        printf("Generation 1 members of Year have bUnique value of %ld\n", bUnique);
        printf("EssOtlGetGenNameEx sts: %ld\n",sts);
    }

    if (!sts && GenName)
    {
        printf("Gen Name: %s\n",GenName);
    }
EssFree(hInst, GenName);
}

See Also
- EssOtlGetGenName
- EssFree
- EssOtlDeleteGenName
- EssOtlSetGenNameEx

EssOtlGetGenNames
Retrieves all generation names specified for a particular dimension.

Syntax

```c
ESS_FUNC_M EssOtlGetGenNames (hOutline, pszDimension, ulOptions, pulCount, pNameArray);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Essbase outline handle.</td>
</tr>
<tr>
<td>pszDimension</td>
<td>ESS_STR_T</td>
<td>The dimension to retrieve generation names for.</td>
</tr>
<tr>
<td>ulOptions</td>
<td>ESS_ULONG_T</td>
<td>This can be one of the following values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_GENLEV_ALL—Returns default and actual generation names.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_GENLEV_ACTUAL—Returns only generation names that are actually defined.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_GENLEV_DEFAULT—Returns all default generation names. This includes the default names for generations that have an actual name.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_GENLEV_NOACTUAL—Returns default generation names. This includes only the generations that don’t have an actual generation name.</td>
</tr>
<tr>
<td>pulCount</td>
<td>ESS_PULONG_T</td>
<td>Return of the number of elements in the pNameArray. It is the number of generation names for the specified member.</td>
</tr>
<tr>
<td>pNameArray</td>
<td>&quot;ESS_GENLEVELNAME_T&quot; on page 698</td>
<td>An array of generation name structures for the specified dimension.</td>
</tr>
</tbody>
</table>

Notes
- The caller should free the pNameArray structure after use by calling EssFree().
- This call will work for both EssOtlOpenOutline() and EssOtlOpenOutlineQuery(). The information will exist locally for both, since it is returned from the server during the EssOtlOpenOutlineQuery() call.

Return Value
The return value is zero if the function was successful.
Example

```c
#include <essapi.h>
#include <essotl.h>

ESS_STS_T        stst = ESS_STS_NOERR;
ESS_HOUTLINE_T    hOutline;
ESS_OBJDEF_T      Object;
ESS_STR_T         Dimension;
ESS_UINTEGER_T    GenOpt;
ESS_UINTEGER_T    pCount = 0, i;
ESS_PGENLEVELNAME_T pNameArray = ESS_NULL;
ESS_ACCESS_T      Access;
ESS_STR_T         AppName;
ESS_STR_T         DbName;

AppName = "Sample";
DbName = "Basic";

sts=EssSetActive(hCtx, AppName, DbName, &Access);

if (sts == 0)
{
    memset(&Object, '\0', sizeof(Object));

    stst = EssOtlOpenOutlineQuery(hCtx, &Object, &hOutline);

    Dimension = "Year";
    GenOpt = ESS_GENLEV_ALL;

    if (!sts)
    {
        stst = EssOtlGetGenNames(hOutline, Dimension, GenOpt, &Count, &pNameArray);

        if(!stst && Count )
        {
            for(i = 0; i<Count; i++)
            {
                printf("\nNumber %ld, Name %s ",
                        pNameArray[i].usNumber, pNameArray[i].szName);
            }
            EssFree(hInst, pNameArray);
        }
    }
}
```

See Also

- EssFree
- EssOtlGetGenName
- EssOtlGetLevelName
- EssOtlGetLevelNames
- EssOtlOpenOutline
- EssOtlOpenOutlineQuery
**EssOtlGetHierarchyType**

Gets the dimension's hierarchy type designation: Multiple hierarchies enabled, dynamic hierarchy, or stored hierarchy.

**Syntax**

```c
ESS_FUNC_M EssOtlGetHierarchyType(hOutline, hMember, pType);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline context handle (input).</td>
</tr>
<tr>
<td>hMember</td>
<td>ESS_HMEMBER_T</td>
<td>A dimension member (input).</td>
</tr>
<tr>
<td>pType</td>
<td>ESS_UCHAR_T</td>
<td>If <code>hMember</code> is a dimension member, one of the following values (output):</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_STORED_HIERARCHY—The dimension is a single, stored hierarchy.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_DYNAMIC_HIERARCHY—The dimension is a single, dynamic hierarchy.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_MULTIPLE_HIERARCHY_IS_ENABLED—The dimension is multiple-hierarchy enabled.</td>
</tr>
</tbody>
</table>

See Notes.

**Notes**

- Once the dimension is multiple-hierarchy enabled, the hierarchy types are determined by the generation 2 members. If `hMember` is a generation 2 member, `pType` can return the following values:
  - ESS_STORED_HIERARCHY—The hierarchy with `hMember` as top is a single, stored hierarchy.
  - ESS_DYNAMIC_HIERARCHY—The hierarchy with `hMember` as top is a single, dynamic hierarchy.
  - ESS_MULTIPLE_HIERARCHY_NOT_ENABLED—The dimension is not multiple-hierarchy enabled.
- If `hMember` is of a generation greater than 2, `pType` returns ESS_NOT_HIERARCHY_MEMBER.

**Return Value**

Returns 0 if successful; otherwise, returns an error.

**See Also**

- `EssOtlSetHierarchyType`
- `EssOtlSetAltHierarchyEnabled`
- `EssOtlGetAltHierarchyEnabled`

**EssOtlGetImpliedShare**

Returns the Implied Share setting of an outline.
Syntax

```
ESS_FUNC_M EssOtlGetImpliedShare(hOutline, &impliedShareSetting);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline context handle</td>
</tr>
<tr>
<td>&amp;impliedShareSetting</td>
<td>ESS_USHORT</td>
<td>Address of an implied share setting.</td>
</tr>
</tbody>
</table>

**Return Value**

- 0—If successful
- Error number—If unsuccessful

The implied share setting value. See “Implied Share Setting (C)” on page 103.

**See Also**

- `EssOtlSetImpliedShare`

---

### EssOtlGetLevelName

Gets the name for a specific level within a dimension.

Syntax

```
ESS_FUNC_M EssOtlGetLevelName(hOutline, pszDimension, usLevel, pszName);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline context handle</td>
</tr>
<tr>
<td>pszDimension</td>
<td>ESS_STR_T</td>
<td>Name of dimension that contains the generation.</td>
</tr>
<tr>
<td>usLevel</td>
<td>ESS_USHORT_T</td>
<td>Number of level number for which to get a name.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Leaf members are level 0.</td>
</tr>
<tr>
<td>pszName</td>
<td>ESS_PSTR_T</td>
<td>Buffer for return of the level of the specified</td>
</tr>
<tr>
<td></td>
<td></td>
<td>dimension, allocated by the API.</td>
</tr>
</tbody>
</table>

**Notes**

- In C programs, call `EssFree()` to free the returned buffer.
- Level names are not automatically assigned. For this function to return the name, a name must have been assigned. The name can be assigned with `EssOtlSetLevelName`

**Return Value**

Returns 0 if successful; otherwise one of the following:

- OTLAPI_NO_GENLEVELNAME
- OTLAPI_ERR_NOTADIM
Example

```c
#include <essapi.h>
#include <essotl.h>

ESS_STS_T          sts = 0;
ESS_HOUTLINE_T     hOutline;
ESS_OBJDEF_T       Object;
ESS_APPNAME_T      szAppName;
ESS_DBNAME_T       szDbName;
ESS_OBJNAME_T      szFileName;
ESS_STR_T          Dimension;
ESS_USHORT_T       LevelNum;
ESS_STR_T          LevelName;

memset(&Object, '\0', sizeof(Object));
Object.hCtx = hCtx;
Object.ObjType = ESS_OBJTYPE_OUTLINE;
strcpy(szAppName, "Sample");
strcpy(szDbName, "Basic");
strcpy(szFileName, "Basic");
Object.AppName = szAppName;
Object.DbName = szDbName;
Object.FileName = szFileName;

sts = EssOtlOpenOutline(hCtx, &Object, ESS_TRUE,
                        ESS_TRUE, &hOutline);

/******* Get Level Name ***********/
Dimension = "Year";
LevelNum = 0;

if (!sts)
{
    sts = EssOtlGetLevelName(hOutline, Dimension,
                               LevelNum, &LevelName);
}

if (!sts && LevelName)
{
    printf("Level Name: %s\n", LevelName);
    EssFree(hInst, LevelName);
}
```

See Also

- EssOtlSetLevelName
- EssOtlDeleteLevelName
- EssOtlSetGenName

**EssOtlGetLevelNameEx**

Returns the member-name uniqueness setting for a specific level within a dimension.
### Syntax

```
ESS_FUNC_M EssOtlGetLevelNameEx (hOutline, pszDimension, usLevel, pszName, pbNameUnique);
```

### Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline context handle.</td>
</tr>
<tr>
<td>pszDimension</td>
<td>ESS_STR_T</td>
<td>Name of dimension that contains the generation.</td>
</tr>
<tr>
<td>usLevel</td>
<td>ESS_USHORT_T</td>
<td>Number of level number for which to get a name. Leaf members are level 0.</td>
</tr>
<tr>
<td>pszName</td>
<td>ESS_PSTR_T</td>
<td>Buffer for return of the level of the specified dimension, allocated by the API (output).</td>
</tr>
<tr>
<td>pbNameUnique</td>
<td>ESS_PBOOL_T</td>
<td>The member-name uniqueness setting (output).</td>
</tr>
</tbody>
</table>

### Notes

- In C programs, call EssFree() to free the returned buffer.
- Level names are not automatically assigned. For this function to return the name, a name must have been assigned. The name can be assigned with `EssOtlSetLevelName`.
- This function gets the member-name uniqueness information for the level. If you want to change the member-name uniqueness setting, use `EssOtlSetLevelNameEx`.

### Return Value

Returns 0 if successful; otherwise, returns an error code.

### Example

```c
ESS_FUNC_M
ESS_GetLevelNameEx()
{
    ESS_STS_T          sts = 0;
    ESS_HOUTLINE_T     hOutline;
    ESS_OBJDEF_T       Object;
    ESS_APPNAME_T      szAppName;
    ESS_DBNAME_T       szDbName;
    ESS_OBJNAME_T      szFileName;
    ESS_STR_T          Dimension;
    ESS_USHORT_T       LevelNum;
    ESS_STR_T          LevelName;
    ESS_BOOL_T         bUnique = ESS_FALSE;
    memset(&Object, '\0', sizeof(Object));
    Object.hCtx = hCtx;
    Object.ObjType = ESS_OBJTYPE_OUTLINE;
    strcpy(szAppName, "Demo");
    strcpy(szDbName, "Test");
    strcpy(szFileName, "Test");
    Object.AppName = szAppName;
    Object.DbName = szDbName;
    Object.FileName = szFileName;
    sts = EssOtlOpenOutline(hCtx, &Object, ESS_TRUE, ESS_TRUE, &hOutline);
}
```

816  C Outline API Functions
Set and Get Level Name

```c
Dimension = "Year";
LevelNum = 0;
LevelName = "Level 0 Year";

//SetLevelNameEx() so that level 0 member of Year cannot be non-unique
if (!sts)
{
    sts = EssOtlSetLevelNameEx(hOutline, Dimension,
                                LevelNum, LevelName, ESS_TRUE);
}

// GetLevelNameEx() to see if the level is able to be non-unique
if (!sts)
{
    sts = EssOtlGetLevelNameEx(hOutline, Dimension,
                                LevelNum, &LevelName, &bUnique);
    printf("Level 0 members of Year have bUnique value of %ld\n", bUnique);
}

if (!sts && LevelName)
{
    printf("Level Name: %s\n", LevelName);
    EssFree(hInst, LevelName);
}

return (sts);
```

See Also
- EssOtlSetLevelNameEx

EssOtlGetLevelNames

Retrieves all level names specified for a particular dimension.

**Syntax**

```
ESS_FUNC_M EssOtlGetLevelNames (hOutline, pszDimension, ulOptions, pulCount, pNameArray);
```

**Parameter** | **Data Type** | **Description**
--- | --- | ---
`hOutline` | ESS_HOUTLINE_T | Essbase outline handle.
`pszDimension` | ESS_STR_T | The dimension to retrieve level names for.
### Parameter Data Type Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ulOptions</td>
<td>ESS_ULONG_T</td>
<td>This can be one of the following values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_GENLEV_ALL—Returns default and actual level names</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_GENLEV_ACTUAL—Returns only level names that are actually defined</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_GENLEV_DEFAULT—Returns all default level names. This includes the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>default names for levels that have an actual name.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_GENLEV_NOACTUAL—Returns default level names. This includes only the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>levels that don’t have an actual level name.</td>
</tr>
<tr>
<td>pulCount</td>
<td>ESS_PULONG_T</td>
<td>Return of the number of elements in the pNameArray. It is the number of</td>
</tr>
<tr>
<td></td>
<td></td>
<td>level names for the specified member.</td>
</tr>
<tr>
<td>pNameArray</td>
<td>“ESS_GENLEVELNAME_T”</td>
<td>An array of level name structures for the specified dimension.</td>
</tr>
</tbody>
</table>

### Notes
- The caller should free the pNameArray structure after use by calling EssFree().
- This call will work for both EssOtlOpenOutline() and EssOtlOpenOutlineQuery(). The information will exist locally for both, since it is returned from the server during the EssOtlOpenOutlineQuery() call.

### Return Value
The return value is zero if the function was successful.

### Example
```c
#include <essapi.h>
#include <essotl.h>

ESS_STS_T          sts = ESS_STS_NOERR;
ESS_HOUTLINE_T     hOutline;
ESS_OBJDEF_T       Object;
ESS_STR_T          Dimension;
ESS_ULONG_T        LevOpt;
ESS_ULONG_T        pCount = 0, i;
ESS_PGENLEVELNAME_T pNameArray = ESS_NULL;
ESS_ACCESS_T       Access;
ESS_STR_T          AppName;
ESS_STR_T          DbName;

AppName = "Sample";
DbName = "Basic";

sts = EssSetActive(hCtx, AppName, DbName, &Access);
if (sts == 0)
{
    memset(&Object, '\0', sizeof(Object));
    sts = EssOtlOpenOutlineQuery(hCtx, &Object, &hOutline);
```
Dimension = "Year";
LevOpt = ESS_GENLEV_ALL;

if (!sts)
{
    sts = EssOtlGetLevelNames(hOutline, Dimension,
    LevOpt, &Count, &pNameArray);

    if(!sts && Count )
    {
        for(i = 0; i<Count; i++)
        {
            printf("\nNumber %ld, Name %s ",
            pNameArray[i].usNumber, pNameArray[i].szName);
        }
        EssFree(hInst, pNameArray);
    }
}

See Also

- EssFree
- EssOtlGetGenName
- EssOtlGetGenNames
- EssOtlGetLevelName
- EssOtlOpenOutline
- EssOtlOpenOutlineQuery

### EssOtlGetLinkedAttributeAttachLevel

Gets the attachment level for a linked attribute dimension.

Linked attribute dimensions enable periodic comparisons between members in a date-time dimension. Every linked attribute has an association level and an attachment level associated with the attribute dimension definition.

The attachment level is always the longer of the two periods in the periodic comparison represented by a linked attribute dimension. For example, in the linked attribute dimension Quarter by Year, Year is the attachment level, and Quarter is the association level.

**Syntax**

```c
ESS_FUNC_M EssOtlGetLinkedAttributeAttachLevel (hOutline, hDimMember, psLevel);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline context handle (input).</td>
</tr>
<tr>
<td>hDimMember</td>
<td>ESS_HMEMBER_T</td>
<td>Linked attribute dimension member handle (input).</td>
</tr>
<tr>
<td>psLevel</td>
<td>ESS_PUSHORT_T</td>
<td>The linked attribute attachment level (output).</td>
</tr>
</tbody>
</table>
Notes

- Before you call this function, open the outline in edit or query mode using either EssOtlOpenOutline or EssOtlOpenOutlineQuery.
- This function is only applicable when hDimMember is a linked attribute dimension.

Return Value

Returns 0 if successful; otherwise, returns an error.

Example

```c
#include <essapi.h>
#include <essotl.h>

ESS_STS_T          sts = 0;
ESS_OBJDEF_T       Object;
ESS_HOUTLINE_T     hOutline;
ESS_HMEMBER        hDimMember;
ESS_USHORT_T       usAttachLevel;
ESS_APPNAME_T      szAppName;
ESS_DBNAME_T       szDbName;
ESS_OBJNAME_T      szFileName;
memset(&Object, '\0', sizeof(Object));
Object.hCtx = hCtx;
Object.ObjType = ESS_OBJTYPE_OUTLINE;
strcpy(szAppName, "Sample");
strcpy(szDbName, "Basic");
strcpy(szFileName, "Basic");
Object.AppName = szAppName;
Object.DbName = szDbName;
Object.FileName = szFileName;

sts = EssOtlOpenOutline(hCtx, &Object, ESS_TRUE,
                         ESS_TRUE, &hOutline);

if (!sts)
{
    sts = EssOtlFindMember(hOutline, "Quarter By Year",
                           &hDimMember);
}

if (!sts && hMemberJan)
{
    sts = EssOtlGetLinkedAttributeAttachLevel(hOutline,
                                               hDimMember, &usAttachLevel);
}
```

See Also

- EssOtlGetAttributeAssocLevel
- EssOtlQueryGenerationInfo
EssOtlGetMemberAlias

Gets the default member alias for the specified member in the specified alias table.

Syntax

```c
ESS_FUNC_M EssOtlGetMemberAlias (hOutline, hMember, pszAliasTable, ppszAlias);
```

Parameter | Data Type | Description
---|---|---
hOutline | ESS_HOUTLINE_T | Outline context handle.
hMember | ESS_HMEMBER_T | Handle of member to get the alias for.
pszAliasTable | ESS_STR_T | Alias table to get the alias from. If this parameter is ESS_NULL, the default alias table is used.
ppszAlias | ESS_PSTR_T | Buffer for the return of the alias. The buffer is allocated by the API.

Notes

- Use `EssFree()` to free the alias buffer.

Return Value

Returns 0 if successful; otherwise:

OTLAPI_BAD_ALIASTABLE

Example

```c
#include <essapi.h>
#include <essotl.h>

ESS_STS_T          sts = 0;
ESS_OBJDEF_T       Object;
ESS_HOUTLINE_T     hOutline;
ESS_HMEMBER_T      hMember;
ESS_STR_T          pszAlias;
memset(&Object, '\0', sizeof(Object));
Object.hCtx = hCtx;
Object.ObjType = ESS_OBJTYPE_OUTLINE;
strcpy(szAppName, "Sample");
strcpy(szDbName, "Basic");
strcpy(szFileName, "Basic");
Object.AppName = szAppName;
Object.DbName = szDbName;
Object.FileName = szFileName;

sts = EssOtlOpenOutline(hCtx, &Object, ESS_TRUE,
                        ESS_TRUE, &hOutline);

if (!sts)
{
    sts = EssOtlFindMember(hOutline, "100",
                           &hMember);
}

if (!sts && hMember)
```
sts = EssOtlGetMemberAlias(hOutline, hMember, ESS_NULL, &pszAlias);
}

if (pszAlias)
{
    EssFree(hInst, pszAlias);
}

See Also

- EssOtlSetMemberAlias
- EssOtlDeleteMemberAlias

EssOtlGetMemberCommentEx

Gets the extended comment for the specified member.

Syntax

ESS_FUNC_M EssOtlGetMemberCommentEx (hOutline, hMember, pszCommentEx);

Parameter | Data Type          | Description
-----------|--------------------|-----------------
           | ESS_HOUTLINE_T    | Outline context handle
hOutline   | ESS_HMEMBER_T     | Member handle.
hMember    | ESS_PSTR_T        | Variable for the return of the extended comment. This buffer is allocated by the API.
pszCommentEx

Notes

- Use EssFree() to release the buffer containing the extended member comment.

Return Value

Returns 0 if successful.

Example

#include <essapi.h>
#include <essotl.h>

ESS_STS_T st = 0;
ESS_OBJDEF_T Object;
ESS_HOUTLINE_T hOutline;
ESS_HMEMBER_T hMember;
ESS_STR_T pszCommentEx = ESS_NULL;
ESS_APPNAME_T szAppName;
ESS_DBNAME_T szDbName;
ESS_OBJNAME_T szFileName;

memset(&Object, '\0', sizeof(Object));
Object.hCtx = hCtx;
Object.ObjType = ESS_OBJTYPE_OUTLINE;
strcpy(szAppName, "Sample");
strcpy(szDbName, "Basic");
strcpy(szFileName, "Basic");
Object.AppName = szAppName;
Object.DbName = szDbName;
Object.FileName = szFileName;

sts = EssOtlOpenOutline(hCtx, &Object, ESS_TRUE, ESS_TRUE, &hOutline);

if (!sts)
{
    sts = EssOtlFindMember(hOutline, "Variance", &hMember);
}

if (!sts && hMember)
{
    sts = EssOtlGetMemberCommentEx(hOutline, hMember, &pszCommentEx);
}

if (pszCommentEx)
{
    EssFree(hInst, pszCommentEx);
}

See Also
● EssFree
● EssOtlOpenOutline
● EssOtlSetMemberCommentEx

EssOtlGetMemberField

Returns data for the specified field of a specified outline member.

Syntax

```
ESS_FUNC_M EssOtlGetMemberField(hOutline, hMember, MbrFieldID, ppFieldElement);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Essbase outline handle. This must have been returned from EssOtlOpenOutlineQuery().</td>
</tr>
<tr>
<td>hMember</td>
<td>ESS_HMEMBER_T</td>
<td>Member handle returned by EssOtlQueryMembersEx().</td>
</tr>
<tr>
<td>MbrFieldID</td>
<td>ESS_UINTEGER_T</td>
<td>A member field identifier constant. See Notes.</td>
</tr>
<tr>
<td>ppFieldElement</td>
<td>ESS_PPVOID_T</td>
<td>Returned pointer to the required field element.</td>
</tr>
</tbody>
</table>

Notes
● EssOtlGetMemberField() takes a member handle and field identifier, and returns a pointer to data for the specified field.
If you specify for `MbrFieldID` a constant that was not in the `fieldSelection` string in EssOtlQueryMembersEx(), EssOtlGetMemberField() returns the error OTLAPI_ERR_MBRINVALID.

The caller of EssOtlGetMemberField() should call EssFree() to free the memory set aside for the specified field data.

The following member field identifier constants are valid values for `MbrFieldID`:

- ESS_OTLQRYMBR_NONE
- ESS_OTLQRYMBR_NAME
- ESS_OTLQRYMBR_LEVEL
- ESS_OTLQRYMBR_GENERATION
- ESS_OTLQRYMBR_CONSOLIDATION
- ESS_OTLQRYMBR_TWOPASS
- ESS_OTLQRYMBR_EXPENSE
- ESS_OTLQRYMBR_CURRENCYCONVTYPE
- ESS_OTLQRYMBR_CURRENCYCONVNAME
- ESS_OTLQRYMBR_TIMEBALANCE
- ESS_OTLQRYMBR_SKIP
- ESS_OTLQRYMBR_SHARE
- ESS_OTLQRYMBR_STORAGE
- ESS_OTLQRYMBR_CATEGORY
- ESS_OTLQRYMBR_STORAGECATEGORY
- ESS_OTLQRYMBR_COMMENT
- ESS_OTLQRYMBR_CHILDCOUNT
- ESS_OTLQRYMBR_NUMBER
- ESS_OTLQRYMBR_DIMNAME
- ESS_OTLQRYMBR_DIMNUMBER
- ESS_OTLQRYMBR_ALIASNAME
- ESS_OTLQRYMBR_NEXTNNAME
- ESS_OTLQRYMBR_PREVNAME
- ESS_OTLQRYMBR_PARENTNAME
- ESS_OTLQRYMBR_CHILDNAME
- ESS_OTLQRYMBR_UDA
- ESS_OTLQRYMBR_FORMULA
- ESS_OTLQRYMBR_LASTFORMULA
- ESS_OTLQRYMBR_EXTCOMMENT
- ESS_OTLQRYMBR_ALIASCOMBO
- ESS_OTLQRYMBR_VALID
- ESS_OTLQRYMBR_CURRENCYCONVDB
- ESS_OTLQRYMBR_STATUS
- ESS_OTLQRYMBR_ATTRIBUTED
  True—If attributes are associated
- ESS_OTLQRYMBR_ASSOCATTRDIMNAME
  Associated Attribute Dimension name
- ESS_OTLQRYMBR_ASSOCATTRMEMNAME
  Associated Attribute Member name
- ESS_OTLQRYMBR_ASSOCATTRVALUE
  Associated Attribute value
- ESS_OTLQRYMBR_ATTRVALUE
  Attribute value of the member
- ESS_OTLQRYMBR_UNIQUENAME
  Unique Name of the member
- ESS_OTLQRYMBR_FORMATSTRING
  Format String of the member
- ESS_OTLQRYTIDIM_TIMEPERIODS
  Query Time dimension for time periods list
- ESS_OTLQRYMBR_MBRINFO

**Return Value**

The return value is zero if the function call was successful.

**Example**

See “Extended Member Query Code Example” on page 982 for an example that uses
EssOtlQueryMembersEx(), EssOtlGetMemberField(), and ESS_OTLQUERYERRORLIST_T,
and includes calls to EssOtlFreeMembers() and EssFree().

**See Also**

- EssFree
- EssOtlGetDimensionUserAttributes
- EssOtlOpenOutlineQuery
- EssOtlQueryMembers
- EssOtlQueryMembersByName
- EssOtlQueryMembersEx
**EssOtlGetMemberFormula**

Gets the formula for the specified member.

**Syntax**

```
ESS_FUNC_M EssOtlGetMemberFormula (hOutline, hMember, pszFormula);
```

**Parameter**

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T Outline context handle</td>
</tr>
<tr>
<td>hMember</td>
<td>ESS_HMEMBER_T Member handle.</td>
</tr>
<tr>
<td>ppszFormula</td>
<td>ESS_PSTR_T Variable for the return of the member formula. This buffer is allocated by the API.</td>
</tr>
</tbody>
</table>

**Notes**

- Use `EssFree()` to free the formula buffer.

**Return Value**

Returns 0 if successful.

**Example**

```c
#include <essapi.h>
#include <essotl.h>

ESS_STS_T          sts = 0;
ESS_OBJDEF_T       Object;
ESS_HOUTLINE_T     hOutline;
ESS_HMEMBER_T      hMember;
ESS_STR_T          pszFormula = ESS_NULL;
ESS_APPNAME_T      szAppName;
ESS_DBNAME_T       szDbName;
ESS_OBJNAME_T      szFileName;

memset(&Object, '\0', sizeof(Object));
Object.hCtx = hCtx;
Object.ObjType = ESS_OBJTYPE_OUTLINE;
strcpy(szAppName, "Sample");
strcpy(szDbName, "Basic");
strcpy(szFileName, "Basic");
Object.AppName = szAppName;
Object.DbName = szDbName;
Object.FileName = szFileName;

sts = EssOtlOpenOutline(hCtx, &Object, ESS_TRUE, ESS_TRUE, &hOutline);

if (!sts)
{
    sts = EssOtlFindMember(hOutline, "Variance",
                           &hMember);
}

if (!sts && hMember)
```

826  C Outline API Functions
sts = EssOtlGetMemberFormula(hOutline, hMember, &pszFormula);

if (pszFormula)
{
    EssFree(hInst, pszFormula);
}

See Also

- EssFree
- EssOtlDeleteMemberFormula
- EssOtlGetMemberLastFormula
- EssOtlOpenOutline
- EssOtlOpenOutlineQuery
- EssOtlSetMemberFormula

EssOtlGetMemberInfo

Gets member information for the specified member.

Syntax

```c
ESS_FUNC_M EssOtlGetMemberInfo (hOutline, hMember, pInfo);
```

Parameter | Data Type | Description
---|---|---
hOutline | ESS_HOUTLINE_T | Outline context handle.
hMember | ESS_HMEMBER_T | Member handle.
pInfo | “ESS_MBRINFO_T” on page 699 | Pointer to a member information structure, allocated by the API.

Notes

- Call EssOtlFindMember() to retrieve the member handle.
- Call EssFreeStructure() to free the information structure.
- Two fields of the “ESS_MBRINFO_T” on page 699 structure are for attributes only:
  - fAttributed
  - Attribute

Return Value

Returns 0 if successful.

Example

```c
#include <essapi.h>
#include <essotl.h>

ESS_STS_T          sts = 0;
ESS_OBJDEF_T       Object;
ESS_HOUTLINE_T     hOutline;
```
ESS_HMEMBER hMemberJan;
ESS_PMBRINFO_T pMbrInfo;
ESS_APPNAME_T szAppName;
ESS_DBNAME_T szDbName;
ESS_OBJNAME_T szFileName;
memset(&Object, '\0', sizeof(Object));
Object.hCtx = hCtx;
Object.ObjType = ESS_OBJTYPE_OUTLINE;
strcpy(szAppName, "Sample");
strcpy(szDbName, "Basic");
strcpy(szFileName, "Basic");
Object.AppName = szAppName;
Object.DbName = szDbName;
Object.FileName = szFileName;

sts = EssOtlOpenOutline(hCtx, &Object, ESS_TRUE,
                           ESS_TRUE, &hOutline);

if (!sts)
{
    sts = EssOtlFindMember(hOutline, "Jan",
                           &hMemberJan);
}

if (!sts && hMemberJan)
{
    sts = EssOtlGetMemberInfo(hOutline,
                               hMemberJan, &pMbrInfo);
}

if (pMbrInfo)
{
    EssOtlFreeStructure(hOutline, ESS_DT_STRUCT_MBRINFO, 1, pMbrInfo);
}

---

**EssOtlGetMemberInfoArray**

Gets member information for the specified member array.

**Syntax**

```c
ESS_FUNC_M EssOtlGetMemberInfoArray (hOutline, memberCount, hMemberArr, pInfoArr, pStsArr);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline handle.</td>
</tr>
<tr>
<td>memberCount</td>
<td>ESS_SHORT_T</td>
<td>Count of members in the input array.</td>
</tr>
<tr>
<td>hMemberArr</td>
<td>ESS_HMEMBER_T</td>
<td>Array of memberCount member handles.</td>
</tr>
<tr>
<td>pInfoArr</td>
<td>ESS_PPMBRINFO_T</td>
<td>Array of memberCount member information pointers.</td>
</tr>
</tbody>
</table>

---

828  C Outline API Functions
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pStsArr</td>
<td>ESS_STS_T</td>
<td>Array of memberCount status return codes. If any errors occur, the function returns the value of the first error encountered.</td>
</tr>
</tbody>
</table>

**Notes**

- Call `EssOtlFindMember()` to retrieve the member handles.
- Call `EssFreeStructure()` to free the information structure.
- Two fields of the “ESS_MBRINFO_T” on page 699 structure are for attributes only:
  - fAttributed
  - Attribute

**Return Value**

Returns 0 if successful. If unsuccessful, the pStsArr has the return code for each member.

**Example**

```c
#include <essapi.h>
#include <essotl.h>

ESS_OBJDEF_T      Object;
ESS_HOUTLINE_T    hOutline;
ESS_HMEMBER       hMemberArr[3];
ESS_PMBRINFO_T    pMbrInfoArr[3];
ESS_STS_T         stsArr[3];
ESS_APPNAME_T     szAppName;
ESS_DBNAME_T      szDbName;
ESS_OBJNAME_T     szFileName;
ESS_SHORT_T       i;

memset(&Object, '\0', sizeof(Object));
Object.hCtx = hCtx;
Object.ObjType = ESS_OBJTYPE_OUTLINE;
strcpy(szAppName, "Sample");
strcpy(szDbName, "Basic");
strcpy(szFileName, "Basic");
Object.AppName = szAppName;
Object.DbName = szDbName;
Object.FileName = szFileName;
sts = EssOtlOpenOutline(hCtx, &Object, ESS_TRUE, ESS_TRUE, &hOutline);
if (!sts)
{
    sts = EssOtlFindMember(hOutline, "Jan", &hMemberArr[0]);
}
if (!sts)
{
    sts = EssOtlFindMember(hOutline, "Feb", &hMemberArr[1]);
}
if (!sts)
{
    sts = EssOtlFindMember(hOutline, "Mar", &hMemberArr[2]);
}
```
if (!sts)
{
    sts = EssOtlGetMemberInfoArray(hOutline, 3, hMemberArr, pMbrInfoArr, stsArr);
}
for (i = 0; i < 3; i++)
{
    if (pMbrInfoArr[i])
    {
        EssOtlFreeStructure(hOutline, ESS_DT_STRUCT_MBRINFO, 1, pMbrInfoArr[i]);
    }
}

---

**EssOtlGetMemberLastFormula**

Returns the last formula used to calculate the member.

**Syntax**

ESS_FUNC_M EssOtlGetMemberLastFormula (hOutline, hMember, ppszFormula);

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline context handle</td>
</tr>
<tr>
<td>hMember</td>
<td>ESS_HMEMBER_T</td>
<td>Member handle.</td>
</tr>
<tr>
<td>ppszFormula</td>
<td>ESS_PSTR_T</td>
<td>Variable for the return of the member formula. This buffer is allocated by the API.</td>
</tr>
</tbody>
</table>

**Notes**

- Use EssFree() to free the formula buffer.
- This call will work for both EssOtlOpenOutline() and EssOtlOpenOutlineQuery().
- EssOtlGetMemberLastFormula() returns the last formula applied to the selected member, which might differ from the Database Outline formula associated with that member.
- The last formula is derived from the last calculation (either from the outline or calc scripts) done on the member.

**Return Value**

The return value is zero if the function was successful.

**Example**

```c
#include <ESSAPI.H>
#include <ESSOTL.H>

ESS_STS_T          sts = 0 ;
ESS_HOUTLINE_T     hOutline;
ESS_OBJDEF_T       Object;
ESS_APPNAME_T      szAppName;
ESS_DBNAME_T       szDbName;
ESS_OBJNAME_T      szFileName;
ESS_HMEMBER_T      hMember;
ESS_STR_T          pszFormula = ESS_NULL;
```
memset(&Object, '\0', sizeof(Object));
Object.hCtx = hCtx;
Object.ObjType = ESS_OBJTYPE_OUTLINE;
strcpy(szAppName, "Sample");
strcpy(szDbName, "Basic");
strcpy(szFileName, "Basic");
Object.AppName = szAppName;
Object.DbName = szDbName;
Object.FileName = szFileName;

sts = EssOtlOpenOutline(hCtx, &Object, ESS_TRUE,
    ESS_TRUE, &hOutline);

if (!sts)
    {
    sts = EssOtlFindMember(hOutline, "Margin", &hMember);
    }

if (!sts && hMember)
    {
    sts = EssOtlGetMemberLastFormula(hOutline, hMember, &pszFormula);
    printf("Member Last Formula: %s\n", pszFormula);
    }

if (pszFormula)
    {
    EssFree(hInst, pszFormula);
    }

See Also

- EssFree
- EssOtlDeleteMemberFormula
- EssOtlGetMemberFormula
- EssOtlOpenOutline
- EssOtlOpenOutlineQuery
- EssOtlSetMemberFormula

**EssOtlGetMemberSmartList**

Returns the Text List (SmartList) associated with the input outline member.

**Syntax**

```c
ESS_FUNC_M EssOtlGetMemberSmartList(hOutline, hMember, *phSmartlist);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline handle (Edit mode only)</td>
</tr>
<tr>
<td>hMember</td>
<td>ESS_HMEMBER_T</td>
<td>Outline member handle</td>
</tr>
<tr>
<td>*phSmartlist</td>
<td>ESS_HSMARTLIST_T</td>
<td>Returns the associate Text List (SmartList) handle</td>
</tr>
</tbody>
</table>
**Return Value**

Returns:

- 0—If successful
  
  *phSmartlist* contains return value.

- Error number—If unsuccessful
  
  *phSmartlist* is NULL.

**Example**

```c
void TestGetMemberSmartList()
{
    ESS_STS_T                   sts = ESS_STS_NOERR;
    ESS_HOUTLINE_T            hOutline = ESS_NULL;
    ESS_OBJDEF_T                Object;
    ESS_HMEMBER_T            hMember;
    ESS_HSMARTLIST_T        hSmartList;

    memset(&Object, '\0', sizeof(Object));
    Object.hCtx =             hCtx;
    Object.ObjType =         ESS_OBJTYPE_OUTLINE;
    Object.AppName =         szAppName;
    Object.DbName =         szDbName;
    Object.FileName =     szFileName;

    /* Open outline */
    sts = EssOtlOpenOutline(hCtx, &Object,
                            ESS_TRUE, ESS_TRUE, &hOutline);

    /* Find member */
    sts = EssOtlFindMember(hOutline, "Original Price",
                          &hMember);

    /* Return SmartList associated with member */
    sts = EssOtlGetMemberSmartList(hOutline, hMember,
                                     &hSmartList);

    /* Unlock object */
    sts = EssUnlockObject(hCtx, Object.ObjType,
                           Object.AppName, Object.DbName, Object.FileName);

    /* Close outline */
    sts = EssOtlCloseOutline(hOutline);
}
```

**See Also**

- EssOtlGetMemberSmartList
- EssOtlCreateObject
- EssOtlDeleteObject
- EssOtlGetSmartListInfo
- EssOtlFindObject
- EssOtlFreeObjectArray
EssOtlGetMemberSolveOrder

Returns the solve order of a member.

**Syntax**

ESS_FUNC_M EssOtlGetMemberSolveOrder (hOutline, hMember, pOrder);

**Parameter**  **Data Type**  **Description**

| hOutline       | ESS_HOUTLINE_T | Outline context handle (input). |
| hMember        | ESS_HMEMBER_T  | Member handle (input).          |
| pOrder         | ESS_PUCHAR_T   | Solve order (output).           |

**Notes**

- Solve order is applicable only to aggregate storage databases.
- Solve order property on a member or dimension specifies its calculation order.
- Member solve order takes precedence over dimension solve order. Solve order can be between 0 and 127. The default is 0.
- Members without formulas that do not have a specified solve order inherit the solve order of their dimension. Members with formulas that do not have a specified solve order have a solve order of zero.

**Return Value**

Returns 0 if successful.

**Example**

```c
ESS_STS_T         sts = ESS_STS_NOERR;
ESS_HOUTLINE_T    hOutline = ESS_NULL;
ESS_HMEMBER_T     hMember = ESS_NULL;
ESS_UCHAR_T       ucOrder = 0;

/* code to assign hOutline variable omitted */
/* code to assign hMember variable omitted */
```
if (hOutline && hMember)
{
    sts = EssOtlGetMemberSolveOrder(hOutline, hMember, &ucOrder);
    if (sts)
        printf("Error [%ld] returned\n", sts);
    else
        printf("Solve Order: %d\n", ucOrder);
} else
    printf("Both hOutline and hMember must have values\n");

EssOtlSetMemberSolveOrder
EssOtlSetDimensionSolveOrder
EssOtlGetDimensionSolveOrder

**EssOtlGetMemberType**

Returns the member type of the input outline member.

**Syntax**

```c
ESS_FUNC_M EssOtlGetMemberType(hOutline, hMember, *pusType)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline handle</td>
</tr>
<tr>
<td>hMember</td>
<td>ESS_HMEMBER_T</td>
<td>Outline member handle</td>
</tr>
<tr>
<td>*pusType</td>
<td>ESS_USHORT_T</td>
<td>Type of the outline member:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_MEMBERTYPE_NUMERIC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Member type is a numeric.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_MEMBERTYPE_SMARTLIST</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Member type is textual and has an associated Text List (SmartList) object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_MEMBERTYPE_DATE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Member type is date typed.</td>
</tr>
</tbody>
</table>

**Return Value**

Returns:

- 0—If successful
   
   `pusType` contains a value.
- Error number—If unsuccessful
   
   `pusType` is NULL.

**Example**

```c
void TestGetSetMemberType()
{
```
```c
ESS_STS_T          sts = ESS_STS_NOERR;
ESS_HOUTLINE_T     hOutline = ESS_NULL;
ESS_OBJDEF_T       Object;
ESS_HMEMBER_T      hMember;
ESS_USHORT_T       usMemberType;
memset(&Object, '\0', sizeof(Object));
Object.hCtx =      hCtx;
Object.ObjType =   ESS_OBJTYPE_OUTLINE;
Object.AppName =   szAppName;
Object.DbName =    szDbName;
Object.FileName =  szFileName;

/* Open outline */
sts = EssOtlOpenOutline(hCtx, &Object, ESS_TRUE,
                        ESS_TRUE, &hOutline);

/* Find a member */
sts = EssOtlFindMember(hOutline, "Original Price", &hMember);

/* Get Member Type of an outline that is not member
type enabled */
/* Get original type */
sts = EssOtlGetMemberType(hOutline, hMember, &usMemberType);
DisplayMemberType(usMemberType); /* a display function */

/* Get Member Type of an outline that is member
type enabled */
EnableSmartList(hOutline);

/* Get original type */
sts = EssOtlGetMemberType(hOutline, hMember, &usMemberType);
printf("EssOtlGetMemberType sts: %d\n", sts);
DisplayMemberType(usMemberType);

/* Set type to NUMERIC */
usMemberType = ESS_MEMBERTYPE_NUMERIC;
sts = EssOtlSetMemberType(hOutline, hMember, usMemberType);
printf("EssOtlSetMemberType sts: %d\n",sts);

sts = EssOtlGetMemberType(hOutline, hMember, &usMemberType);
printf("EssOtlGetMemberType sts: %d\n", sts);
DisplayMemberType(usMemberType);

/* Clean up */
sts = EssUnlockObject(hCtx, Object.ObjType,
                        Object.AppName, Object.DbName, Object.FileName);

/* Close outline */
sts = EssOtlCloseOutline(hOutline);
}

See Also
- EssOtlGetMemberSmartList
- EssOtlCreateObject
```
EssOtlDeleteObject
EssOtlGetSmartListInfo
EssOtlFindObject
EssOtlFreeObjectArray
EssOtlFreeSmartListInfo
EssOtlGetMemberSmartList
EssOtlGetMemberType
EssOtlGetObjectReferenceCount
EssOtlGetObjectReferences
EssOtlImportExportObject
EssOtlListObjects
EssOtlQueryObjects
EssOtlSetMemberType
EssOtlSetMemberTypeToSmartList

EssOtlGetMemberUniqueName

Returns the member name (if the member name is unique) or the minimum qualified name required to distinguish the member (if the member name is duplicate).

Syntax

```c
ESS_FUNC_M EssOtlGetMemberUniqueName (hOutline, hMember, *szFullName);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline context handle (input).</td>
</tr>
<tr>
<td>hMember</td>
<td>ESS_HMEMBER_T</td>
<td>Member handle (input).</td>
</tr>
<tr>
<td>*szFullName</td>
<td>ESS_STR_T</td>
<td>The returned member name or qualified member name (output).</td>
</tr>
</tbody>
</table>

Notes

- Before you call this function, call `EssOtlOpenOutline` to open the outline in editing mode, or call `EssOtlOpenOutlineQuery` to open the outline in query mode.
- Use a Member Traversal Function to get a member handle for the second argument of this function.
- In an outline that allows duplicate member names, if the member handle passed in is an extended shared member or a regular shared member, this function returns the unique name.

Return Value

Returns 0 if successful; otherwise, returns an error.
Example

Example 1

The output of this function in the following example is the fully qualified member name of Qtr1:

\[2004\].[Qtr1]\n
ESS_FUNC_M ESS_GetMemberUniq()
{
    ESS_STS_T     sts = 0;
    ESS_HOUTLINE_T hOutline;
    ESS_OBJDEF_T   Object;
    ESS_APPNAME_T  szAppName;
    ESS_DBNAME_T   szDbName;
    ESS_OBJNAME_T  szFileName;
    ESS_STR_T      szFullName;
    ESS_HMEMBER_T  hMemberParent;
    ESS_HMEMBER_T  hMemberChild;

    memset(&Object, '\0', sizeof(Object));
    Object.hCtx =   hCtx;
    Object.ObjType =  ESS_OBJTYPE_OUTLINE;
    strcpy(szAppName, "Demo");
    strcpy(szDbName, "Test");
    strcpy(szFileName, "Test");
    Object.AppName =  szAppName;
    Object.DbName =   szDbName;
    Object.FileName =  szFileName;
    sts = EssOtlOpenOutline(hCtx, &Object, ESS_TRUE, 
                             ESS_TRUE, &hOutline);

    if (!sts)
    {
        sts = EssOtlFindMember(hOutline, "2004", &hMemberParent);
    }

    if (!sts && hMemberParent)
    {
        sts = EssOtlGetChild(hOutline, hMemberParent, &hMemberChild);
    }

    /*Get the qualified name of the first child of 2004, Qtr1*/
    if (!sts)
    {
        sts = EssOtlGetMemberUniqueName (hOutline, hMemberChild, &szFullName);

        if (sts)
            printf("EssOtlGetMemberUniqueName failed sts %ld\n",sts);
        else
            printf("Qtr1's qualified name is: %s\n", szFullName);
    }
Example 2

The following example shows this function used in query mode.

```c
member_fields     = "<SelectMbrInfo (membername, uniquename) ";
member_selection  = "@SHARE(@DESCENDANTS(product))";
MaxCount          = -1;
phMemberArray     = ESS_NULL;
pqryErrorList     = ESS_NULL;

status = EssOtlQueryMembersEx(hOutline, 
member_fields, 
member_selection, 
&MaxCount, 
&phMemberArray, 
&pqryErrorList);

if (status) goto exit;

for (int i = 0; i < MaxCount; i++)
{
    status = EssOtlGetMemberField(hOutline, phMemberArray[i], ESS_OTLQRYMBR_NAME, 
       (ESS_PPVOID_T) &pName);
    if (status) goto exit;

    status = EssOtlGetMemberUniqueName(hOutline, phMemberArray[i], &pUniqueName2);
    if (status) goto exit;
}
```

**EssOtlGetNextSharedMember**

Returns the member handle to the next shared member of the specified member.

**Syntax**

```
ESS_FUNC_M EssOtlGetNextSharedMember (hOutline, hMember, phMember);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline context handle.</td>
</tr>
<tr>
<td>hMember</td>
<td>ESS_HMEMBER_T</td>
<td>Handle of member to find the next shared member for.</td>
</tr>
<tr>
<td>phMember</td>
<td>ESS_PHMEMBER_T</td>
<td>Pointer for return of a member handle of the next shared member in the outline. This parameter is ESS_NULL if there are no more shared members.</td>
</tr>
</tbody>
</table>

**Notes**

- If *hmember* is the actual member, the first shared member is returned in the *phMember* parameter. If *hmember* is a shared member, the next shared member is returned in the *phMember* parameter.
● If there are no (more) shared members, `phMember` is set to `ESS_NULL` and the call returns
0.

**Return Value**

Returns 0 if successful.

**Example**

```c
#include <essapi.h>
#include <essotl.h>

ESS_STS_T      sts = 0;
ESS_OBJDEF_T   Object;
ESS_HOUTLINE_T hOutline;
ESS_HMEMBER_T  hMember;
ESS_HMEMBER_T  hMemberShared;
ESS_HMEMBER_T  hNextShared;
ESS_APPNAME_T  szAppName;
ESS_DBNAME_T   szDbName;
ESS_OBJNAME_T  szFileName;

memset(&Object, '\0', sizeof(Object));
Object.hCtx = hCtx;
Object.ObjType = ESS_OBJTYPE_OUTLINE;
strcpy(szAppName, "Sample");
strcpy(szDbName, "Basic");
strcpy(szFileName, "Basic");
Object.AppName = szAppName;
Object.DbName = szDbName;
Object.FileName = szFileName;

sts = EssOtlOpenOutline(hCtx, &Object, ESS_TRUE,
                        ESS_TRUE, &hOutline);

if (!sts)
{
    sts = EssOtlFindMember(hOutline, "200-20",
                           &hMember);
}

if (!sts  && hMember)
{
    /* get first shared member of actual member */
    sts = EssOtlGetNextSharedMember(hOutline, hMember, &hMemberShared);

    /* do something with hMemberShared */
    /* get next shared member, if any*/

    while(!sts  && hMemberShared)
    {
        sts = EssOtlGetNextSharedMember(hOutline,
                                         hMemberShared, &hNextShared);
        hMemberShared = hNextShared;

        /* do something with hMemberShared */
    }
}
```
See Also

- *EssOtlFindMember*

**EssOtlGetNextSibling**

Returns the next sibling of a member.

**Syntax**

```c
ESS_FUNC_M EssOtlGetNextSibling (hOutline, hMember, phMember);
```

**Parameter** | **Data Type** | **Description**
--- | --- | ---
`hOutline` | ESS_HOUTLINE_T | Outline context handle
`hMember` | ESS_HMEMBER_T | Handle of member whose sibling you want to retrieve
`phMember` | ESS_PHMEMBER_T | Pointer for return of a member handle of the sibling of the `hMember` parameter

**Notes**

- If there is no next sibling, `*phMember` is set to ESS_NULL and the call returns 0.

**Return Value**

Returns 0 if successful.

**Example**

```c
#include <essapi.h>
#include <essotl.h>

ESS_STS_T          sts = 0;
ESS_HOUTLINE_T     hOutline;
ESS_HMEMBER_T      hMemberJan;
ESS_HMEMBER_T      hMemberSibling;
ESS_OBJDEF_T       Object;
ESS_APPNAME_T      szAppName;
ESS_DBNAME_T       szDbName;
ESS_OBJNAME_T      szFileName;
memset(&Object, '\0', sizeof(Object));
Object.hCtx = hCtx;
Object.ObjType = ESS_OBJTYPE_OUTLINE;
strcpy(szAppName, "Sample");
strcpy(szDbName, "Basic");
strcpy(szFileName, "Basic");
Object.AppName = szAppName;
Object.DbName = szDbName;
Object.FileName = szFileName;

sts = EssOtlOpenOutline(hCtx, &Object, ESS_TRUE,
                        ESS_TRUE, &hOutline);

if (!sts) {
```
sts = EssOtlFindMember(hOutline, "Jan", &hMemberJan);
}

if (!sts && hMemberJan)
{
    sts = EssOtlGetNextSibling(hOutline, hMemberJan, &hMemberSibling);
}

See Also

- EssOtlGetPrevSibling
- EssOtlGetParent
- EssOtlGetChild
- EssOtlGetFirstMember

**EssOtlGetNumQueryHints**

Returns the hint numbers of all query hints in the outline.

**Syntax**

```c
ESS_FUNC_M EssOtlGetNumQueryHints (hOutline, pNumHints);
```

**Parameter** | **Data Type** | **Description**
--- | --- | ---
| hOutline | ESS_HOUTLINE_T | Outline context handle (input). |
| pNumHints | ESS_PSHORT_T | Pointer to an array of query hint numbers (output). |

**Notes**

- Query hints enable you to influence normal view selection by informing Essbase about the profile of common queries.
- This function is applicable only to Release 9.3 or higher aggregate storage databases.

**Return Value**

Returns 0 if successful.

**Example**

See **EssOtlSetQueryHint**.

**See Also**

- EssOtlAddQueryHint
- EssOtlSetQueryHint
- EssOtlGetQueryHint
- EssOtlGetQueryHintSize
- EssOtlDeleteQueryHint
EssOtlGetObjectReferenceCount

Returns the count of outline members referencing the input object handle.

Syntax

```c
ESS_FUNC_M EssOtlGetObjectReferenceCount(hOutline, objHandle, pCount)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline handle (Edit mode only)</td>
</tr>
<tr>
<td>objHandle</td>
<td>ESS_HOBJECT_T</td>
<td>Object handle to be imported or exported</td>
</tr>
<tr>
<td>pCount</td>
<td>ESS_ULONG_T*</td>
<td>Count of outline members</td>
</tr>
</tbody>
</table>

Return Value

Returns:

- 0—If successful
   - `pCount` contains values.
- Error number—If unsuccessful
   - `pCount` is NULL.

Example

```c
void TestGetObjectReferenceCount()
{
    ESS_STS_T        sts = ESS_STS_NOERR;
    ESS_HOUTLINE_T        hOutline = ESS_NULL;
    ESS_OBJDEF_T        Object;
    ESS_HOBJECT_T        hObjHandle = ESS_NULL;
    ESS_ULONG_T            Count = 0;
    ESS_OBJECT_TYPES    objType;
    ESS_STR_T            objName;
    memset(&Object, '\0', sizeof(Object));
    Object.hCtx = hCtx;
    Object.ObjType = ESS_OBJTYPE_OUTLINE;
    Object.AppName = szAppName;
    Object.DbName = szDbName;
    Object.FileName = szFileName;
    sta = EssOtlOpenOutline(hCtx, &Object,
                ESS_TRUE, ESS_TRUE, &hOutline);

    /* Get count of an object that is referenced */
    objType = OBJECT_SMARTLIST;
    objName = "Smartlist1";
    sta = EssOtlFindObject(hOutline, objType,
                objName, &hObjHandle);
    printf("EssOtlFindObject sta: %ld\n", sta);
    sta = EssOtlGetObjectReferenceCount(hOutline,
                hObjHandle, &Count);
}
```
printf("EssOtlGetObjectReferenceCount sts: %ld\n",sts);
printf("\tCount returned: %d\n", Count);

sts = EssUnlockObject(hCtx, Object.ObjType,
Object.AppName, Object.DbName, Object.FileName);
sts = EssOtlCloseOutline(hOutline);

See Also
● EssOtlGetMemberSmartList
● EssOtlCreateObject
● EssOtlDeleteObject
● EssOtlGetSmartListInfo
● EssOtlFindObject
● EssOtlFreeObjectArray
● EssOtlFreeSmartListInfo
● EssOtlGetMemberSmartList
● EssOtlGetMemberType
● EssOtlGetObjectReferenceCount
● EssOtlGetObjectReferences
● EssOtlImportExportObject
● EssOtlListObjects
● EssOtlQueryObjects
● EssOtlSetMemberType
● EssOtlSetMemberTypeToSmartList

EssOtlGetObjectReferences

Returns an array of outline members referencing the input object handle. This function should be followed by deallocating phMembers using EssFree.

Syntax

ESS_FUNC_M EssOtlGetObjectReferences(hOutline, objHandle, ulMaxCount, phMembers, pulNumMembers)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline handle (Edit mode only)</td>
</tr>
<tr>
<td>objHandle</td>
<td>ESS_HOBJECT_T</td>
<td>Object handle</td>
</tr>
<tr>
<td>ulMaxCount</td>
<td>ESS_ULONG_T</td>
<td>Count of max outline members that the client can handle</td>
</tr>
<tr>
<td>phMembers</td>
<td>ESS_HMEMBER_T*</td>
<td>Output array of outline members</td>
</tr>
<tr>
<td>pulNumMembers</td>
<td>ESS_ULONG_T*</td>
<td>Number of outline members returned</td>
</tr>
</tbody>
</table>

Return Value

Returns:
0—If successful

ulMaxCount, phMembers, and pulNumMembers contain values.

Error number—If unsuccessful

ulMaxCount, phMembers, and pulNumMembers are NULL.

Example

```c
void TestGetObjectReferences()
{
    ESS_STS_T                    sts = ESS_STS_NOERR;
    ESS_HOUTLINE_T            hOutline = ESS_NULL;
    ESS_OBJDEF_T                Object;
    ESS_HOBJECT_T            hObjHandle = ESS_NULL;
    ESS_ULONG_T                ulMaxCount;
    ESS_HMEMBER_T            hMembers[256];
    ESS_ULONG_T                ulNumMembers, i;
    ESS_OBJECT_TYPES        objType;
    ESS_STR_T                    objName;
    ESS_PMBRINFO_T         pMbrInfo;

    memset(&Object, '\0', sizeof(Object));
    Object.hCtx =             hCtx;
    Object.ObjType =         ESS_OBJTYPE_OUTLINE;
    Object.AppName =         szAppName;
    Object.DbName =         szDbName;
    Object.FileName =         szFileName;
    sts = EssOtlOpenOutline(hCtx, &Object,
                              ESS_TRUE, ESS_TRUE, &hOutline);

    /* Get the member(s) of the object that is referenced */
    objType = OBJECT_SMARTLIST;
    objName = "SmartList1";
    sts = EssOtlFindObject(hOutline, objType, objName, &hObjHandle);

    ulMaxCount = 256;
    sts = EssOtlGetObjectReferences(hOutline, hObjHandle,
                                     ulMaxCount, hMembers, &ulNumMembers);
    printf("EssOtlGetObjectReferences sts: %ld\n", sts);

    for(i = 0; i < ulNumMembers; i++)
    {
        sts = EssOtlGetMemberInfo(hOutline, hMembers[i], &pMbrInfo);
        if(pMbrInfo)
            printf("\tMember: %s\n", pMbrInfo->szMember);
    }

    sts = EssUnlockObject(hCtx, Object.ObjType,
                          Object.AppName, Object.DbName, Object.FileName);
    sts = EssOtlCloseOutline(hOutline);
}
```

See Also

- EssOtlGetMemberSmartList
- EssOtlCreateObject
EssOtlGetOriginalMember

Returns the original member name of a shared or extended shared member. If the member is not shared, the return value is NULL. This function returns the fully qualified original member name.

**Syntax**

```c
ESS_FUNC_M EssOtlGetOriginalMember (hOutline, hMember, ppOriMember);
```

**Parameter** | **Data Type** | **Description**
--- | --- | ---
`hOutline` | ESS_HOUTLINE_T | Outline context handle (input).
`hMember` | ESS_HMEMBER_T | Member name (input).
`ppOriMember` | ESS_PSTR_T | The original member name (output).

**Notes**

- This function works in both edit and query modes.
- If you use this function on an outline in which all member names are unique, it will have no effect.
- In an outline that allows duplicate member names, if the member handle passed in is an extended shared member or a regular shared member, this function returns its original member as a path expression.
- Given the following hierarchy, if you pass to this function the member handle corresponding to `[Diet].[100-10]`, it returns `[200].[100-10].`.

```
100
100-10
200
100-10 (duplicate)
Diet
100-10 (shared with [200.100-10])
```
Return Value

Returns 0 if successful; otherwise, returns an error.

Example

The "original member" returned for the Sample Basic shared member 100-10 is [100].
[100-20].

ESS_FUNC_M ESS_GetOrigMember()
{
    ESS_STS_T    sts = 0;
    ESS_HOUTLINE_T   hOutline;
    ESS_OBJDEF_T   Object;
    ESS_APPNAME_T   szAppName;
    ESS_DBNAME_T   szDbName;
    ESS_OBJNAME_T   szFileName;
    ESS_HMEMBER_T   hMember = ESS_NULL, ChildMember = ESS_NULL;
    ESS_STR_T    OriMember;

    memset(&Object, '\0', sizeof(Object));
    Object.hCtx =   hCtx;
    Object.ObjType =  ESS_OBJTYPE_OUTLINE;
    strcpy(szAppName, "Sample");
    strcpy(szDbName, "Basic");
    strcpy(szFileName, "Basic");
    Object.AppName =  szAppName;
    Object.DbName =   szDbName;
    Object.FileName =  szFileName;

    sts = EssOtlOpenOutline(hCtx, &Object, ESS_TRUE,
            ESS_TRUE, &hOutline);

    // sts = EssOtlOpenOutlineQuery (hCtx, &Object, &hOutline);

    if (!sts)
    {
        sts = EssOtlFindMember(hOutline, "Diet", &hMember);
    }

    //Get member handle for shared member "100-10"
    if (!sts & hMember)
    {
        sts = EssOtlGetChild(hOutline, hMember, &ChildMember);
    }

    if (!sts & ChildMember)
    {
        sts = EssOtlGetOriginalMember (hOutline, ChildMember, &OriMember);
        printf("Original member for shared member \"100-10\" is: %s", OriMember);
    }
See Also

- EssOtlSetOriginalMember

EssOtlGetOutlineInfo

Returns information about the outline file.

Syntax

```c
ESS_FUNC_M EssOtlGetOutlineInfo (hOutline, ppInfo);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline context handle.</td>
</tr>
<tr>
<td>ppInfo</td>
<td>“ESS_OUTLINEINFO_T” on page 706</td>
<td>Pointer to a pointer to a structure allocated by the API for storing outline information.</td>
</tr>
</tbody>
</table>

Notes

- Use EssFree() to free the information structure.

Return Value

Returns 0 if successful.

Example

```c
#include <essapi.h>
#include <essotl.h>

ESS_STS_T   sts = 0;
ESS_HOUTLINE_T   hOutline;
ESS_POUTLINEINFO_T pInfo = ESS_NULL;
ESS_OBJDEF_T  Object;
ESS_APPNAME_T         szAppName;
ESS_DBNAME_T          szDbName;
ESS_OBJNAME_T         szFileName;
memset(&Object, '\0', sizeof(Object));
Object.hCtx = hCtx;
Object.ObjType = ESS_OBJTYPE_OUTLINE;
strncpy(szAppName, "Sample");
strncpy(szDbName, "Basic");
strncpy(szFileName, "Basic");
Object.AppName = szAppName;
Object.DbName = szDbName;
Object.FileName = szFileName;

sts = EssOtlOpenOutline(hCtx, &Object, ESS_TRUE,
                        ESS_TRUE, &hOutline);

if (!sts)
```

EssOtlGetParent

Returns the parent of a member.

Syntax

\[
\text{ESS\_FUNC\_M EssOtlGetParent (hOutline, hMember, phMember);}
\]

Parameter | Data Type       | Description
----------|----------------|-------------------
hOutline   | ESS\_HOUTLINE\_T | Outline context handle.
hMember    | ESS\_HMEMBER\_T | Handle of member to retrieve the parent of.
phMember   | ESS\_PHMEMBER\_T | Pointer for return of a member handle of the parent of the \textit{hMember} parameter.

Notes

- If there is no parent, \textit{phMember} is set to ESS\_NULL and the call returns 0. (The \textit{hMember} is a dimension.)

Return Value

Returns 0 if successful.

Example

```c
#include <essapi.h>
#include <essotl.h>

ESS\_STS\_T st = 0;
ESS\_HOUTLINE\_T hOutline;
ESS\_HMEMBER\_T hMemberChild;
ESS\_HMEMBER\_T hMemberParent;
ESS\_OBJDEF\_T Object;
ESS\_APPNAME\_T szAppName;
ESS\_DBNAME\_T szDbName;
ESS\_OBJNAME\_T szFileName;

memset(&Object, '\0', sizeof(Object));
Object.hCtx = hCtx;
Object.ObjType = ESS\_OBJTYPE\_OUTLINE;
strcpy(szAppName, "Sample");
```
strcpy(szDbName, "Basic");
strcpy(szFileName, "Basic");
Object.AppName = szAppName;
Object.DbName = szDbName;
Object.FileName = szFileName;

sts = EssOtlOpenOutline(hCtx, &Object, ESS_TRUE,
                         ESS_TRUE, &hOutline);

if (!sts)
{
    sts = EssOtlFindMember(hOutline, "Jan",
                           &hMemberChild);
}

if (!sts  && hMemberChild)
{
    sts = EssOtlGetParent(hOutline,
                           hMemberChild, &hMemberParent);
}

See Also

- EssOtlGetChild
- EssOtlGetNextSibling
- EssOtlGetPrevSibling
- EssOtlGetFirstMember

EssOtlGetPrevSibling

Returns the previous sibling of a member.

Syntax

ESS_FUNC_M EssOtlGetPrevSibling (hOutline, hMember, phMember);

Parameter | Data Type | Description
----------|-----------|------------------
          |           | Outline context handle.
          | ESS_HMEMBER_T | Handle of member to retrieve the previous sibling of.
          | ESS_PHMEMBER_T | Pointer for return of a member handle of the previous sibling of the hMember parameter.

Notes

- If there is no previous sibling, *phMember is set to ESS_NULL and the call returns 0.

Return Value

Returns 0 if successful.

Example

#include <essapi.h>
#include <essotl.h>
memset(&Object, '\0', sizeof(Object));
Object.hCtx = hCtx;
Object.ObjType = ESS_OBJTYPE_OUTLINE;
strcpy(szAppName, "Sample");
strcpy(szDbName, "Basic");
strcpy(szFileName, "Basic");
Object.AppName = szAppName;
Object.DbName = szDbName;
Object.FileName = szFileName;

sts = EssOtlOpenOutline(hCtx, &Object, ESS_TRUE, ESS_TRUE, &hOutline);

if (!sts)
{
    sts = EssOtlFindMember(hOutline, "Feb",
                         &hMemberFeb);
}
if (!sts && hMemberFeb)
{
    sts = EssOtlGetPrevSibling(hOutline,
                                hMemberFeb, &hMemberSibling);
}

See Also
- EssOtlGetNextSibling
- EssOtlGetParent
- EssOtlGetChild
- EssOtlGetFirstMember

**EssOtlGetQueryHint**

Returns the query hint indicated by the input outline and hint number.

Hints are numbered from 1 to \( n \). The first query hint has a hint number of 1. Each new query hint is added to the end of the list, with its number increased by 1.

**Syntax**

```c
ESS_FUNC_M EssOtlGetQueryHint (hOutline, hintNum, numMembers, pMemberArray);
```
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline context handle (input).</td>
</tr>
<tr>
<td>hintNum</td>
<td>ESS_SHORT_T</td>
<td>Query hint number (input).</td>
</tr>
<tr>
<td>numMembers</td>
<td>ESS_SHORT_T</td>
<td>Number of members that the array provided is able to hold - usually the number of real dimensions in the outline (input)</td>
</tr>
<tr>
<td>pMemberArray</td>
<td>ESS_PHMEMBER_T</td>
<td>An array of members for the hint. Usually the array has one member per real dimension, with NULL used for dimensions that are not part of the hint. This array needs to be allocated with size numMembers. (Output)</td>
</tr>
</tbody>
</table>

Notes
- Query hints enable you to influence normal view selection by informing Essbase about the profile of common queries.
- This function is applicable only to Release 9.3 or higher aggregate storage databases.

Return Value
Returns 0 if successful.

Example

```c
ESS_STS_T         sts = ESS_STS_NOERR;
ESS_HOUTLINE_T    hOutline = ESS_NULL;
ESS_SHORT_T       nmHints = 0;
ESS_SHORT_T       i, j, hintNum;
ESS_HMEMBER_T     hMember[10]; /* (nm real dimensions) < 10 */

/* clear array just to be safe */
memset(hMember, 0x00, 10*sizeof(ESS_HMEMBER_T));

/* Code to assign hOutline variable omitted */

sts = EssOtlGetNumQueryHints(hOutline, &nmHints);
if (sts) return sts;  /* error out */
for (i = 0; i < nmHints; i++)
{
    hintNum = i+1;
    sts = EssOtlGetQueryHint(hOutline, hintNum, 10, hMember);
    if (sts) return sts;  /* error out */
    for (j = 0; j < 10; j++)
    {
        if (hMember[j] != AD_NULL)
        {
            sts = EssOtlGetMemberInfo(hOutline, hMember[j], &pMemberInfo);
            if (sts) return sts; /* error out */
            printf("Hint (%d), member (%d): [%s]\n",
            hintNum, j, pMemberInfo->szMember);
        /* Code to free pMemberInfo omitted */
        }
        else
        {
        }
```
printf("Hint (%d), member (%d): [NULL]\n", hintNum, j);
}
}
}

See Also

- EssOtlAddQueryHint
- EssOtlSetQueryHint
- EssOtlGetNumQueryHints
- EssOtlGetQueryHintSize
- EssOtlDeleteQueryHint

**EssOtlGetQueryHintSize**

Returns the size (in number of members) of the query hints defined on the outline.

Hints are numbered from 1 to $n$. The first query hint has a hint number of 1. Each new query hint is added to the end of the list, with its number increased by 1.

**Syntax**

```c
ESS_FUNC_M EssOtlGetQueryHintSize (hOutline, pHintSize);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline context handle (input).</td>
</tr>
<tr>
<td>pHintSize</td>
<td>ESS_SHORT_T</td>
<td>Query hint size (output).</td>
</tr>
</tbody>
</table>

**Notes**

Usually the number of members in a query hint is the same as the number of real dimensions. But if you add or delete dimensions after the hints were added, the number of members in the `hMember` array might be different than the number of real dimensions. This function returns how large the member array should be in GetQueryHint.

**Return Value**

Returns 0 if successful.

**See Also**

- EssOtlAddQueryHint
- EssOtlSetQueryHint
- EssOtlGetQueryHint
- EssOtlGetNumQueryHints
- EssOtlDeleteQueryHint
EssOtlGetSmartListInfo

Returns the Text List (SmartList) information for the Text List (SmartList) passed in the hSmartList handle. This must be followed by an EssOtlFreeSmartListInfo call on ppSmartListInfo.

Syntax

ESS_FUNC_M EssOtlGetSmartListInfo(hOutline, hSmartList, **ppSmartListInfo);

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline handle</td>
</tr>
<tr>
<td>hSmartList</td>
<td>ESS_HSMARTLIST_T</td>
<td>Text List (SmartList) handle</td>
</tr>
<tr>
<td>**ppSmartListInfo</td>
<td>ESS_SMARTLISTINFO_T</td>
<td>Contains the Text List (SmartList) information structure.</td>
</tr>
</tbody>
</table>

Return Value

Returns:

- 0—If successful
  - ppSmartListInfo contains the Text List (SmartList) information.
- Error number—If unsuccessful
  - ppSmartListInfo is NULL.

Example

DisplaySmartListInfo(ESS_HOUTLINE_T hOutline, ESS_PHOBJECT_T ObjHandles)
{
    ESS_STS_T                        sts = ESS_STS_NOERR;
    ESS_PSMARTLISTINFO_T    SmartListInfo;
    ESS_ULONG_T                    i;

    sts = EssOtlGetSmartListInfo(hOutline, ObjHandles,
                                             &SmartListInfo);
    if(!sts)
    {
        printf("\n");
        printf("\tName: %s\n", SmartListInfo->szName);
        printf("\tMissing Name: %s\n",  
                                             SmartListInfo->szMissingName);
        printf("\tOut of Range Name: %s\n",  
                                             SmartListInfo->szOutOfRangeName);
        printf("\tusLen: %d\n", SmartListInfo->usLen);
        for (i = 0; i < SmartListInfo->usLen; i++)
        {  
            printf("\tpIDs: %d, \tpszText[%d]: %s\n",  
                                             SmartListInfo->pIDs[i], i,
                                             SmartListInfo->ppszText[i]);
        }
        printf("\n");
    }
    else
    {
        printf("\t\tEssOtlGetSmartListInfo   sts: %d\n",sts);
    }
}

C Outline API Function Reference  853
if(SmartListInfo)
    sts = EssOtlFreeSmartListInfo(hOutline, SmartListInfo);

**EssOtlGetServerDateFormats**

This function returns the list of server date formats supported.

**Syntax**

```c
ESS_FUNC_M EssOtlGetServerDateFormats(
    ESS_HCTX_T hCtx,
    ESS_STR_T localeStr,
    ESS_USHORT_T* pcount,
    ESS_STR_T** ppdateStrings,
    ESS_STR_T** ppformatStrings)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>Server context handle</td>
</tr>
<tr>
<td>localeStr</td>
<td>ESS_STR_T</td>
<td>Locale in which the example date Strings to be generated.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• If <code>localeStr</code> is empty, the default environment locale is used to generate the date Strings.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• If <code>localeStr</code> is invalid, invalid error message is returned</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• If <code>localeStr</code> is null, error message is returned</td>
</tr>
<tr>
<td>pcount</td>
<td>ESS_USHORT_T*</td>
<td>Count of date formats supported</td>
</tr>
<tr>
<td>ppdateStrings</td>
<td>ESS_STR_T**</td>
<td>Returns the example current date in different date formats as an array (to be deallocated).</td>
</tr>
<tr>
<td>ppformatStrings</td>
<td>ESS_STR_T**</td>
<td>Returns the array of formats supported (to be deallocated).</td>
</tr>
</tbody>
</table>

**Return Value**

Returns:

• 0—If successful

    Values are contained in `ppdateStrings` and `ppformatStrings`.

• Error number—If unsuccessful

**Example**

```c
void TestGetSetDateFormatString()
{
    ESS_STS_T                    sts = ESS_STS_NOERR;
    ESS_HOUTLINE_T              hOutline = ESS_NULL;
    ESS_OBJDEF_T                Object;
    ESS_SHORT_T                 length = 80;
    ESS_STR_T                   dateFormatString = "";
    ESS_STR_T                   localeStr;
    ESS_USHORT_T                count, i;
```
ESS_STR_T* pdateStrings;
ESS_STR_T* pformatStrings;

memset(&Object, '\0', sizeof(Object));
Object.hCtx = hCtx;
Object.ObjType = ESS_OBJTYPE_OUTLINE;
Object.AppName = szAppName;
Object.DbName = szDbName;
Object.FileName = szFileName;

sts = EssOtlOpenOutline(hCtx, &Object,
                         ESS_TRUE, ESS_TRUE, &hOutline);

/* Get current value */
sts = EssOtlGetDateFormatString(hOutline, &dateFormatString);
printf("EssOtlGetSMDateFormatString sts: %d \n", sts);
printf("\tDate format string: %s\n", dateFormatString);

localeStr = "English_UnitedStates.Latin1@Binary";
sts = EssOtlGetServerDateFormats(hCtx, localeStr,
                                  &Count, &pdateStrings, &pformatStrings);
printf("EssOtlGetServerDateFormats sts: %d \n", sts);
for (i = 0; i < count; i++)
{
    printf("\nCase with %s:\n", pformatStrings[i]);
    sts = EssOtlSetDateFormatString(hOutline,
                                     pformatStrings[i]);
    printf("\tEssOtlSetSMDateFormatString sts: %d \n", sts);
    SaveOutline(hOutline);

    sts = EssOtlGetDateFormatString(hOutline,
                                     &dateFormatString);
    printf("\tEssOtlGetSMDateFormatString sts: %d \n", sts);
    printf("\tDate format string: %s\n", dateFormatString);
}

sts = EssUnlockObject(hCtx, Object.ObjType,
                      Object.AppName, Object.DbName, Object.FileName);
sts = EssOtlCloseOutline(hOutline);
printf("EssOtlCloseOutline   sts: %d\n",sts);

See Also

- EssOtlSetDateFormatString
- EssOtlGetDateFormatString

EssOtlGetUpdateTime

Returns a timestamp for the specified outline.

Syntax
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline;</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline handle</td>
</tr>
<tr>
<td>pOtTimeStamp;</td>
<td>ESS_PTIME_T</td>
<td>Pointer to the timestamp for the outline</td>
</tr>
</tbody>
</table>

**Notes**

- The value for time, of type ESS_ULONGLONG_T, is represented by the number of seconds since 00:00:00 1/1/1970 GMT.
- The value for time is not persistent; that is, the value for time is reset whenever the server loads the database.

**Return Value**

Returns the timestamp for the specified outline.

**Example**

```c
ESS_HOUTLINE_T hOutline;
ESS_TIME_T       TimeStamp;

sts = EssOtlGetUpdateTime(hOutline, &TimeStamp);
```

**See Also**

- EssOtlGetOutlineInfo
- EssOtlSetOutlineInfo
- EssOtlVerifyOutline
- EssOtlSortChildren
- EssOtlGenerateCurrencyOutline

---

**EssOtlGetUserAttributes**

Retrieves all user-defined attributes for a member.

**Syntax**

`ESS_FUNC_M EssOtlGetUserAttributes (hOutline, hMember, pusCount, ppAttributeList);`

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline context handle.</td>
</tr>
<tr>
<td>hMember</td>
<td>ESS_HMEMBER_T</td>
<td>Handle of member for which to get the user-defined attribute.</td>
</tr>
<tr>
<td>pusCount</td>
<td>ESS_PUSHORT_T</td>
<td>Count of user attributes returned; defines the number of elements in the</td>
</tr>
<tr>
<td>ppAttributeList</td>
<td>ESS_PPMBRNAME_T</td>
<td>Array of <code>*pusCount</code> members. Each element of the array contains a single user-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>defined attribute string.</td>
</tr>
</tbody>
</table>

---

856  C Outline API Functions
Notes

- A caller can set any number of user-defined attributes for a member using the `EssOtlSetUserAttribute()` call. Each attribute is defined as a unique string that follows the same conventions as member names.
- A user attribute can be the same as any member name, alias, or generation or level name.
- Call `EssFree()` to free the returned attribute list.

Return Value

Returns 0 if successful.

Example

```c
#include <essapi.h>
#include <essotl.h>

ESS_STS_T    sts = 0;
ESS_HOUTLINE_T hOutline;
ESS_OBJDEF_T  Object;
ESS_APPNAME_T szAppName;
ESS_DBNAME_T  szDbName;
ESS_OBJNAME_T szFileName;
ESS_HMEMBER_T hMember;
ESS_USHORT_T  Count, ind;
ESS_PMBRNAME_T AttributeList = ESS_NULL;

memset(&Object, '\0', sizeof(Object));
Object.hCtx = hCtx;
Object.ObjType = ESS_OBJTYPE_OUTLINE;
strcpy(szAppName, "Sample");
strcpy(szDbName, "Basic");
strcpy(szFileName, "Basic");
Object.AppName  = szAppName;
Object.DbName   = szDbName;
Object.FileName = szFileName;

sts = EssOtlOpenOutline(hCtx, &Object, ESS_TRUE,
                        ESS_TRUE, &hOutline);

/************  Get User Attributes ***********/

if (!sts)
{
    sts = EssOtlFindMember(hOutline, "Jan",
                           &hMember);
}

if (!sts && hMember)
{
    sts = EssOtlGetUserAttributes(hOutline,
                                  hMember, &Count, &AttributeList);
}

if (!sts && AttributeList)
{
    printf("User Attribute:\n");
}
```
EssOtlImportExportObject

Imports or exports the contents of the input object to the input file based on \texttt{bImport} being true or false.

**Syntax**

\[
\text{ESS\_FUNC\_M} \ EssOtlImportExportObject(hOutline, objHandle, FileName, bImport)
\]

**Parameter** | **Data Type** | **Description**
--- | --- | ---
\texttt{hOutline} | \texttt{ESS\_HOUTLINE\_T} | Outline handle (Edit mode only)
\texttt{objHandle} | \texttt{ESS\_HOBJECT\_T} | Object handle to be imported or exported
\texttt{FileName} | \texttt{ESS\_STR\_T} | Name of the file to which object needs to be exported or imported from
\texttt{bImport} | \texttt{ESS\_BOOL\_T} | \begin{itemize}
  \item \texttt{true} \hspace{1cm} Import
  \item \texttt{false} \hspace{1cm} Export
\end{itemize}

**Return Value**

Returns:

- \texttt{0}—If successful
- Error number—If unsuccessful

**Example**

```c
void TestImportExportObject()
{
    \texttt{ESS\_STS\_T} \hspace{1cm} \texttt{sts} = \texttt{ESS\_STS\_NOERR};
    \texttt{ESS\_OBJDEF\_T} \hspace{1cm} \texttt{Object};
    \texttt{ESS\_HOUTLINE\_T} \hspace{1cm} \texttt{hOutline} = \texttt{ESS\_NULL};
    \texttt{ESS\_HOBJECT\_T} \hspace{1cm} \texttt{hObjHandle} = \texttt{ESS\_NULL};
    \texttt{ESS\_PHOBJECT\_T} \hspace{1cm} \texttt{hObjHandles};
    \texttt{ESS\_STR\_T} \hspace{1cm} \texttt{sFileName};
    \texttt{ESS\_BOOL\_T} \hspace{1cm} \texttt{bImport};
    \texttt{ESS\_OBJECT\_TYPES} \hspace{1cm} \texttt{objType};
    \texttt{ESS\_STR\_T} \hspace{1cm} \texttt{objName} = \texttt{""};
    \texttt{ESS\_ULONG\_T} \hspace{1cm} \texttt{Count}, \texttt{i};
}
memset(&Object, '\0', sizeof(Object));
Object.hCtx = hCtx;
Object.ObjType = ESS_OBJTYPE_OUTLINE;
Object.AppName = szAppName;
Object.DbName = szDbName;
Object.FileName = szFileName;

sts = EssOtlOpenOutline(hCtx, &Object,
    ESS_TRUE, ESS_TRUE, &hOutline);

/* Create an object for the test */
objType = OBJECT_SMARTLIST;
objName = "CSRatings";

sts = EssOtlCreateObject(hOutline, objType,
    objName, &hObjHandle);

/* Import a SmartList */
sFileName = "F:\testArea\Smartlist\ImpCSRatingsSL.txt";
bImport = ESS_TRUE;
sts = EssOtlImportExportObject(hOutline, hObjHandle,
    sFileName, bImport);
printf("EssOtlImportExportObject sts: %ld\n",sts);

/* Verify import results */
sts = EssOtlListObjects(hOutline, objType,
    &Count, &hObjHandles);
for (i = 0; i < Count; i++)
    DisplaySmartListInfo(hOutline, hObjHandles[i]);

SaveOutline(hOutline);

printf("\n");
objName = "CSRatings";
sts = EssOtlFindObject(hOutline, objType,
    objName, &hObjHandle);
printf("EssOtlFindObject sts: %ld\n",sts);

/* Export a SmartList */
bImport = ESS_FALSE;
sts = EssOtlImportExportObject(hOutline, hObjHandle,
    sFileName, bImport);
/* Unlock objects */
sts = EssUnlockObject(hCtx, Object.ObjType,
    Object.AppName, Object.DbName, Object.FileName);
/* Close */
sts = EssOtlCloseOutline(hOutline);

See Also
- EssOtlGetMemberSmartList
- EssOtlCreateObject
- EssOtlDeleteObject
EssOtlIsMemberNonUnique

Discovers if a member name is duplicate.

Syntax

ESS_FUNC_M EssOtlIsMemberNonUnique (hOutline, hMember, fNameNonUnique);

Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline context handle (input).</td>
</tr>
<tr>
<td>hMember</td>
<td>ESS_HMEMBER_T</td>
<td>The member to query for non-uniqueness (input).</td>
</tr>
<tr>
<td>*fNameNonUnique</td>
<td>ESS_BOOL_T</td>
<td>TRUE if the member queried is a duplicate member name (output).</td>
</tr>
</tbody>
</table>

Notes

- Before you call this function, call EssOtlOpenOutline to open the outline in editing mode.
- Use aMember Traversal Function to get a member handle for the second argument of this function.
- This function checks whether a member name is duplicated. If a member name is duplicated, you might be interested in getting the fully qualified name of a member (its unique name or its key), because if a non specific name is used by other functions in your program to refer to a member name that is duplicated, unexpected behavior could occur.
- However, if all names are unique, you do not need to spend the resources to use fully qualified names or keys.
- After you use this function, if you discover that a member name is duplicated, you can get the fully qualified name and save it somewhere using EssOtlGetMemberUniqueName.

Return Value

Returns 0 if successful; otherwise, returns an error.
Example

ESS_FUNC_M ESS_ISUniqMemberName()
{
    ESS_STS_T    sts = 0;
    ESS_HOUTLINE_T   hOutline;
    ESS_OBJDEF_T   Object;
    ESS_APPNAME_T   szAppName;
    ESS_DBNAME_T   szDbName;
    ESS_OBJNAME_T   szFileName;
    ESS_HMEMBER_T   hMemberParent, hMemberChild;
    ESS_BOOL_T    pbNameUnique;

    memset(&Object, '\0', sizeof(Object));
    Object.hCtx =   hCtx;
    Object.ObjType =  ESS_OBJTYPE_OUTLINE;
    strcpy(szAppName, "Demo");
    strcpy(szDbName, "Test");
    strcpy(szFileName, "Test");
    Object.AppName =  szAppName;
    Object.DbName =   szDbName;
    Object.FileName =  szFileName;

    sts = EssOtlOpenOutline(hCtx, &Object, ESS_FALSE,
                                    ESS_TRUE, &hOutline);

    if (!sts)
    {
        sts = EssOtlFindMember(hOutline, "2004", &hMemberParent);
    }

    if (!sts && hMemberParent)
    {
        sts = EssOtlGetChild(hOutline, hMemberParent, &hMemberChild);
    }

    if (!sts)
    {
        //Check whether Qtr1 is unique member name, returns 0 if unique and 1 if non-unique
        sts =  EssOtlIsMemberNameNonUnique (hOutline, hMemberChild, &pbNameUnique);
        if (sts)
            printf("EssOtlIsMemberNameNonUnique failed sts %ld\n",sts);
    }

    return sts;
}

See Also

- ESS_IsMemberNameUniqueWithinDim
- ESS_IsMemberNameUniqueWithinDimAtGenLevel
EssOtlIsMemberNameUniqueWithinDim

Discovers if member names are all unique within a dimension.

Syntax

ESS_FUNC_M EssOtlIsMemberNameUniqueWithinDim (hOutline, hDim, *pbNameUnique);

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline context handle (input).</td>
</tr>
<tr>
<td>hDim</td>
<td>ESS_HMEMBER_T</td>
<td>Input dimension, returned by EssOtlQueryGetFirstDimension() or EssOtlQueryGetNextDimension().</td>
</tr>
<tr>
<td>*pbNameUnique</td>
<td>ESS_BOOL_T</td>
<td>TRUE if the dimension queried contains no duplicate member names; FALSE otherwise.</td>
</tr>
</tbody>
</table>

Notes

- This function is one of three functions that query for member name uniqueness or non-uniqueness.
  - EssOtlIsMemberNameNonUnique discovers if a member name is duplicate within an outline.
  - EssOtlIsMemberNameUniqueWithinDim discovers if all member names are unique within a dimension.
  - EssOtlIsMemberNameUniqueWithinDimAtGenLevel discovers if all member names are unique within a dimension at the generation or level specified.
- Before you call this function, call EssOtlOpenOutlineQuery() to open the outline in query mode.

Return Value

Returns 0 if successful; otherwise, returns an error.

Example

ESS_FUNC_M ESS_ISUniq()
{
    ESS_STS_T    sts = 0;
    ESS_HOUTLINE_T  hOutline;
    ESS_OBJDEF_T    Object;
    ESS_APPNAME_T   szAppName;
    ESS_DBNAME_T    szDbName;
    ESS_OBJNAME_T   szFileName;
    ESS_HMEMBER_T   hDim = ESS_NULL;
    ESS_BOOL_T     pbNameUnique = 0;

    memset(&Object, '\0', sizeof(Object));
    Object.hCtx = hCtx;
    Object.ObjType = ESS_OBJTYPE_OUTLINE;
    strcpy(szAppName, "Demo");
    strcpy(szDbName, "Test");
    strcpy(szFileName, "Test");
}
Object.AppName = szAppName;
Object.DbName = szDbName;
Object.FileName = szFileName;

sts = EssOtlOpenOutlineQuery (hCtx, &Object, &hOutline);

if (!sts)
{
    sts = EssOtlQueryGetFirstDimension (hOutline, &hDim);
    if (sts)
        printf("EssOtlQueryGetFirstDimension failed sts %ld\n", sts);
}

if (!sts)
{
    sts = EssOtlIsMemberNameUniqueWithinDim (hOutline, hDim, &pbNameUnique);
    if (sts)
        printf("EssOtlIsMemberNameUniqueWithinDim failed sts %ld\n", sts);
    else
        printf("pbNameUnique is %d\n", pbNameUnique);
}

return sts;

See Also

- EssOtlGetCountOfDupMemberNameInDim
- EssOtlIsMemberNameNonUnique
- EssOtlIsMemberNameUniqueWithinDimAtGenLevel

EssOtlIsMemberNameUniqueWithinDimAtGenLevel

Discovers if all member names are unique within a dimension at the generation or level specified.

Syntax

ESS_FUNC_M EssOtlIsMemberNameUniqueWithinDimAtGenLevel (hOutline, hDim, bGen, usGenLevel, *pbNameUnique);

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline context handle.</td>
</tr>
<tr>
<td>hDim</td>
<td>ESS_HMEMBER_T</td>
<td>Input dimension, returned by EssOtlQueryGetFirstDimension() or EssOtlQueryGetNextDimension().</td>
</tr>
<tr>
<td>bGen</td>
<td>ESS_BOOL_T</td>
<td>Input. If TRUE, usGenLevel is considered a generation number. If FALSE, usGenLevel is considered a level number.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>---------------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>usGenLevel</td>
<td>ESS_USHORT_T</td>
<td>Input generation or level number.</td>
</tr>
<tr>
<td>*pbNameUnique</td>
<td>ESS_BOOL_T</td>
<td>Output. TRUE if the dimension queried contains duplicate member names at</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the generation or level specified; FALSE otherwise.</td>
</tr>
</tbody>
</table>

**Notes**

- This function is one of three functions that query for member name uniqueness or non uniqueness.
  - `EssOtlIsMemberNameNonUnique` discovers if a member name is duplicate within an outline.
  - `EssOtlIsMemberNameUniqueWithinDim` discovers if all member names are unique within a dimension.
  - `EssOtlIsMemberNameUniqueWithinDimAtGenLevel` discovers if all member names are unique within a dimension at the generation or level specified.
- Before you call this function, call `EssOtlOpenOutlineQuery()` to open the outline in query mode.

**Return Value**

Returns 0 if successful; otherwise, returns an error.

**Example**

```c
ESS_FUNC_M ESS_ISUniqMemberNameWithinDimatGenLev()
{
    ESS_STS_T    sts = 0;
    ESS_HOUTLINE_T   hOutline;
    ESS_OBJDEF_T   Object;
    ESS_APPNAME_T   szAppName;
    ESS_DBNAME_T   szDbName;
    ESS_OBJNAME_T   szFileName;
    ESS_HMEMBER_T   hDim,hNextDim;
    ESS_BOOL_T    pbNameUnique, bGen = ESS_TRUE;
    ESS_USHORT_T   usGenLevel = 3;

    memset(&Object, '\0', sizeof(Object));
    Object.hCtx =   hCtx;
    Object.ObjType =  ESS_OBJTYPE_OUTLINE;
    strcpy(szAppName, "Demo");
    strcpy(szDbName, "Test");
    strcpy(szFileName, "Test");
    Object.AppName =  szAppName;
    Object.DbName =   szDbName;
    Object.FileName =  szFileName;

    sts = EssOtlOpenOutlineQuery (hCtx, &Object, &hOutline);

    if (!sts)
```
{ 
  sts = EssOtlQueryGetFirstDimension(hOutline, &hDim);

  if (sts)
    printf("EssOtlQueryGetFirstDimension failed sts %ld\n",sts);
}

if (!sts)
{
  sts = EssOtlIsMemberNameUniqueWithinDimAtGenLevel (hOutline, hDim, bGen, usGenLevel, &pbNameUnique);
  if (sts)
    printf("EssOtlIsMemberNameUniqueWithinDimAtGenLevel failed sts %ld\n",sts);
  else
    printf("pbNameUnique is %d\n", pbNameUnique);
}

  if (!sts)
  {
    sts = EssOtlQueryGetNextDimension (hOutline, hDim, &hNextDim);

    if (sts)
      printf("EssOtlQueryGetFirstDimension failed sts %ld\n",sts);
  }

  if (!sts)
  {
    sts = EssOtlIsMemberNameUniqueWithinDimAtGenLevel (hOutline, hNextDim, bGen, usGenLevel, &pbNameUnique);
    if (sts)
      printf("EssOtlIsMemberNameUniqueWithinDimAtGenLevel failed sts %ld\n",sts);
    else
      printf("pbNameUnique is %d\n", pbNameUnique);
  }

  return sts;
}

See Also

● EssOtlGetCountOfDupMemberNameInDim
● EssOtlIsMemberNameNonUnique
● EssOtlIsMemberNameUniqueWithinDim

EssOtlListObjects

Returns an array of all object handles of the specified type.

Syntax

ESS_FUNC_M EssOtlListObjects(hOutline, objType, pCount, pObjHandles)
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline handle (Edit mode only)</td>
</tr>
<tr>
<td>objType</td>
<td>ESS_OBJECT_TYPES</td>
<td>Object type with one of the following values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- OBJECT_SMARTLIST</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Object type is Text List (SmartList)</td>
</tr>
<tr>
<td>pCount</td>
<td>ESS_ULONGLONG_T*</td>
<td>Count of object handles</td>
</tr>
<tr>
<td>pObjHandles</td>
<td>ESS_PPPOBJECT_T</td>
<td>Returns an array of object handles. Must be deallocated using EssFree.</td>
</tr>
</tbody>
</table>

**Return Value**

Returns:

- **0**—If successful
  
  `pCount` and `pObjHandles` contain values.

- **Error number**—If unsuccessful
  
  `pCount` and `pObjHandles` are NULL.

**Example**

```c
void TestCreateObject()
{
    ESS_STS_T           sts = ESS_STS_NOERR;
    ESS_HOUTLINE_T      hOutline = ESS_NULL;
    ESS_OBJDEF_T        Object;
    ESS_OBJECT_TYPES    objType;
    ESS_STR_T           smartListName;
    ESS_HOBJECT_T       ObjHandle;
    ESS_ULONGLONG_T     Count, i;
    ESS_PHOBJECT_T      ObjHandles;
    ESS_HOBJECT_T       hObjHandle;
    ESS_HSMARTLIST_T    hSmartList;
    ESS_STR_T           objName;

    memset(&Object, '\0', sizeof(Object));
    Object.hCtx = hCtx;
    Object.ObjType = ESS_OBJTYPE_OUTLINE;
    Object.AppName = szAppName;
    Object.DbName = szDbName;
    Object.FileName = szFileName;

    /* Open outline */
    sts = EssOtlOpenOutline(hCtx, &Object,
                             ESS_TRUE, ESS_TRUE, &hOutline);

    /* Create a static SmartList */
    objType = OBJECT_SMARTLIST;
    smartListName = "SList1";
    sts = EssOtlCreateObject(hOutline, objType,
                             smartListName, &ObjHandle);

    /* List all SmartList objects */
    objType = OBJECT_SMARTLIST;
```
sts = **EssOtlListObjects**(hOutline, objType, 
    &Count, &ObjHandles);

/* Save */
SaveOutline(hOutline);

/* Find objects */
objName = "SList1";
sts = EssOtlFindObject(hOutline, objType, objName, 
    &hObjHandle);

/* Delete objects */
hSmartList = (ESS_HSMARTLIST_T)hObjHandle;
sts = EssOtlDeleteObject(hOutline, hSmartList);
SaveOutline(hOutline);

if(ObjHandles)
    EssFree(hInst, ObjHandles);

/* Unlock objects */
sts = EssUnlockObject(hCtx, Object.ObjType, 
    Object.AppName, Object.DbName, Object.FileName);

/* Close outline */
sts = EssOtlCloseOutline(hOutline);
}

**See Also**
- EssOtlGetMemberSmartList
- EssOtlCreateObject
- EssOtlDeleteObject
- EssOtlGetSmartListInfo
- EssOtlFindObject
- EssOtlFreeObjectArray
- EssOtlFreeSmartListInfo
- EssOtlGetMemberSmartList
- EssOtlGetMemberType
- EssOtlGetObjectReferenceCount
- EssOtlGetObjectReferences
- EssOtlImportExportObject
- EssOtlListObjects
- EssOtlQueryObjects
- EssOtlSetMemberType
- EssOtlSetMemberTypeToSmartList

**EssOtlMoveMember**

Moves a member.

**Syntax**

```c
ESS_FUNC_M EssOtlMoveMember (hOutline, hMember, hNewParent, hNewPrevSibling);
```
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline context handle</td>
</tr>
<tr>
<td>hMember</td>
<td>ESS_HMEMBER_T</td>
<td>Handle of member to move</td>
</tr>
<tr>
<td>hNewParent</td>
<td>ESS_HMEMBER_T</td>
<td>Handle of new parent. Use this field only if the hNewPrevSibling field is ESS_NULL.</td>
</tr>
<tr>
<td>hNewPrevSibling</td>
<td>ESS_HMEMBER_T</td>
<td>Handle of new previous sibling</td>
</tr>
</tbody>
</table>

**Notes**

- The moved member is inserted following the `hPrevSibling` member. If this field is ESS_NULL, the moved member becomes the first child of the parent specified by `hParent`.
- If both `hParent` and `hPrevSibling` are ESS_NULL, the moved member becomes the first dimension in the outline.
- Moving a zero-level (leaf node) attribute member that is not of type ESS_ATTRMRBDFRT_STRING resets the member’s long name, using the specifications for the outline in the "ESS_ATTRSPECS_T" on page 119 structure.
- Moving an ancestor may affect the long name of a zero-level attribute member.

**Return Value**

Returns 0 if successful; otherwise:

OTLAPI_BAD_MOVE

**Example**

```c
#include <essapi.h>
#include <essotl.h>

ESS_STS_T          sts = 0;
ESS_HOUTLINE_T     hOutline;
ESS_HMEMBER_T      hMemberJan;
ESS_HMEMBER_T      hMemberMar;
ESS_OBJDEF_T       Object;
ESS_APPNAME_T      szAppName;
ESS_DBNAME_T       szDbName;
ESS_OBJNAME_T      szFileName;
memset(&Object, '\0', sizeof(Object));
Object.hCtx = hCtx;
Object.ObjType = ESS_OBJTYPE_OUTLINE;
strcpy(szAppName, "Sample");
strcpy(szDbName, "Basic");
strcpy(szFileName, "Basic");
Object.AppName = szAppName;
Object.DbName = szDbName;
Object.FileName = szFileName;

sts = EssOtlOpenOutline(hCtx, &Object, ESS_TRUE, ESS_TRUE, &hOutline);
```

---

868  C Outline API Functions
if (!sts)
{
    sts = EssOtlFindMember(hOutline, "Jan",
                        &hMemberJan);
}

if (!sts && hMemberJan)
{
    sts = EssOtlFindMember(hOutline, "Mar",
                        &hMemberMar);
}

if (!sts && hMemberMar)
{
    sts = EssOtlMoveMember(hOutline, hMemberJan,
                        ESS_NULL, hMemberMar);
}

See Also

- EssOtlFindMember
- EssOtlRenameMember
- EssOtlAddMember
- EssOtlDeleteMember

**EssOtlNewOutline**

Creates an outline without creating a file. This call is used as an alternative to EssOtlOpenOutline().

**Syntax**

```c
ESS_FUNC_M EssOtlNewOutline (hCtx, pNewInfo, phOutline);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>Essbase Context handle.</td>
</tr>
<tr>
<td>pNewInfo</td>
<td>&quot;ESS_OUTLINEINFO_T&quot; on page 706</td>
<td>Structure describing the new outline.</td>
</tr>
<tr>
<td>phOutline</td>
<td>ESS_PHOUTLINE_T</td>
<td>Pointer to ESS_HOUTLINE_T variable. This handle is set by the API and should be passed in to subsequent Outline API functions.</td>
</tr>
</tbody>
</table>

**Notes**

- This function creates an empty outline in memory.
- No transactions are kept when this call is used. See EssOtlOpenOutline() for more information on keeping transactions.

**Return Value**

Returns 0 if successful.
# EssOtlOpenOutline

Opens and reads in an existing outline. This function (or EssOtlNewOutline()) must be called before any operations on the outline can take place.

## Syntax

```c
ESS_FUNC_M EssOtlOpenOutline (hCtx, pObject, fLock, fKeepTrans, phOutline);
```

### Parameter | Data Type | Description
--- | --- | ---
| hCtx | ESS_HCTX_T | Essbase Context handle.
| pObject | ESS_POBJDEF_T | Pointer to an object structure defining the outline object to open.
| fLock | ESS_BOOL_T | Flag to determine if the outline should be locked when it is opened. This is valid only for server outlines.
| fKeepTrans | ESS_BOOL_T | Flag to determine whether to keep transactions. If you are opening an existing outline to make changes, and you intend to restructure the database and keep data, we recommend that you set this flag to ESS_TRUE. When ESS_TRUE, a log is kept of activities done to the outline. If you are starting from an empty outline or are not planning on saving data when you restructure, we recommend that you set this field to ESS_FALSE. When ESS_FALSE, no log is kept, saving time and memory.
| phOutline | ESS_PHOUTLINE_T | Pointer to an ESS_HOUTLINE_T variable. This handle is set by the API and should be passed to subsequent Outline API functions.

### Notes
- For Unicode mode outlines, use EssOtlOpenOutlineEx.
- If the outline file exists on the server, this call copies the file locally for client access.
For aggregate storage database outlines, EssOtlOpenOutline keeps the outline open until EssOtlCloseOutline is called. Because aggregate storage outlines are paged into memory (instead of being read entirely into memory) the outline is kept open. As a result, temporary files remain in your computer’s Temp folder until EssOtlCloseOutline is called.

Return Value

Returns 0 if successful; otherwise one of the following:

- OTLAPI_BAD_OBJTYPE
- OTLAPI_ERR_FILEOPEN
- OTLAPI_ERR_FILEIO

Access

This function requires you to have the appropriate level of access to the specified application and/or database to contain the outline object. To lock the outline object (lock flag is ESS_TRUE), you must have Application Designer or Database Designer privilege (ESS_PRIV_APPDESIGN or ESS_PRIV_DBDESIGN) for the specified application or database containing the outline.

Example

```c
#include <essapi.h>
#include <essotl.h>

ESS_STS_T    sts = 0;
ESS_OBJDEF_T  Object;
ESS_HOUTLINE_T  hOutline;
ESS_APPNAME_T szAppName;
ESS_DBNAME_T  szDbName;
ESS_OBJNAME_T szFileName;

memset(&Object, '\0', sizeof(Object));
Object.hCtx = hCtx;
Object.ObjType = ESS_OBJTYPE_OUTLINE;
strcpy(szAppName, "Sample");
strcpy(szDbName, "Basic");
strcpy(szFileName, "Basic");
Object.AppName = szAppName;
Object.DbName = szDbName;
Object.FileName = szFileName;
sts = EssOtlOpenOutline(hCtx, &Object, ESS_TRUE,
                        ESS_TRUE, &hOutline);
```

See Also

- EssOtlOpenOutlineEx
- EssOtlCloseOutline
- EssOtlGetMemberCommentEx
- EssOtlNewOutline
- EssOtlRestructure
- EssOtlSetMemberCommentEx
- EssOtlVerifyOutline
- EssOtlWriteOutline
EssOtlOpenOutlineEx

Opens and reads in an existing outline, identifying the correct locale. This function (or EssOtlNewOutline()) must be called before any operations on the outline can take place.

Syntax

```c
ESS_FUNC_M EssOtlOpenOutlineEx(hCtx, pObject, fLock, fKeepTrans, pLocaleDescription, phOutline);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>Essbase Context handle.</td>
</tr>
<tr>
<td>pObject</td>
<td>ESS_POBJDEF_T</td>
<td>Pointer to an object structure defining the outline object to open.</td>
</tr>
<tr>
<td>fLock</td>
<td>ESS_BOOL_T</td>
<td>Flag to determine if the outline should be locked when it is opened. This is valid only for server outlines.</td>
</tr>
<tr>
<td>fKeepTrans</td>
<td>ESS_BOOL_T</td>
<td>Flag to determine whether to keep transactions.</td>
</tr>
<tr>
<td>pLocaleDescription</td>
<td></td>
<td>The identifier used by GlobalC to identify the Locale. The LocaleDescription is in the form of: <code>[language]_[territory].[codepage]@[sort]</code> For example: <code>Japanese_Japan.MS932@binary</code> to provide the locale description of the language of the outline file. It is the program’s responsibility to pass pLocaleDescription in the current format.</td>
</tr>
<tr>
<td>phOutline</td>
<td>ESS_PHOUTLINE_T</td>
<td>Pointer to an ESS_HOUTLINE_T variable. This handle is set by the API and should be passed to subsequent Outline API functions.</td>
</tr>
</tbody>
</table>

Notes

- This function works like EssOtlOpenOutline(), but with the addition of a Unicode-specific LocaleDescription argument.
- If the outline file exists on the server, this call copies the file locally for client access.
- For aggregate storage database outlines, EssOtlOpenOutline keeps the outline open until EssOtlCloseOutline is called. Because aggregate storage outlines are paged into memory (instead of being read entirely into memory) the outline is kept open. As a result, temporary files remain in your computer’s Temp folder until EssOtlCloseOutline is called.

Return Value

Returns 0 if successful; otherwise one of the following:

- OTLAPI_BAD_OBJTYPE
Access

This function requires you to have the appropriate level of access to the specified application and/or database to contain the outline object. To lock the outline object (lock flag is ESS_TRUE), you must have Application Designer or Database Designer privilege (ESS_PRIV_APPDESIGN or ESS_PRIV_DBDESIGN) for the specified application or database containing the outline.

See Also

- EssOtlCloseOutline
- EssOtlGetMemberCommentEx
- EssOtlNewOutline
- EssOtlRestructure
- EssOtlSetMemberCommentEx
- EssOtlVerifyOutline
- EssOtlWriteOutlineEx

**EssOtlOpenOutlineQuery**

Opens an existing outline.

**Syntax**

```plaintext
ESS_FUNC_M EssOtlOpenOutlineQuery (hCtx, pObject, phOutline);
```

**Parameter**  |  **Data Type**  |  **Description**
--- | --- | ---
`hCtx` | ESS_HCTX_T | Outline context handle. This must be a valid server login context.
`pObject` | ESS_POBJDEF_T | Pointer to object structure defining the outline object to open. Currently this is ignored. You should call EssSetActive() for the database you are accessing.
`phOutline` | ESS_PHOUTLINE_T | Pointer to an ESS_HOUTLINE_T variable. This will be set by the API and should be passed in to subsequent API functions.

**Notes**

- Use this function to access an outline using EssOtlQueryMembers().
- The call will not download the outline and load the entire file into memory.
- Therefore, many of the outline API calls will not work with `hOutline` that is passed back from this call.
- The following calls are accessible after this call is made. All other Outline API calls will return an error.
  - EssOtlCloseOutline
  - EssOtlGetMemberAlias
  - EssOtlGetMemberFormula
EssOtlGetMemberInfo
EssOtlGetNextAliasCombination
EssOtlGetOutlineInfo
EssOtlGetUserAttributes
EssOtlGetGenName
EssOtlGetGenNames
EssOtlGetLevelName
EssOtlGetLevelNames
EssOtlGetMemberLastFormula

Return Value
The return value is zero if the function was successful.

- OTLAPI_BAD_OBJTYPE
- OTLAPI_ERR_FILEOPEN
- OTLAPI_ERR_FILEIO

Example
#include <essapi.h>
#include <essotl.h>

ESS_STS_T          sts = ESS_STS_NOERR;
ESS_HOUTLINE_T     hOutline;
ESS_OBJDEF_T       Object;
ESS_ACCESS_T       Access;
ESS_STR_T          AppName;
ESS_STR_T          DbName;

AppName = "Sample";
DbName = "Basic";

sts = EssSetActive(hCtx, AppName, DbName, &Access);

if (sts == 0)
{
    memset(&Object, '\0', sizeof(Object));
    sts = EssOtlOpenOutlineQuery(hCtx, &Object, &hOutline);
}

See Also
- EssOtlCloseOutline
- EssOtlOpenOutline
- EssOtlQueryMembers
- EssOtlQueryMembersByName
- EssSetActive
EssOtlPutSmartList

Populates the contents of the Text List (SmartList) handle created by EssOtlCreateObject. The object handle created can be typecast to an ESS_HSMARTLIST_T handle.

Verification rules:
- Each entry in pIDs and in ppszText must be unique
- The strings in ppszText must pass the same name validation rules as specified for text list names.
- ppszText text strings may not be empty, #OUTOF RANGE, or the same as pszMissingName or pszOutOfRangeName
- The number of entries len cannot be more than 1024.

Syntax

```c
ESS_FUNC_M EssOtlPutSmartList(hOutline, hSmartList, len, *pIDs, *ppszText, pszMissingName, pszOutOfRangeName);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>The source Essbase outline for the text list.</td>
</tr>
<tr>
<td>hSmartList</td>
<td>ESS_HSMARTLIST_T</td>
<td>Text list handle</td>
</tr>
<tr>
<td>len</td>
<td>ESS_UINT16</td>
<td>Number of items</td>
</tr>
<tr>
<td>*pIDs</td>
<td>ESS_UINT32_T</td>
<td>Integer IDs</td>
</tr>
<tr>
<td>*ppszText</td>
<td>ESS_STR_T</td>
<td>Enumerated text</td>
</tr>
<tr>
<td>pszMissingName</td>
<td>ESS_STR_T</td>
<td>Name of the missing smart text</td>
</tr>
<tr>
<td>pszOutOfRangeName</td>
<td>ESS_STR_T</td>
<td>Name of the out of range smart text.</td>
</tr>
</tbody>
</table>

Return Value

Returns:
- 0—If successful
  - pIDs and ppszText contain values.
- Error number—If unsuccessful
  - pIDs and ppszText are NULL.

Example

```c
void TestPutSmartList()
{
    ESS_STS_T                        sts = ESS_STS_NOERR;
    ESS_HOUTLINE_T                hOutline = ESS_NULL;
    ESS_OBJECT_TYPES            objType;
    ESS_HOBJECT_T                hObjHandle;
    ESS_PHOBJECT_T                hObjHandles;
    ESS_PSMARTLISTINFO_T    SmartListInfo = ESS_NULL;
    ESS_OBJDEF_T                    Object;
```
/** Open outline */
sts = EssOtlOpenOutline(hCtx, &Object, ESS_TRUE,
                         ESS_TRUE, &hOutline);

/* Create a SmartList */
objType = OBJECT_SMARTLIST;
sts = EssOtlCreateObject(hOutline, objType,
                         smartListNames[0], &hObjHandle);

/* Set up and put SmartList */
hSmartList = (ESS_HSMARTLIST_T)hObjHandle;
len = 4;
pIds[0] = 1;
pIds[1] = 2;
pIds[2] = 3;
pIds[3] = -1;
ppszText[0] = "Red";
ppszText[1] = "Green";
ppszText[2] = "Blue";
ppszText[3] = "Yellow";
pszMissingName = "Missing";
pszOutOfRangeName = "OutOfRange";
sts = EssOtlPutSmartList(hOutline, hSmartList,
                         len, pIds, ppszText, pszMissingName,
                         pszOutOfRangeName);

SaveOutline(hOutline);

/* Clean up */
for(i = 0; i <= 12; i++)
{
    sts = EssOtlFindObject(hOutline, objType,
                           smartListNames[i], &hObjHandle);
    hSmartList = (ESS_HSMARTLIST_T)hObjHandle;
    sts = EssOtlDeleteObject(hOutline, hSmartList);
}

SaveOutline(hOutline);

objType = OBJECT_SMARTLIST;
sts = EssOtlListObjects(hOutline, objType, &Count, &hObjHandles);
for (i = 0; i < Count; i++)
    DisplaySmartListInfo(hOutline, hObjHandles[i]);
if(hObjHandles)
    EssFree (hInst, hObjHandles);

sts = EssUnlockObject(hCtx, Object.ObjType,
                Object.AppName, Object.DbName, Object.FileName);
sts = EssOtlCloseOutline(hOutline);

EssOtlQueryAttributes

Queries member information for a given attribute member or dimension.

Syntax

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline;</td>
<td>ESS_HOUTLINE_T</td>
<td>Handle to the outline</td>
</tr>
<tr>
<td>pAttributeQuery;</td>
<td>“ESS_ATTRIBUTEQUERY_T” on page 697</td>
<td>Structure that defines the query</td>
</tr>
<tr>
<td>pCount;</td>
<td>ESS_PULONG_T</td>
<td>Number of member handles returned</td>
</tr>
<tr>
<td>pphMemberArray;</td>
<td>ESS_PPHMEMBER_T</td>
<td>Pointer to an array of member handles returned</td>
</tr>
</tbody>
</table>

Notes

Before you call this function, call EssOtlOpenOutlineQuery to open the outline in query mode.

Example

void ESS_OtlQueryAttributes()
{
    ESS_STS_T             sts = ESS_STS_NOERR;
    ESS_HMEMBER_T         hMember;
    ESS_OBJDEF_T          Object;
    ESS_APPNAME_T         szAppName;
    ESS_DBNAME_T          szDbName;
    ESS_OBJNAME_T         szFileName;
    ESS_SHORT_T           hOutlineQuery;
    ESS_ATTRIBUTEQUERY_T  pAttributeQuery;
    ESS ULONG_T           Count = 0;
    ESS_PPHMEMBER_T       phMemberArray = ESS_NULL;
    ESS_PMBRINFO_T        pMbrInfo = ESS_NULL;
    int                   index;

    memset(&pAttributeQuery, 0x00, sizeof(ESS_ATTRIBUTEQUERY_T));
    memset(&Object, '\0', sizeof(Object));
    Object.hCtx = hCtx;
    Object.ObjType = ESS_OBJTYPE_OUTLINE;
    strcpy(szAppName, "Sample");
strcpy(szDbName, "Basic");
strcpy(szFileName, "Basic");
Object.AppName = szAppName;
Object.DbName = szDbName;
Object.FileName = szFileName;

sts = EssOtlOpenOutlineQuery(hCtx, &Object, &hOutlineQuery);
printf("EssOtlOpenOutlineQuery() sts: %ld\n", sts);
pAttributeQuery.bInputMemberIsHandle == ESS_FALSE;
pAttributeQuery.uInputMember.szMember = "100-10";
pAttributeQuery.usInputMemberType = ESS_BASE_MEMBER;
pAttributeQuery.usOutputMemberType = ESS_ATTRIBUTE_MEMBER;
pAttributeQuery.usOperation = ESS_ALL;
pAttributeQuery.Attribute.usDataType = ESS_ATTRMBRDT_NONE;

sts = EssOtlQueryAttributes(hOutlineQuery, &pAttributeQuery, &Count, &phMemberArray);
printf("EssOtlQueryAttributes() sts: %ld\n", sts);

if (!sts && phMemberArray)
{
    if (Count && phMemberArray)
    {
        sts = EssOtlFreeMembers(hOutlineQuery, Count, phMemberArray);
    }
    printf("\n------- Query Results -------\n");
    for (index = 0; index < Count; index++)
    {
        sts = EssOtlGetMemberInfo(hOutlineQuery,phMemberArray[index],&pMbrInfo);
        printf("%s\n",pMbrInfo->szMember);
    }
}

See Also

- EssCheckAttributes
- EssFreeStructure
- EssGetAssociatedAttributesInfo
- EssGetAttributeInfo
- EssGetAttributeSpecifications
- EssOtlAssociateAttributeDimension
- EssOtlAssociateAttributeMember
- EssOtlDisassociateAttributeDimension
- EssOtlDisassociateAttributeMember
- EssOtlFindAttributeMembers
- EssOtlFreeStructure
- EssOtlGetAssociatedAttributes
- EssOtlGetAttributeInfo
- EssOtlGetAttributeSpecifications
- EssOtlSetAttributeSpecifications

C Outline API Functions
EssOtlQueryAttributesEx
Queries member information for a given attribute member or dimension.

Syntax

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline;</td>
<td>ESS_HOUTLINE_T</td>
<td>Handle to the outline</td>
</tr>
<tr>
<td>pAttributeQuery;</td>
<td>“ESS_ATTRIBUTEQUERY_T” on page 697</td>
<td>Structure that defines the query</td>
</tr>
<tr>
<td>pCount;</td>
<td>ESS_PMBRCOUNTS_T</td>
<td>Number of member handles returned</td>
</tr>
<tr>
<td>pphMemberArray;</td>
<td>ESS_PPHMEMBER_T</td>
<td>Pointer to an array of member handles returned</td>
</tr>
</tbody>
</table>

Notes

Before you call this function, call EssOtlOpenOutlineQuery to open the outline in query mode.

See Also

- EssCheckAttributes
- EssFreeStructure
- EssGetAssociatedAttributesInfo
- EssGetAttributeInfo
- EssGetAttributeSpecifications
- EssOtlQueryAttributes
- EssOtlOpenOutlineQuery
- EssOtlAssociateAttributeDimension
- EssOtlAssociateAttributeMember
- EssOtlDisassociateAttributeDimension
- EssOtlDisassociateAttributeMember
- EssOtlFindAttributeMembers
- EssOtlFreeStructure
- EssOtlGetAssociatedAttributes
- EssOtlGetAttributeInfo
- EssOtlGetAttributeSpecifications
- EssOtlSetAttributeSpecifications

EssOtlQueryGenerationInfo

EssOtlQueryGenerationInfo() queries for the time dimension generation information contained in the comment field for the dimension's top member. Once this information is known, it can be used with EssOtlGetLinkedAttributeAttachLevel() to provide period over period analysis.

Syntax

ESS_FUNC_M EssOtlQueryGenerationInfo (hOutline, szName, queryID, ppReturns);
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline handle. This must have been returned from EssOtlOpenOutlineQuery().</td>
</tr>
<tr>
<td>szName</td>
<td>ESS_STS_T</td>
<td>Name of the top member of the date-time dimension</td>
</tr>
<tr>
<td>queryID</td>
<td>ESS_ULONG_T</td>
<td>Use the query identifier constant ESS_OTLQRYTIDIM_TIMEPERIODS</td>
</tr>
<tr>
<td>ppReturns</td>
<td>ESS_PPVOID_T</td>
<td>A pointer to the query information structure for this dimension.</td>
</tr>
</tbody>
</table>

**Notes**

The caller of EssOtlQueryGenerationInfo() should call EssOtlFreeStructure() with structure ID ESS_DT_STRUCT_TGENINFO to free the memory set aside for the returned structure pointer.

**Return Value**

If successful, returns a pointer to a ESS_PTIMEDIM_GENINFO_T structure.

**Example**

```c
SS_STR_T strBuf1 = "Year";
ESS_ULONG_T queryId = ESS_OTLQRYTIDIM_TIMEPERIODS;
ESS_PVOID_T pReturns;
ESS_PTIMEDIM_GENINFO_T tpStruc = NULL;

sts = EssOtlQueryGenerationInfo (hOutline, /*query outline handle*/
   strBuf1, /* IN - date-time dimension member name*/
   queryId, /* IN - query ID */
   &pReturns);

if (sts)
   goto exit;

switch (queryId)
{
   case ESS_OTLQRYTIDIM_TIMEPERIODS:
      tpStruc = (ESS_PTIMEDIM_GENINFO_T)pReturns;

      for (ii = 0; ii < tpStruc->usCount; ii++)
         fprintf(cmdctxp->output, "Time period for Gen %d = %s\n", ii+1, TimePeriodNames[tpStruc->ptps[ii]]);

      sts = EssOtlFreeStructure (cmdctxp->hOutline[hOutlineChoice], ESS_DT_STRUCT_TGENINFO, 1, pReturns);
      if (sts)
         goto exit;
      break;

exit:
   ...;  // Clean up and exit
```

880  C Outline API Functions
See Also

- `EssOtlGetLinkedAttributeAttachLevel`
- `EssOtlFreeStructure`

## EssOtlQueryGetFirstDimension

Returns the dimension handle of the first dimension in the outline.

### Syntax

#### Parameter | Data Type | Description
--- | --- | ---
`hOutline` | ESS_HOUTLINE_T | Handle to the outline (input).
`phDim` | ESS_PHMEMBER_T | The dimension handle (output).

### Notes

- Before you call this function, call `EssOtlOpenOutlineQuery` to open the outline in query mode.
- This function returns the dimension handle of the first dimension in the outline. The handle returned by this function can then be used to call `EssOtlGetDimensionNameUniqueness`, `EssOtlGetCountOfDupMemberNameInDim`, or `EssOtlIsMemberNameUniqueWithinDim`.

### Return Value

Returns 0 if successful; otherwise, returns an error.

### Example

```c
ESS_FUNC_M ESS_ISUniq()
{
    ESS_STS_T    sts = 0;
    ESS_HOUTLINE_T   hOutline;
    ESS_OBJDEF_T   Object;
    ESS_APPNAME_T   szAppName;
    ESS_DBNAME_T   szDbName;
    ESS_OBJNAME_T   szFileName;
    ESS_HMEMBER_T   hDim = ESS_NULL;
    ESS_BOOL_T    pbNameUnique = 0;

    memset(&Object, '\0', sizeof(Object));
    Object.hCtx =   hCtx;
    Object.ObjType =  ESS_OBJTYPE_OUTLINE;
```
strcpy(szAppName, "Demo");
strcpy(szDbName, "Test");
strcpy(szFileName, "Test");
Object.AppName = szAppName;
Object.DbName = szDbName;
Object.FileName = szFileName;

sts = EssOtlOpenOutlineQuery(hCtx, &Object, &hOutline);

if (!sts)
{
    sts = EssOtlQueryGetFirstDimension(hOutline, &hDim);
    if (sts)
        printf("EssOtlQueryGetFirstDimension failed sts %ld\n",sts);
}

if (!sts)
{
    sts = EssOtlIsMemberNameUniqueWithinDim(hOutline, hDim, &pbNameUnique);
    if (sts)
        printf("EssOtlIsMemberNameUniqueWithinDim failed sts %ld\n",sts);
    else
        printf("pbNameUnique is %d\n", pbNameUnique);
}

return sts;

See Also

- EssOtlQueryGetNextDimension
- EssOtlIsMemberNameUniqueWithinDim

EssOtlQueryGetNextDimension

Returns the next dimension handle of the dimension in the outline opened in query mode.

Syntax

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline;</td>
<td>ESS_HOUTLINE_T</td>
<td>Handle to the outline (input)</td>
</tr>
<tr>
<td>hDim;</td>
<td>ESS_HMEMBER_T</td>
<td>The dimension handle (input)</td>
</tr>
<tr>
<td>phNextDim;</td>
<td>ESS_PHMEMBER_T</td>
<td>The handle of the next dimension (output)</td>
</tr>
</tbody>
</table>
Notes

- Before you call this function, call EssOtlOpenOutlineQuery to open the outline in query mode.
- As shown in the example, you must call EssOtlQueryGetFirstDimension before you call this function. Otherwise, an error will be returned.
- If you pass in the handle of the dimension that appears last in the dimension, this function returns null.

Return Value

Returns 0 if successful; otherwise, returns an error.

Example

```c
ESS_FUNC_M ESS_ISUniqMemberNameWithinDimAtGenLev()
{
    ESS_STS_T    sts = 0;
    ESS_HOUTLINE_T   hOutline;
    ESS_OBJDEF_T   Object;
    ESS_APPNAME_T   szAppName;
    ESS_DBNAME_T   szDbName;
    ESS_OBJNAME_T   szFileName;
    ESS_HMEMBER_T   hDim,hNextDim;
    ESS_BOOL_T    pbNameUnique, bGen = ESS_TRUE;
    ESS_USHORT_T   usGenLevel = 3;

    memset(&Object, '\0', sizeof(Object));
    Object.hCtx =   hCtx;
    Object.ObjType =  ESS_OBJTYPE_OUTLINE;
    strcpy(szAppName, "Demo");
    strcpy(szDbName, "Test");
    strcpy(szFileName, "Test");
    Object.AppName =  szAppName;
    Object.DbName =   szDbName;
    Object.FileName =  szFileName;

    sts = EssOtlOpenOutlineQuery (hCtx, &Object, &hOutline);

    if (!sts)
    {
        sts = EssOtlQueryGetFirstDimension(hOutline, &hDim);

        if (sts)
            printf("EssOtlQueryGetFirstDimension failed sts %ld\n",sts);
    }

    if (!sts)
    {
        sts = EssOtlIsMemberNameUniqueWithinDimAtGenLevel (hOutline, hDim, bGen, usGenLevel,
 &pbNameUnique);
        if (sts)
```
printf("EssOtlIsMemberNameUniqueWithinDimAtGenLevel failed sts %ld\n",sts);
else
    printf("pbNameUnique is %d\n", pbNameUnique);
}

if (!sts)
{
    
sts = EssOtlQueryGetNextDimension (hOutline, hDim, &hNextDim);
    
    if (sts)
        printf("EssOtlQueryGetFirstDimension failed sts %ld\n",sts);
}

if (!sts)
{
    
sts = EssOtlIsMemberNameUniqueWithinDimAtGenLevel (hOutline, hNextDim, bGen,
        usGenLevel, &pbNameUnique);
    
    if (sts)
        printf("EssOtlIsMemberNameUniqueWithinDimAtGenLevel failed sts %ld\n",sts);
    else
        printf("pbNameUnique is %d\n", pbNameUnique);
}

return sts;
}

See Also

- EssOtlQueryGetFirstDimension

**EssOtlQueryMembers**

Queries the outline.

**Syntax**

```c
ESS_FUNC_M EssOtlQueryMembers (hOutline, hMember, pPredicate, pMbrCounts, phMemberArray);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Essbase outline handle. This must have been returned from EssOtlOpenOutlineQuery.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>hMember</td>
<td>ESS_HMEMBER_T</td>
<td>The handle of the member on which execute the operation. If this value is NULL, it is assumed to be the very top of the outline, representing the logical parent of the dimensions. If the handle is a shared member, this function executes on the stored member on which it is based. This value will be ignored for the following options:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESS_NAMEDGENERATION</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESS_NAMEDLEVEL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESS_USERATTRIBUTE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESS_SEARCH</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESS_WILDSSEARCH</td>
</tr>
<tr>
<td>pPredicate</td>
<td>“ESS_PREDICATE_T” on page 709</td>
<td>Structure defining the query. The fields of this structure are described in Notes.</td>
</tr>
<tr>
<td>pMbrCounts</td>
<td>“ESS_MBCOUNTS_T” on page 699</td>
<td>Structure defining information about counts. It contains the following fields:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ulStart—Starting number to return</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ulMaxCount—Maximum number of member handles to return</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ulTotalCount—Total number of members that are defined in the results of the query</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● pulReturnCount—Number of member handles returned in this query</td>
</tr>
<tr>
<td>phMemberArray</td>
<td>ESS_PPHMEMBER_T</td>
<td>An array of member handles returned from the query.</td>
</tr>
</tbody>
</table>

**Notes**

- The call takes a member handle to operate on and returns an array of member handles satisfying the criteria specified by the option value.
- The caller should call EssOtlFreeMembers when the returned phMembers member array is no longer needed.
- Each hMember element in the array can be used only in calls that are listed in EssOtlOpenOutlineQuery. For example, a returned member handle cannot be used to call EssOtlGetSibling.
- The fields of the pPredicate structure are used as follows:
  - ulQuery—Value defining the operation to perform. It can be one of the following:
    - ESS_CHILDREN
    - ESS_DESCENDANTS
    - ESS_BOTTOMLEVEL
    - ESS_SIBLINGS
    - ESS_SAMELEVEL
    - ESS_SAMEGENERATION
- ESS_PARENT
- ESS_DIMENSION
- ESS_NAMEDGENERATION
- ESS_NAMEDLEVEL
- ESS_SEARCH
- ESS_WILDSEARCH
- ESS_USERATTRIBUTE
- ESS_ANCESTORS
- ESS_DTSMEMBERS

- ulOptions—Value defining search options. Valid values:
  - ESS_COUNTONLY—Returns no member handles, but fills in the pTotalCount field in the pCounts structure
  - ESS_NOTOTALCOUNTS
  - ESS_INCLUDEHYBRIDANALYSIS
  - ESS_EXCLUDEHYBRIDANALYSIS
  - ESS_FORCECASESENSITIVE
  - ESS_FORCEIGNORECASE

  When the Query type is set to ESS_SEARCH or ESS_WILDSEARCH, three additional values for Option are valid:
  - ESS_MEMBERSONLY
  - ESS_ALIASESONLY
  - ESS_MEMBERSANDALIASES

  To specify multiple values, use bitwise OR ( | ); for example:
  ESS_FORCECASESENSITIVE | ESS_MEMBERSONLY

- szDimension—Dimension to limit the scope of the query. It is used with the following query options and ignored otherwise:
  - ESS_NAMEDGENERATION
  - ESS_NAMEDLEVEL
  - ESS_USERATTRIBUTE
  - ESS_SEARCH—Set to NULL to search through all dimensions
  - ESS_WILDSEARCH—Set to NULL to search through all dimensions

- pszString1—
  Input string that is determined by the option. It is used with the following query options and ignored otherwise:
  - ESS_NAMEDGENERATION—Name of the generation
- **ESS_NAMEDLEVEL**—Name of the level
- **ESS_SEARCH**—String to search for. The string is defined as an exact match.
- **ESS_WILDSEARCH**—String to search for. The string is defined as an exact search string with an optional '*' at the end to mean any set of characters.
- **ESS_USERATTRIBUTE**—User defined attribute.
- **pszString2**—Input string that is determined by the option. It is used with the following query options and ignored otherwise:
  - **ESS_USERATTRIBUTE**—User defined attribute.
  - **ESS_SEARCH, ESS_WILDSEARCH**—If the options are set to look in the alias tables, this string specifies the alias table in which to search. If null, all alias tables are searched.

**Return Value**

The return value is zero if the function was successful.

**Example**

```c
#include <essapi.h>
#include <essotl.h>

ESS_STS_T          sts = ESS_STS_NOERR;
ESS_HOUTLINE_T     hOutline;
ESS_OBJDEF_T       Object;
ESS_HMEMBER_T      hMember = 0;
ESS_PREDICATE_T    Predicate;
ESS_MBRCOUNTS_T    Counts;
ESS_PHMEMBER_T     phMemberArray = ESS_NULL;
ESS_ULONG_T        i;
ESS_ACCESS_T       Access;
ESS_STR_T          AppName;
ESS_STR_T          DbName;

AppName = "Sample";
DbName = "Basic";

sts = EssSetActive(hCtx, AppName, DbName, &Access);
if (sts == 0)
{
    memset(&Object, '\0', sizeof(Object));
    sts = EssOtlOpenOutlineQuery(hCtx, &Object, &hOutline);
    memset(&Predicate, '\0', sizeof(Predicate));
    Predicate.ulQuery           = ESS_CHILDREN;
    Predicate.pszDimension = "Year";
    memset(&Counts, '\0', sizeof(Counts));
    Counts.ulStart          = 0;
    Counts.ulMaxCount = 10;
    if(!sts)
```
EssOtlQueryMembers

Queries the outline.

Syntax

```
ESS_FUNC_M EssOtlQueryMembersByName (hOutline, pszMember, pPredicate, pMbrCounts, phMemberArray);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline context handle. This must have been returned from EssOtlOpenOutlineQuery().</td>
</tr>
<tr>
<td>pszMember</td>
<td>ESS_STR_T</td>
<td>The member name string of the member to do the operation on. If this value is NULL, it is assumed to be the very top of the outline,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>representing the logical parent of the dimensions. This value will be ignored for the following options:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESS_NAMEDGENERATION</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESS_NAMEDLEVEL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESS_USERATTRIBUTE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESS_SEARCH</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESS_WILDSERCH</td>
</tr>
<tr>
<td>pPredicate</td>
<td>“ESS_PREDICATE_T”</td>
<td>Structure defining the query. The fields of this structure are described in Notes.</td>
</tr>
<tr>
<td></td>
<td>on page 709</td>
<td></td>
</tr>
</tbody>
</table>

See Also

- EssOtlFreeMembers
- EssOtlGetDimensionUserAttributes
- EssOtlOpenOutlineQuery
- EssOtlQueryMembersByName
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pMbrCounts</td>
<td>“ESS_MBCOUNTS_T” on page 699</td>
<td>Structure defining information about member counts. It contains the following fields:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ulStart—Starting number to return</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ulMaxCount—Maximum number of member handles to return.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ulTotalCount—Total number of members that are defined in the results of the query.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- pulReturnCount—Number of member handles returned in this query.</td>
</tr>
<tr>
<td>phMemberArray</td>
<td>ESS_PPHMEMBER_T</td>
<td>An array of member handles returned from the query.</td>
</tr>
</tbody>
</table>

Notes

- The call takes a member name string to operate on and returns an array of member handles satisfying the criteria specified by the option value.
- The caller should call `EssOtlFreeMembers()` when the returned phMembers member array is no longer needed.
- Each hMember element in the array can only be used in calls that are listed in `EssOtlOpenOutlineQuery()`. For example, a returned member handle cannot be used to call `EssOtlGetSibling()`.
- The fields of the `pPredicate` structure are used as follows:
  - `ulQuery`—Value defining the operation to perform. It can be one of the following:
    - ESS_CHILDREN
    - ESS_DESCENDANTS
    - ESS_BOTTOMLEVEL
    - ESS_SIBLINGS
    - ESS_SAMELEVEL
    - ESS_SAMEGENERATION
    - ESS_PARENT
    - ESS_DIMENSION
    - ESS_NAMEDGENERATION
    - ESS_NAMEDLEVEL
    - ESS_SEARCH
    - ESS_WILDSEARCH
    - ESS_USERATTRIBUTE
    - ESS_ANCESTORS
    - ESS_DTMEMBERS
  - `ulOptions`—Value defining search options. Valid values:
- ESS_COUNTONLY—Returns no member handles, but only fills in the pTotalCount field in the pCounts structure
- ESS_NOTOTALCOUNTS
- ESS_INCLUDEHYBRIDANALYSIS
- ESS_EXCLUDEHYBRIDANALYSIS
- ESS_FORCECASESENSITIVE
- ESS_FORCEIGNORECASE

When the Query type is set to ESS_SEARCH or ESS_WILDSEARCH three additional values for Option are valid:
- ESS_MEMBERSONLY
- ESS_ALIASESONLY
- ESS_MEMBERSANDALIASES

To specify multiple values, use bitwise OR ( | ); for example:

```c
ESS_FORCECASESENSITIVE | ESS_MEMBERSONLY
```

- **pszString1**—Input string that is determined by the option. It is used with the following query options and ignored otherwise:
  - ESS_NAMEDGENERATION—Name of the generation
  - ESS_NAMEDLEVEL—Name of the level
  - ESS_SEARCH—String to search for. The string is defined as an exact search string with an optional '*' at the end to mean any set of characters.
  - ESS_WILDSEARCH—String to search for. The string is defined as an exact search string with an optional '*' at the end to mean any set of characters.
  - ESS_USERATTRIBUTE—User defined attribute

- **pszString2**—Input string that is determined by the option. It is used with the following query options and ignored otherwise:
  - ESS_USERATTRIBUTE—User defined attribute.
  - ESS_SEARCH, ESS_WILDSEARCH—If the options are set to look in the alias tables, this string specifies the alias table to search in. If it’s null, all alias tables will be searched.

**Return Value**

The return value is zero if the function was successful.

**Example**

```c
#include <essapi.h>
#include <essotl.h>

ESS_STS_T          sts = ESS_STS_NOERR;
ESS_HOUTLINE_T     hOutline;
ESS_OBJDEF_T       Object;
ESS_STR_T          pszMember;
```
pszMember = "Qtr1";
AppName = "Sample";
DbName = "Basic";

sts = EssSetActive(hCtx, AppName, DbName, &Access);

if (sts == 0)
{
    memset(&Object, '\0', sizeof(Object));
    sts = EssOtlOpenOutlineQuery(hCtx, &Object, &hOutline);

    memset(&Predicate, '\0', sizeof(Predicate));
    Predicate.ulQuery = ESS_CHILDREN;
    Predicate.pszDimension = "Year";

    memset(&Counts, '\0', sizeof(Counts));
    Counts.ulStart = 0;
    Counts.ulMaxCount = 10;

    if(!sts)
    {
        sts = EssOtlQueryMembersByName(hOutline, pszMember, &Predicate, &Counts, &phMemberArray);

        if(!sts && Counts.ulReturnCount)
        {
            sts = EssOtlFreeMembers(hOutline, Counts.ulReturnCount, phMemberArray);
        }
    }
}

See Also

- EssOtlFreeMembers
- EssOtlGetDimensionUserAttributes
- EssOtlOpenOutlineQuery
- EssOtlQueryMembers

### EssOtlQueryMembersEx

Queries the outline for specific members and member fields, and returns an array of member handles. The returned member handles can be used with other Outline API functions such as EssOtlGetMemberInfo(). (EssOtlGetMemberInfo() can retrieve any of the individual fields contained in "ESS_MEMBERINFO_T" on page 149 and "ESS_MBRINFO_T" on page 699.)
### Syntax

ESS_FUNC_M **EssOtlQueryMembersEx** (hOutline, pszFieldSelection, pszMemberSelection, pMaxCount, ppMemberArray, ppqryErrorList)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Essbase outline handle. This must have been returned from <strong>EssOtlOpenOutlineQuery()</strong>.</td>
</tr>
<tr>
<td>pszFieldSelection</td>
<td>ESS_STR_T</td>
<td>The query string which defines the set of fields that will be returned for each member. The syntax of pszFieldSelection is shown in Notes.</td>
</tr>
<tr>
<td>pszMemberSelection</td>
<td>ESS_STR_T</td>
<td>The query string which defines the set of members to be returned. The syntax of this query string is the syntax for member selection; that is, the query string can be anything that you can use in a FIX() statement.</td>
</tr>
<tr>
<td>pMaxCount</td>
<td>ESS_PULONG_T</td>
<td>Input: A pointer to the maximum number of member handles to be returned. Output: A pointer to the number of member handles returned.</td>
</tr>
<tr>
<td>ppMemberArray</td>
<td>ESS_PPHMEMBER_T</td>
<td>Reference to a pointer to the first in an array of member handles returned.</td>
</tr>
<tr>
<td>ppqryErrorList</td>
<td>“ESS_OTLQUERYERRORLIST_T” on page 703</td>
<td>Reference to a pointer to a structure containing the list of errors in the query.</td>
</tr>
</tbody>
</table>

### Notes

- In an outline that allows duplicate member names, this function returns the fully qualified names of shared members. For example, in Sample Basic, any query that includes the shared member `100-20` would return its fully qualified name, `[Diet].[100-20]`.
- Use of *UniqueName* as part of the member fields selection automatically includes *ShareOption* as part of the field selection.
- **EssOtlQueryMemberEx()** takes an outline handle and returns an array of member handles specified by `pszMemberSelection`.
- The caller should call **EssOtlFreeMembers()** when the returned `pphMembers` member array is no longer needed.
- Each member handle element of the array can only be used in calls that are listed in **EssOtlOpenOutlineQuery()**. For example, a returned member handle cannot be used to call **EssOtlGetSibling()**.
- The syntax of `pszFieldSelection` is the following:

  ```
  QueryString ==: <SelectMbrInfo (FieldName {, FieldName}, ... )
  where FieldName is one of the following:
  
  MemberName /* Member name */
  MemberLevel /* Member level number */
  MemberGeneration /* Member generation number */
  Cosolidation /* Whether this member is consolidated */
  ```
TwoPass  //@ Whether this member undergoes a two pass operation
/* 
Expense  //@ Whether this is an expense member */
CurrencyConvType  //@ Currency conversion type */
CurrencyMember  //@ Whether this is a currency member */
TimeBalance  //@ Time balance measure */
SkipOption  //@ Whether this member skips the time balance
operation */
ShareOption  //@ Whether this is a shared member*/
StorageType  //@ Dimension's storage type */
DimensionCategory  //@ Dimension category: accounts, time, currency, etc.
*/
DimensionStorageCategory  //@ Dimension storage category: time, units, scenario,
etc. */
Comment  //@ Member comment */
ChildrenCount  //@ Number of children */
MemberNumber  //@ Member number */
DimensionName  //@ Dimension name */
DimensionNumber  //@ Dimension number */
MemberAliasName  //@ Alias for this member */
ParentMemberName  //@ Parent's name */
ChildMemberName  //@ Child's name */
PreviousMemberName  //@ Left sibling's name */
NextMemberName  //@ Right sibling's name */
CurrencyConversionDatabase  //@ Whether this database has currency conversion */
MemberStatus  //@ Member status */
UDAList  //@ List of UDAs attached to this member */
MemberFormula  //@ Formula for this member */
MemberValidity  //@ Whether this member is valid */
Attributes  //@ All attribute fields. If the member is not
attributed, then attribute name is set to NULL */
UniqueName  //@ If the member is duplicate, its fully qualified,
unique name. */

Note: There is no leading '<' character for the individual fieldnames.

- To use this function with EssOtlGetMemberField(), include in this function's
pszFieldSelection string the same fields that you will specify using the MbrFieldID
constants of EssOtlGetMemberField(). Otherwise, EssOtlGetMemberField() returns the error
OTLAPI_ERR_MBRINVALID.

Return Value

The return value is zero if the function was successful.

Example

The following code snippet return the name, consolidation and formula for each member
which is a child of Market or a child of Product. Upon return, MaxCount contains the number
of members returned, and phMemberArray contains the array of handles for the set of members
returned. Further Outline API calls allow interrogation of the members using the returned array
of member handles in phMemberArray.

member_fields = "<SelectMbrInfo ( MemberName, Consolidation, MemberFormula ) ";
member_selection = "@ichild(Product), @ichild(Market)"
MaxCount = -1;
phMemberArray = ESS_NULL;
pqryErrorList = ESS_NULL;

sts = EssOtlQueryMembersEx(hOutline,
    member_fields,
    member_selection,
    &MaxCount,
    &phMemberArray,
    &pqryErrorList);

if (sts != 0) goto error_exit;

See “Extended Member Query Code Example” on page 982 for an example that uses
EssOtlQueryMembersEx(), EssOtlGetMemberField(), and ESS_OTLQUERYERRORLIST_T,
and includes calls to EssOtlFreeMembers() and EssFree().

See Also
● EssOtlFreeMembers
● EssOtlGetDimensionUserAttributes
● EssOtlGetMemberField
● EssOtlOpenOutlineQuery
● EssOtlQueryMembers
● EssOtlQueryMembersByName

**EssOtlQueryMembersExArray**

Queries the outline for specific members and member fields, and returns an array of member handles. The returned member handles can be used with other Outline API functions such as
EssOtlGetMemberInfo(). (EssOtlGetMemberInfo() can retrieve any of the individual fields
contained in “ESS_MEMBERINFO_T” on page 149 and “ESS_MBRINFO_T” on page 699.)

**Syntax**

ESS_FUNC_M EssOtlQueryMembersExArray (hOutline, pszFieldSelection, queryCount, 
pszMemberSelectionArr, pMaxCountArr, pphMemberArr, ppqryErrorList)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline handle returned from EssOtlOpenOutlineQuery().</td>
</tr>
<tr>
<td>pszFieldSelection</td>
<td>ESS_STR_T</td>
<td>Selects the member fields that the queries return. The same selections are used for all queries in the array.</td>
</tr>
<tr>
<td>queryCount</td>
<td>ESS_SHORT_T</td>
<td>Count of members in the input array.</td>
</tr>
<tr>
<td>pszMemberSelectionArr</td>
<td>ESS_STR_T</td>
<td>Array of queryCount query strings for member selection. The syntax of this query string is the syntax for member selection; that is, the query string can be anything that you can use in a FIX() statement.</td>
</tr>
<tr>
<td>pMaxCountArr</td>
<td>ESS_PULONG_T</td>
<td>Array of queryCount values for how many members each query in the array at most should return. Each value is replaced with the actual returned count.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>pphMemberArr</td>
<td>ESS_PPHMEMBER_T</td>
<td>queryCount array of returned member handle arrays (each with pMaxCountArr[i] values).</td>
</tr>
<tr>
<td>ppqryErrorList</td>
<td>“ESS_OTLQUERYERRORLIST_T”</td>
<td>List of members with errors.</td>
</tr>
</tbody>
</table>

**Notes**

- In an outline that allows duplicate member names, this function returns the fully qualified names of shared members. For example, in Sample Basic, any query that includes the shared member 100-20 would return its fully qualified name, [Diet].[100-20].
- Use of **UniqueName** as part of the member fields selection automatically includes **ShareOption** as part of the field selection.
- **EssOtlQueryMemberExArray()** takes an outline handle and returns an array of member handles specified by **pszMemberSelection**.
- The caller should call **EssOtlFreeMembers()** when the returned **pphMembers** member array is no longer needed.
- Each member handle element of the array can only be used in calls that are listed in **EssOtlOpenOutlineQuery()**. For example, a returned member handle cannot be used to call **EssOtlGetSibling()**.
- The syntax of **pszFieldSelection** is the following:

  QueryString ==: <SelectMbrInfo ( FieldName {, FieldName}, ... )

  where **FieldName** is one of the following:

  - **MemberName** /* Member name */
  - **MemberLevel** /* Member level number */
  - **MemberGeneration** /* Member generation number */
  - **Consolidation** /* Whether this member is consolidated */
  - **TwoPass** /* Whether this member undergoes a two pass operation */
  - **Expense** /* Whether this is an expense member */
  - **CurrencyConvType** /* Currency conversion type */
  - **CurrencyMember** /* Whether this is a currency member */
  - **TimeBalance** /* Time balance measure */
  - **SkipOption** /* Whether this member skips the time balance operation */
  - **ShareOption** /* Whether this is a shared member */
  - **StorageType** /* Dimension's storage type */
  - **DimensionCategory** /* Dimension category: accounts, time, currency, etc. */
  - **DimensionStorageCategory** /* Dimension storage category: time, units, scenario, etc. */
  - **Comment** /* Member comment */
  - **ChildrenCount** /* Number of children */
  - **MemberNumber** /* Member number */
  - **DimensionName** /* Dimension name */
  - **DimensionNumber** /* Dimension number */
  - **MemberAliasName** /* Alias for this member */
  - **ParentMemberName** /* Parent’s name */

C Outline API Function Reference  895
ChildMemberName /* Child's name */
PreviousMemberName /* Left sibling's name */
NextMemberName /* Right sibling's name */
CurrencyConversionDatabase /* Whether this database has currency conversion */
MemberStatus /* Member status */
UDAList /* List of UDAs attached to this member */
MemberFormula /* Formula for this member */
MemberValidity /* Whether this member is valid */
Attributes /* All attribute fields. If the member is not attributed, then attribute name is set to NULL */
UniqueName /* If the member is duplicate, its fully qualified, unique name. */

**Note:** There is no leading '<' character for the individual fieldnames.

**Return Value**

Returns zero (0) if successful.

**Example**

The following code snippet returns the name, consolidation and formula for each member that is a child of Market and for each member that is a child of Product in two separate member arrays. It combines what would have been two queries in EssOtIQueryMembersEx into just one call to EssOtIQueryMembersExArray. Note that the member fields returned will be the same for all queries in the array, and that the size of all arrays must match `queryCount`.

Upon return, `MaxCountArray[i]` contains the number of members returned in each query, and `phMemberArrayArray[i]` contains the array of handles for the set of members returned for each query. Further Outline API calls allow interrogation of the members using the returned array of member handles in `phMemberArrayArray[i]`.

```c
member_fields = "<SelectMbrInfo ( MemberName, Consolidation, MemberFormula ) ";
queryCount = 2;
member_selectionArray[0] = "@ichild(Product)";
member_selectionArray[1] = "@ichild(Market)";
MaxCountArray[0] = -1;
MaxCountArray[1] = -1;
phMemberArrayArray[0] = ESS_NULL;
phMemberArrayArray[1] = ESS_NULL;
pqryErrorListArray[0] = ESS_NULL;
pqryErrorListArray[1] = ESS_NULL;

sts = EssOtIQueryMembersExArray(hOutline, member_fields, queryCount,
member_selectionArray, MaxCountArray, &phMemberArrayArray, pqryErrorListArray);

if (sts != 0) goto error_exit;
```

See “Extended Member Query Code Example” on page 982 for an example that uses `EssOtIQueryMembersEx()`, `EssOtIGetMemberField()`, and `ESS_OTLQUERYERRORLIST_T`, and includes calls to `EssOtIFreeMembers()` and `EssFree()`.

**See Also**

- `EssOtIFreeMembers`
- `EssOtIGetDimensionUserAttributes`
● EssOtlGetMemberField
● EssOtlOpenOutlineQuery
● EssOtlQueryMembers
● EssOtlQueryMembersByName

EssOtlQueryObjects

Returns an array of object handles of the specified type for the input object names, or if the *pcount is zero, then it returns all the object handles.

Syntax

ESS_FUNC_M EssOtlQueryObjects(hOutline, objType, objNames, pcount, ppObjHandles)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline handle (Query mode only)</td>
</tr>
<tr>
<td>objType</td>
<td>ESS_OBJECT_TYPES</td>
<td>Object type:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● OBJECT_SMARTLIST</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Object type is Text List (SmartList)</td>
</tr>
<tr>
<td>objNames</td>
<td>ESS_PSTR_T</td>
<td>Array of object names to be queried</td>
</tr>
<tr>
<td>pcount</td>
<td>ESS_PULONG_T</td>
<td>Count of object names. If pcount is zero, this contains the number of Text List (SmartList) handles on execution.</td>
</tr>
</tbody>
</table>
| ppObjHandles  | ESS_PPHOBJECT_T | Array of object handles
|               |                 | This must be de allocated using EssOtlFreeObjectArray                       |

Return Value

Returns:

● 0—If successful
   pcount contains a value.

● Error number—If unsuccessful
   pcount is NULL.

Example

void TestFreeObjectArray()
{
    ESS_STS_T                    sts = ESS_STS_NOERR;
    ESS_HOUTLINE_T              hOutline = ESS_NULL;
    ESS_OBJDEF_T                Object;
    ESS_STR_T                   objNames[1];
    ESS_OBJECT_TYPES            objType;
    ESS ULONG_T                 count;
    ESS PHOBJECT_T              hObjiHandles = ESS_NULL;

    memset(&Object, '\0', sizeof(Object));
Object.hCtx = hCtx;
Object.ObjType = ESS_OBJTYPE_OUTLINE;
Object.AppName = szAppName;
Object.DbName = szDbName;
Object.FileName = szFileName;

/* Set up */
sts = EssOtlOpenOutlineQuery(hCtx, &Object, &hOutline);
count = 2;
objType = OBJECT_SMARTLIST;
objNames[0] = "Smartlist1";
objNames[1] = "Smartlist2";

/* Query objects */
sts = EssOtlQueryObjects(hOutline, objType,
                         objNames, &Count, &hObjHandles);

/* Free object array */
if(hObjHandles)
{
    sts = EssOtlFreeObjectArray(hOutline, count, hObjHandles);
}

/* Close outline */
sts = EssOtlCloseOutline(hOutline);

See Also

- EssOtlGetMemberSmartList
- EssOtlCreateObject
- EssOtlDeleteObject
- EssOtlGetSmartListInfo
- EssOtlFindObject
- EssOtlFreeObjectArray
- EssOtlFreeSmartListInfo
- EssOtlGetMemberSmartList
- EssOtlGetMemberType
- EssOtlGetObjectReferenceCount
- EssOtlGetObjectReferences
- EssOtlImportExportObject
- EssOtlListObjects
- EssOtlQueryObjects
- EssOtlSetMemberType
- EssOtlSetMemberTypeToSmartList

EssOtlQueryVaryingAttributes

Queries member information for a given attribute member or function, enabling specification of the perspective for varying attributes.
Syntax

```c
ESS_FUNC_M EssOtlQueryVaryingAttributes (hOutline, pAttrQuery, pPerspective, pCount, pphMembers);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline context handle.</td>
</tr>
<tr>
<td>pAttrQuery</td>
<td>ESS_PATTRIBUTEQUERY_T</td>
<td>Pointer to the structure that defines the query.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- If pAttrQuery.bInputMemberIsHandle = ESS_TRUE, make sure pAttrQuery.uInputMember.hMember is assigned</td>
</tr>
<tr>
<td></td>
<td></td>
<td>a handle to a member.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- If pAttrQuery.bInputMemberIsHandle = ESS_FALSE, make sure pAttrQuery.uInputMember.szMember is assigned</td>
</tr>
<tr>
<td></td>
<td></td>
<td>a member name.</td>
</tr>
<tr>
<td>pPerspective</td>
<td>ESS_PPERSiSTIVE_T</td>
<td>Pointer to a collection of independent members used when querying the client or server for associations.</td>
</tr>
<tr>
<td>pCount</td>
<td>ESS_PMBRCOUNTS_T</td>
<td>Pointer to the number of base members returned.</td>
</tr>
<tr>
<td>pphMembers</td>
<td>ESS_PPHMEMBER_T</td>
<td>Pointer to the array of attribute member handles.</td>
</tr>
</tbody>
</table>

Notes

Similar to `EssOtlQueryAttributesEx`, this function performs an attribute query. When the query involves an input base member and output attribute members, or an input attribute member and output base members, the given perspective is used to restrict the results based on the associations that are valid in the perspective.

The structure `ESS_VARYING_ATTRIBUTEQUERY_T` is identical to `ESS_ATTRIBUTEQUERY_T`, except that the varying version contains a field for the attribute dimension.

Note that the perspective must specify discrete independent members individually.

If no perspective is specified, or if the perspective specifies a NULL set of independent members, the routine will consider all associations that exist for any combination of independent members. In this case, the returned validity sets may contain ranges for discrete independent members, and it is the responsibility of the client to split this accordingly.

Return Value

Returns 0 if successful.

Example

```c
void TestEssOtlQueryVaryingAttributes()
{
    ESS_STS_T    sts = ESS_STS_NOERR;
    ESS_HOUTLINE_T  hOutline = ESS_NULL;
    ESS_OBJDEF_T   Object;
    ESS_USHORT_T   i = 0;
    ESS_HMEMBER_T  hBaseMbr = ESS_NULL;
}  ```
memset(&Object, '\0', sizeof(Object));
Object.hCtx = hCtx;
Object.ObjType = ESS_OBJTYPE_OUTLINE;
Object.AppName = szAppName;
Object.DbName = szDbName;
Object.FileName = szDbName;

printf("\n");
sts = EssOtlOpenOutlineQuery(hCtx, &Object, &hOutline);
printf("EssOtlOpenOutlineQuery sts: %ld\n",sts);

memset(&Counts, '\0', sizeof(Counts));
Counts.ulStart = 0;
Counts.ulMaxCount = 0;

/* Get handles for independent members */
memset(&Predicate, '\0', sizeof(Predicate));
Predicate.ulQuery = ESS_SEARCH;
Predicate.ulOptions = ESS_MEMBERSONLY;
Predicate pszDimension = "";
Predicate pszString1 = "Jan";
Predicate pszString2 = "";
sts = EssOtlQueryMembersByName(hOutline, ESS_NULL, &Predicate, &Counts, &phMbrHandles);
hIndepMbrHandlesArray[0] = phMbrHandles[0];
hIndepMbrHandlesArray[2] = phMbrHandles[0];

Predicate pszString1 = "FY03";
sts = EssOtlQueryMembersByName(hOutline, ESS_NULL, &Predicate, &Counts, &phMbrHandles);
hIndepMbrHandlesArray[1] = phMbrHandles[0];
Predicate pszString1 = "FY04";
sts = EssOtlQueryMembersByName(hOutline, ESS_NULL, &Predicate, &Counts, &phMbrHandles);
hIndepMbrHandlesArray[3] = phMbrHandles[0];

/* Get handles for attribute member and dimension */
Predicate pszString1 = "Type";
sts = EssOtlQueryMembersByName(hOutline, ESS_NULL, &Predicate, &Counts, &phMbrHandles);
hAttrDim = phMbrHandles[0];
Predicate pszString1 = "Contractor";
sts = EssOtlQueryMembersByName(hOutline, ESS_NULL, &Predicate, &Counts, &phMbrHandles);
hAttrMbr = phMbrHandles[0];
memset(&Perspective, '\0', sizeof(ESS_PERSPECTIVE_T));
Perspective.usValiditySetType = ESS_VALIDITYSET_TYPE_MBRHDLS;
Perspective.countOfIndepDims = 2;
Perspective.countOfIndepRanges = 1;
Perspective.pIndepMbrs = hIndepMbrHandlesArray;
/* Query by handle with InputMemberType of ESS_ATTRIBUTE_MEMBER and OutputMemberType of ESS_BASE_MEMBER*/
printf("*** Query by handle with InputMemberType of ESS_ATTRIBUTE_MEMBER and OutputMemberType of ESS_BASE_MEMBER:\n");
memset(&pAttrQuery, '\0', sizeof(ESS_ATTRIBUTEQUERY_T));
pAttrQuery.bInputMemberIsHandle = ESS_TRUE;
pAttrQuery.uInputMember.hMember = hAttrMbr;
pAttrQuery.uAttributeDimension.hMember = hAttrDim;
pAttrQuery.usInputMemberType = ESS_ATTRIBUTE_MEMBER;
pAttrQuery.usOutputMemberType = ESS_BASE_MEMBER;
pAttrQuery.Attribute.usDataType = ESS_ATTRMBRDTYPE_NONE;
pAttrQuery.usOperation = ESS_ALL;
sts = EssOtlQueryVaryingAttributes(hOutline, &pAttrQuery, &Perspective, &Counts, &phMbrHandles);
printf("EssOtlQueryVaryingAttributes sts: %d\n", sts);
if (!sts)
{
    if(phMbrHandles)
    {
        GetMemberInfo(hOutline, Counts, phMbrHandles);
        if(Counts.ulReturnCount && phMbrHandles)
            sts = EssOtlFreeMembers(hOutline, Counts.ulReturnCount, phMbrHandles);
    }
    else
        printf("\tNo member returned.\n");
}

sts = EssUnlockObject(hCtx, ESS_OBJTYPE_OUTLINE, Object.AppName, Object.DbName, Object.FileName);
printf("\nEssUnlockObject sts: %d\n", sts);

sts = EssOtlCloseOutline(hOutline);
printf("EssOtlCloseOutline sts: %d\n",sts);
}

See Also

- EssOtlVaryingAssociateAttribute
- EssOtlVaryingAssociateAttributeDimension
- EssOtlVaryingDisassociateAttribute
- EssOtlVaryingGetAssociatedAttributes
- EssOtlVaryingGetAttributeIndepDims

**EssOtlRenameAliasTable**

Renames an existing alias table.
Syntax

ESS_FUNC_M EssOtlRenameAliasTable (hOutline, pszAliasTable, pszNewAliasTable);

Parameter | Data Type          | Description
---        | ---                | ---

hOutline   | ESS_HOUTLINE_T    | Outline context handle.
pszAliasTable | ESS_STR_T        | Name of alias table to rename.
pszNewAliasTable | ESS_STR_T        | New name for alias table.

Notes

- The default alias table cannot be renamed from "Default".
- When renaming an alias table, language codes associated with the alias table are preserved in the renamed alias table.

Return Value

Returns 0 if successful; otherwise one of the following:

- OTLAPI_BAD_ALIASTABLE
- OTLAPI_ERR_RENAMEDEFALIAS
- OTLAPI_ERR_ALIASTABLENAME
- OTLAPI_ERR_ALIASTABLEEXISTS

Example

```c
#include <essapi.h>
#include <essotl.h>

ESS_STS_T          sts = 0;
ESS_HOUTLINE_T     hOutline;
ESS_OBJDEF_T       Object;
ESS_APPNAME_T      szAppName;
ESS_DBNAME_T       szDbName;
ESS_OBJNAME_T      szFileName;
memset(&Object, '\0', sizeof(Object));
Object.hCtx = hCtx;
Object.ObjType = ESS_OBJTYPE_OUTLINE;
strcpy(szAppName, "Sample");
strcpy(szDbName, "Basic");
strcpy(szFileName, "Basic");
Object.AppName = szAppName;
Object.DbName = szDbName;
Object.FileName = szFileName;

sts = EssOtlOpenOutline(hCtx, &Object, ESS_TRUE,
                       ESS_TRUE, &hOutline);

if (!sts)
{
    sts = EssOtlRenameAliasTable(hOutline,
```

902  C Outline API Functions
EssOtlRenameMember

Renames a member.

**Syntax**

ESS_FUNC_M EssOtlRenameMember (hOutline, hMember, pszNewMember);

**Parameter** | **Data Type** | **Description**
--- | --- | ---
 hOutline | ESS_HOUTLINE_T | Outline context handle
 hMember | ESS_HMEMBER_T | Handle of member to rename
 pszNewMember | ESS_STR_T | New member name

**Notes**

- All shared members are also renamed.
- This call fails if the *hMember* parameter points to a shared member.
- Renaming a zero-level (leaf node) attribute member that is not of type ESS_ATTRMBRDT_STRING resets the following:
  - the attribute value
  - the member's long name, using the specifications for the outline in the "ESS_ATTRSPECS_T" on page 119 structure
- Renaming an ancestor may affect the long name of a zero-level attribute member.

**Return Value**

Returns 0 if successful; otherwise one of the following:

- OTLAPI_BAD_MBRNAME
- OTLAPI_BAD_RENAMESHARE
- OTLAPI_ERR_RENAMENAMEUSED

**Example**

```c
#include <essapi.h>
#include <essotl.h>

ESS_STS_T          sts = 0;
```
NULL
hCtx = hCtx;
Object.ObjType = ESS_OBJTYPE_OUTLINE;
strcpy(szAppName, "Sample");
strcpy(szDbName, "Basic");
strcpy(szFileName, "Basic");
Object.AppName = szAppName;
Object.DbName = szDbName;
Object.FileName = szFileName;

sts = EssOtlOpenOutline(hCtx, &Object, ESS_TRUE,
                       ESS_TRUE, &hOutline);

if (!sts)
{
    sts = EssOtlFindMember(hOutline, "Jan",
                           &hMemberJan);
}

if (!sts && hMemberJan)
{
    sts = EssOtlRenameMember(hOutline, hMemberJan,
                              "January prelim");
}

See Also
- EssOtlFindMember
- EssOtlMoveMember
- EssOtlAddMember
- EssOtlDeleteMember

**EssOtlRestructure**

Restructures an outline on the server. This is an asynchronous call.

**Syntax**

ESS_FUNC_M EssOtlRestructure (hCtx, usRestructType);

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>Server login context handle. This must be the server on which the outline was saved using EssOtlWriteOutline().</td>
</tr>
</tbody>
</table>
Parameter | Data Type | Description
--- | --- | ---
usRestructType | ESS_USHORT_T | Type of restructuring to do. This can be one of the following values:
- ESS_DOR_ALLDATA
- ESS_DOR_INDATA
- ESS_DOR_LOWDATA
- ESS_DOR_NODATA
- ESS_DOR_FORCE_ALLDATA

Notes
- You must save the outline using `EssOtlWriteOutline()` before calling this function.
- This call is valid only for outlines saved to the server.
- This call is asynchronous. You should call `EssGetProcessState()` after making this call until `EssGetProcessState()` returns a status indicating the restructure operation is complete.
- In order for data to be properly restructured (saving data), the outline must have been opened using `EssOtlOpenOutline()` with the `fKeepTrans` flag set to ESS_TRUE.

Return Value
Returns 0 if successful; otherwise returns OTLAPI_BAD_RESTRUCTTYPE structure.

Access
This function requires you to have the appropriate level of access to the specified application and/or database to contain the outline object. To restructure the outline object, you must have Application Designer or Database Designer privilege (ESS_PRIV_APPDESIGN or ESS_PRIV_DBDESIGN) for the specified application or database containing the outline.

Example
```c
#include <essapi.h>
#include <essotl.h>

ESS_STS_T          sts = 0;
ESS_HCTX_T         hCtx;
ESS_HOUTLINE_T     hOutline;
ESS_OBJDEF_T       Object;
ESS_APPNAME_T      szAppName;
ESS_DBNAME_T       szDbName;
ESS_OBJNAME_T      szFileName;
memset(&Object, '\0', sizeof(Object));
Object.hCtx = hCtx;
Object.ObjType = ESS_OBJTYPE_OUTLINE;
strcpy(szAppName, "Sample");
strcpy(szDbName, "Basic");
strcpy(szFileName, "Basic");
Object.AppName = szAppName;
Object.DbName = szDbName;
Object.FileName = szFileName;

sts = EssOtlOpenOutline(hCtx, &Object, ESS_TRUE,
```
ESS_TRUE, &hOutline);

/* body of code */
/* write outline to server using */
/* EssOtlWriteOutline() */

if (!sts)
{
    sts = EssOtlRestructure(hCtx, ESS_DOR_ALLDATA);
}

/* need to call EssGetProcessState() */
/* to check for completion before proceeding */

See Also

- EssOtlOpenOutline
- EssOtlNewOutline
- EssOtlWriteOutline
- EssOtlVerifyOutline
- EssOtlCloseOutline

**EssOtlSetAggLevelUsage**

Applies view selection properties to stored hierarchies.

**Syntax**

ESS_FUNC_M EssOtlSetAggLevelUsage (hOutline, hMember, sAgglevelUsage);

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline context handle (input)</td>
</tr>
<tr>
<td>hMember</td>
<td>ESS_HMEMBER_T</td>
<td>A hierarchy member (input).</td>
</tr>
</tbody>
</table>

Value

0
### Parameter Data Type Description

- **sAgglevelUsage**  ESS_SHORT_T  One of the Level Usage Constants (input).

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_AGGLEVELUSAGE_DEFAULT</td>
<td>11</td>
<td>On primary hierarchies, consider all levels. Do not aggregate secondary</td>
</tr>
<tr>
<td></td>
<td></td>
<td>hierarchies unless alternate rollups are enabled.</td>
</tr>
<tr>
<td>ESS_AGGLEVELUSAGE_ALL</td>
<td>12</td>
<td>Consider all levels for aggregation. This is same as default for primary</td>
</tr>
<tr>
<td></td>
<td></td>
<td>hierarchies, but not for secondary hierarchies.</td>
</tr>
<tr>
<td>ESS_AGGLEVELUSAGE_NOAGGREGATION</td>
<td>13</td>
<td>Do not aggregate along this hierarchy. All views selected are at the input</td>
</tr>
<tr>
<td></td>
<td></td>
<td>level.</td>
</tr>
<tr>
<td>ESS_AGGLEVELUSAGE_BOTTOMONLY</td>
<td>14</td>
<td>Applies only to secondary hierarchies. Consider only lowest level of this</td>
</tr>
<tr>
<td></td>
<td></td>
<td>hierarchy for aggregation.</td>
</tr>
<tr>
<td>ESS_AGGLEVELUSAGE_TOPONLY</td>
<td>15</td>
<td>Applies only to primary hierarchies. Consider only topmost level of this</td>
</tr>
<tr>
<td></td>
<td></td>
<td>hierarchy for aggregation.</td>
</tr>
<tr>
<td>ESS_AGGLEVELUSAGE_BOTTOMTOP</td>
<td>16</td>
<td>Applies to primary hierarchies. Select top and bottom levels only.</td>
</tr>
</tbody>
</table>

### Notes

- This function is applicable only to Release 9.3 or higher aggregate storage databases.
- Use this function to apply view selection properties to stored hierarchies to restrict Essbase from choosing certain levels for aggregation.

### Return Value

Returns 0 if successful.

### Example

```c
ESS_STS_T         sts = ESS_STS_NOERR;
ESS_HOUTLINE_T    hOutline = ESS_NULL;
ESS_HMEMBER_T     hMember = ESS_NULL;
ESS_SHORT_T       sAggLevelUsage = 0;

/* code to assign hOutline variable omitted */
/* code to assign hMember variable omitted */
/* code to assign sAggLevelUsage variable omitted */

if (hOutline && hMember)
{
    sts = EssOtlSetAggLevelUsage (hOutline, hMember, sAggLevelUsage);
    if (sts)
        printf("Error (%ld) setting AggLevelUsage\n", sts);
    else
```


```c
{
    if (!hOutline)
        printf("Outline not provided\n");
    if (!hMember)
        printf("Member not provided\n");
}

See Also

- EssOtlGetAggLevelUsage

### EssOtlSetAliasTableLanguage

Sets a language code for the specified alias table.

By setting alias table language codes, when an application running in an ApplCore session accesses an Essbase database, the correct alias table is automatically selected on application selection.

#### Syntax

```
ESS_FUNC_M EssOtlSetAliasTableLanguage (hOutline, pszAliasTable, pszLanguageCode);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline context handle.</td>
</tr>
<tr>
<td>pszAliasTable</td>
<td>ESS_STR_T</td>
<td>Name of the alias table for which to set a language code.</td>
</tr>
<tr>
<td>pszLanguageCode</td>
<td>ESS_STR_T</td>
<td>A language code to assign to the alias table specified in <code>pszAliasTable</code>. The language code should be a middle-tier language tag from an ApplCore session. Language codes are not case-sensitive.</td>
</tr>
</tbody>
</table>

#### Notes

- You cannot set a language code on the default alias table.
- Any number of language codes can be assigned to an alias table. To set multiple language codes, call this function for each language code.
- Setting a new language code does not override language codes currently assigned to the alias table.
- The same language code must not be assigned to another alias table within the same database.

#### Return Value

- If successful, returns 0.
- If unsuccessful, returns one of the following errors:
  - OTLAPI_BAD_ALIASTABLE (invalid alias table)
  - OTLAPI_ERR_DUP_LANGCODE (the language code is assigned to another alias table within the same database)
Access

This function does not require special privileges.

Example

```c
#include <essapi.h>
#include <essotl.h>

ESS_STS_T sts = 0;
ESS_OUTLINEINFO_T NewInfo;
ESS_HOUTLINE_T hOutline;
ESS_PALIASLANG_T pLangs=ESS_NULL;
ESS_ULONG_T nLangs = 0, i=0;

memset(&NewInfo, '\0', sizeof(NewInfo));
sts = EssOtlNewOutline(hCtx, &NewInfo, &hOutline);

if (!sts)
{
    sts = EssOtlCreateAliasTable(hOutline,
        "French Alias Table");
}

if (!sts)
{
    sts = EssOtlSetAliasTableLanguage (hOutline,
        "French Alias Table", "fr");
}

if (!sts)
{
    sts = EssOtlSetAliasTableLanguage (hOutline,
        "French Alias Table", "fr-CA");
}

if (!sts)
{
    sts = EssOtlGetAliasTableLanguages(hOutline, "French Alias Table", &nLangs,
        &pLangs);

    if ( !sts == ESS_STS_NOERR && (pLangs) )
    {
        for (i=0;i<nLangs ;++i)
        {
            if (pLangs[i])
            {
                printf("Language Code:  %s\n", pLangs[i]);
            }
        }
        EssFree(hInst, pLangs);
    }
}
if (!sts)
{
    sts = EssOtlClearAliasTableLanguages (hOutline,
        "French Alias Table");
}
```
EssOtlSetAltHierarchyEnabled

Sets a dimension to be multiple-hierarchy enabled.

**Syntax**

```
ESS_FUNC_M EssOtlSetAltHierarchyEnabled(hOutline, hDimMember, cEnabled);
```

**Parameter**  
**Data Type**  
**Description**

- **hOutline**  
  ESS_HOUTLINE_T  
  Outline context handle (input).

- **hDimMember**  
  ESS_HMEMBER_T  
  A dimension member (input).

- **cEnabled**  
  ESS_BOOL_T  
  If TRUE, the dimension is set to enable multiple hierarchies. If FALSE, the dimension is set to a single, stored hierarchy.

**Return Value**

- 0—If successful
- Returns error OTLAPI_ERR_BADDIM if `hDimMember` is not a dimension member.

See Also

- EssOtlGetAltHierarchyEnabled
- EssOtlGetHierarchyType
- EssOtlSetHierarchyType

EssOtlSetASOCOMpressionDimension

Tags an aggregate storage dimension as Compression.

**Syntax**

```
ESS_FUNC_M EssOtlSetASOCOMpressionDimension (hOutline, hDim);
```

**Parameter**  
**Data Type**  
**Description**

- **hOutline**  
  ESS_HOUTLINE_T  
  Outline context handle (input).

- **hDim**  
  ESS_HMEMBER_T  
  Dimension handle (input).

**Notes**

- By default, the compression dimension in aggregate storage databases is the Accounts dimension. To get the current compression dimension, use EssOtlGetASOCOMpressionDimension. Changing the compression dimension triggers a full restructure of the database.
- Only one dimension can be the compressed dimension at any time. The API will automatically unset any previous dimension when a new one is set. Attribute dimensions cannot be compression dimensions.

- It is legal for an outline to not have any dimension selected as the compressed dimension. Calling this function with `hDim` set to NULL will unset the current compression dimension.

- Essbase requires the compression dimension to be a single, dynamic hierarchy. If the dimension has a different hierarchy setting, such as multiple hierarchies, it will be set to single dynamic hierarchy automatically. The original hierarchy setting is lost (setting a different dimension as compression does not return the original hierarchy setting).

- The choice of compression dimension can significantly affect performance. Large dimensions are never good choices for compression dimensions.

**Return Value**

Returns 0 if successful.

**Example**

```c
ESS_STS_T         sts = ESS_STS_NOERR;
ESS_HOUTLINE_T    hOutline = ESS_NULL;
ESS_HMEMBER_T     hMember = ESS_NULL;

/* code to assign hOutline variable omitted */
/* code to assign hMember variable omitted */

if (hOutline)
{
    sts = EssOtlSetASOCompressionDimension(hOutline, hMember);
    if (sts)
        printf("Error (%ld) setting compression dimension\n", sts);
    else
        if (hMember)
            printf("Compression dimension set\n");
        else
            printf("Compression dimension cleared\n");
} else
{
    printf("Outline not provided\n");
}
```

**See Also**

- `EssOtlGetASOCompressionDimension`

**EssOtlSetAttributeSpecifications**

Sets attribute specifications for the outline.

**Syntax**
### Parameter Data Type Description

- **hOutline:** ESS_HOUTLINE_T Handle to the outline
- **pAttrSpecs:** “ESS_ATTRSPECS_T” on page 119 Attribute specifications

### Notes

- Attribute specifications are used to do the following:
  - Generate a long name
  - Indicate the format of a datetime attribute
  - Indicate a numeric attribute's bucketing type
  - Provide the name of the attribute calculations dimension and the names for the values used with it
- If you do not set attribute specifications, the outline uses the default attribute specifications.
- Changing attribute specifications may cause a restructure.

### Return Value

If renaming attribute members fails, OTLAPI_ERR_ATTRRENAME_ERROR is returned.

### Example

```c
void ESS_OtlSetAttributeSpecifications()
{
    ESS_STS_T         sts = ESS_STS_NOERR;
    ESS_ATTRSPECS_T   AttrSpecs;
    ESS_CHAR_T        buffer[8][20];
    ESS_OBJDEF_T      Object;
    ESS_HOUTLINE_T    hOutline;
    ESS_APPNAME_T     szAppName;
    ESS_DBNAME_T      szDbName;
    ESS_OBJNAME_T     szFileName;
    ESS_PROCSTATE_T   pState;
    int              test;

    memset(&Object, '\0', sizeof(Object));
    Object.hCtx = hCtx;
    Object.ObjType = ESS_OBJTYPE_OUTLINE;
    strcpy(szAppName, "Sample");
    strcpy(szDbName, "Basic");
    strcpy(szFileName, "Basic");
    Object.AppName = szAppName;
    Object.DbName = szDbName;
    Object.FileName = szFileName;

    sts = EssOtlOpenOutline(hCtx, &Object, ESS_TRUE, ESS_TRUE, &hOutline);

    printf("\n\nEnter the NUMBERS for the appropriate choices that follow.");
    printf("\n\nEnter GenNameBy:
		0. ESS_GENNAMEBY_PREFIX
		1. ESS_GENNAMEBY_SUFFIX
\n\nChoice: ");
```

921 C Outline API Functions
test = atoi(gets(buffer[0]));
switch (test)
{
    case 0:
        AttrSpecs.usGenNameBy=ESS_GENNAMEBY_PREFIX;
        break;
    case 1:
        AttrSpecs.usGenNameBy=ESS_GENNAMEBY_SUFFIX;
        break;
    default:
        printf("\n\nInvalid choice.\n\n");
}
printf("\n\nEnter UseNameOf: \n	0. ESS_USENAMEOF_NONE
	1. ESS_USENAMEOF_PARENT\n	2. ESS_USENAMEOF_GRANDPARENTANDPARENT
	3. ESS_USENAMEOF_ALLANCESTORS\n	4. ESS_USENAMEOF_DIMENSION\nChoice: ");
    test = atoi(gets(buffer[0]));
switch (test)
{
    case 0:
        AttrSpecs.usUseNameOf=ESS_USENAMEOF_NONE;
        break;
    case 1:
        AttrSpecs.usUseNameOf=ESS_USENAMEOF_PARENT;
        break;
    case 2:
        AttrSpecs.usUseNameOf=ESS_USENAMEOF_GRANDPARENTANDPARENT;
        break;
    case 3:
        AttrSpecs.usUseNameOf=ESS_USENAMEOF_ALLANCESTORS;
        break;
    case 4:
        AttrSpecs.usUseNameOf=ESS_USENAMEOF_DIMENSION;
        break;
    default:
        printf("\n\nInvalid choice.\n\n");
}
printf("Enter Delimiter: \n	0. ESS_DELIMITER_UNDERSCORE
	1. ESS_DELIMITER_PIPE\n	2. ESS_DELIMITER_CARET\nChoice: ");
    test = atoi(gets(buffer[0]));
switch (test)
{
    case 0:
        AttrSpecs.cDelimiter=ESS_DELIMITER_UNDERSCORE;
        break;
    case 1:
        AttrSpecs.cDelimiter=ESS_DELIMITER_PIPE;
        break;
    case 2:
        AttrSpecs.cDelimiter=ESS_DELIMITER_CARET;
        break;
    default:
        printf("\n\nInvalid choice.\n\n");
}
printf("Enter DateFormat:
		0.  ESS_DATEFORMAT_MMDDYYYY
		1.  ESS_DATEFORMAT_DDMMYYYY

Choice: ");
test = atoi(gets(buffer[0]));
switch (test) {
    case 0:
        AttrSpecs.usDateFormat=ESS_DATEFORMAT_MMDDYYYY;
        break;
    case 1:
        AttrSpecs.usDateFormat=ESS_DATEFORMAT_DDMMYYYY;
        break;
    default:
        printf("\nInvalid choice.\n\n");
}
printf("Enter BucketingType:
		0.  ESS_UPPERBOUNDINCLUSIVE
		1.  ESS_LOWERBOUNDINCLUSIVE
		2.  ESS_UPPERBOUNDNONINCLUSIVE
		3.  ESS_LOWERBOUNDNONINCLUSIVE\n
Choice: ");
test = atoi(gets(buffer[0]));
switch (test) {
    case 0:
        AttrSpecs.usBucketingType=ESS_UPPERBOUNDINCLUSIVE;
        break;
    case 1:
        AttrSpecs.usBucketingType=ESS_LOWERBOUNDINCLUSIVE;
        break;
    default:
        printf("\nInvalid choice.\n\n");
}
printf("Enter a word for your default true string (or 'ESS_DEFAULT_TRUESTRING'):
\n");
gets(buffer[0]);
if (buffer[0] == "ESS_DEFAULT_TRUESTRING")
    AttrSpecs.pszDefaultTrueString = "";
else
    AttrSpecs.pszDefaultTrueString=buffer[0];
printf("Enter your default false string (or 'ESS_DEFAULT_FALSESTRING'):\n");
gets(buffer[1]);
if (buffer[1] == "ESS_DEFAULT_FALSESTRING")
    AttrSpecs.pszDefaultFalseString = "";
else
    AttrSpecs.pszDefaultFalseString=buffer[1];
printf("Enter your default attribute calculation dimension name (or 'ESS_DEFAULT_ATTRIBUTECALCULATIONS'):\n");
gets(buffer[2]);
if (buffer[2] == "ESS_DEFAULT_ATTRIBUTECALCULATIONS")
    AttrSpecs.pszDefaultAttrCalcDimName="";
else
    AttrSpecs.pszDefaultAttrCalcDimName=buffer[2];
printf("Enter your default sum member name (or 'ESS_DEFAULT_SUM'): \n");
gets(buffer[3]);
if (buffer[3] == "ESS_DEFAULT_SUM")
    AttrSpecs pszDefaultSumMbrName = "";
else
    AttrSpecs pszDefaultSumMbrName = buffer[3];

printf("Enter your default count member name (or 'ESS_DEFAULT_COUNT'): \n");
gets(buffer[4]);
if (buffer[4] == "ESS_DEFAULT_COUNT")
    AttrSpecs pszDefaultCountMbrName = "";
else
    AttrSpecs pszDefaultCountMbrName = buffer[4];

printf("Enter your default average member name (or 'ESS_DEFAULT_AVERAGE'): \n");
gets(buffer[5]);
if (buffer[5] == "ESS_DEFAULT_AVERAGE")
    AttrSpecs pszDefaultAverageMbrName = "";
else
    AttrSpecs pszDefaultAverageMbrName = buffer[5];

printf("Enter your default minimum member name (or 'ESS_DEFAULT_MIN'): \n");
gets(buffer[6]);
if (buffer[6] == "ESS_DEFAULT_MIN")
    AttrSpecs pszDefaultMinMbrName = "";
else
    AttrSpecs pszDefaultMinMbrName = buffer[6];

printf("Enter your default maximum member name (or 'ESS_DEFAULT_MAX'): \n");
gets(buffer[7]);
if (buffer[7] == "ESS_DEFAULT_MAX")
    AttrSpecs pszDefaultMaxMbrName = "";
else
    AttrSpecs pszDefaultMaxMbrName = buffer[7];

sts = EssOtlSetAttributeSpecifications(hOutline, &AttrSpecs);
printf("EssOtlSetAttributeSpecifications() sts: %ld\n", sts);

sts = EssOtlWriteOutline(hOutline, &Object);
printf("EssOtlWriteOutline() sts: %ld\n", sts);

sts = EssOtlRestructure(hCtx, ESS_DOR_ALLDATA);
printf("EssOtlRestructure() sts: %ld\n", sts);

if (!sts)
{
    sts = EssGetProcessState(hCtx, &pState);
    while (!sts || (pState.State != ESS_STATE_DONE))
    {
        sts = EssGetProcessState(hCtx, &pState);
    }
    sts = EssOtlCloseOutline(hOutline);
    printf("EssOtlCloseOutline() sts: %ld\n", sts);
}

See Also

- EssCheckAttributes
EssOtlSetDateFormatString

This function sets the outline property date format string.

**Syntax**

ESS_FUNC_M EssOtlSetDateFormatString(
    ESS_HOUTLINE_T hOutline,
    ESS_STR_T formatString)

**Parameter**

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T Outline for the Smartlist.</td>
</tr>
<tr>
<td>formatString</td>
<td>ESS_STR_T Returns the outline date format string to this argument.</td>
</tr>
</tbody>
</table>

**Return Value**

Returns:

- 0—If successful
  
  *formatString* contains a date format string.

- Error number—If unsuccessful

**Example**

```c
void TestGetSetDateFormatString()
{
    ESS_STS_T            sts = ESS_STS_NOERR;
    ESS_HOUTLINE_T      hOutline = ESS_NULL;
    ESS_OBJDEF_T        Object;
    ESS_SHORT_T         length = 80;
    ESS_STR_T           dateFormatString = "";
    ESS_STR_T           localeStr;
    ESS_USHORT_T        count, i;
    ESS_STR_T*          pdateStrings;
    ESS_STR_T*          pformatStrings;
}
memset(&Object, '\0', sizeof(Object));
Object.hCtx = hCtx;
Object.ObjType = ESS_OBJTYPE_OUTLINE;
Object.AppName = szAppName;
Object.DbName = szDbName;
Object.FileName = szFileName;

sts = EssOtlOpenOutline(hCtx, &Object,
                        ESS_TRUE, ESS_TRUE, &hOutline);

/* Get current value */
sts = EssOtlGetDateFormatString(hOutline, &dateFormatString);
printf("EssOtlGetDateFormatString sts: %d \n", sts);
printf("\tDate format string: %s\n", dateFormatString);

printf("\n");
localeStr = "English_UnitedStates.Latin1@Binary";
sts = EssOtlGetServerDateFormats(hCtx, localeStr,
                                 &Count, &pdateStrings, &pformatStrings);
printf("EssOtlGetServerDateFormats sts: %d \n", sts);

for (i = 0; i < count; i++)
{
    printf("\nCase with %s:\n", pformatStrings[i]);
    sts = EssOtlSetDateFormatString(hOutline,
                                     pformatStrings[i]);
    printf("EssOtlSetSMDateFormatString sts: %d \n", sts);
    SaveOutline(hOutline);

    sts = EssOtlGetDateFormatString(hOutline,
                                     &dateFormatString);
    printf("\tDate format string: %s\n", dateFormatString);
}

sts = EssUnlockObject(hCtx, Object.ObjType,
                      Object.AppName, Object.DbName, Object.FileName);
sts = EssOtlCloseOutline(hOutline);
printf("EssOtlCloseOutline  sts: %d\n",sts);

See Also
- EssOtlGetServerDateFormats
- EssOtlGetDateFormatString

EssOtlSetDimensionNameUniqueness

Sets the dimension to prohibit duplicate (non-unique) member names.

Syntax

**ESS_FUNC_M EssOtlSetDimensionNameUniqueness (hOutline, hMember, bNameUnique);**
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline context handle (input).</td>
</tr>
<tr>
<td>hMember</td>
<td>ESS_HMEMBER_T</td>
<td>Member handle of the dimension root member (input).</td>
</tr>
<tr>
<td>bNameUnique</td>
<td>ESS_BOOL_T</td>
<td>The dimension member-name uniqueness setting (input). If set to TRUE, then the dimension cannot have duplicate member names.</td>
</tr>
</tbody>
</table>

**Notes**

Call `EssOtlFindMember` to set up the ESS_HMEMBER_T (hDim) variable.

**Return Value**

Returns 0 if successful; otherwise, returns an error.

**Example**

```c
ESS_FUNC_M ESS_GetSetDimNameUniq()
{
    ESS_STS_T    sts = 0;
    ESS_POUTLINEINFO_T  pInfo = ESS_NULL;
    ESS_HOUTLINE_T   hOutline;
    ESS_OBJDEF_T   Object;
    ESS_APPNAME_T   szAppName;
    ESS_DBNAME_T   szDbName;
    ESS_OBJNAME_T   szFileName;
    ESS_BOOL_T    pbNameUnique;
    ESS_HMEMBER_T   hDim = ESS_NULL;

    memset(&Object, '\0', sizeof(Object));
    Object.hCtx =   hCtx;
    Object.ObjType =  ESS_OBJTYPE_OUTLINE;
    strcpy(szAppName, "Demo");
    strcpy(szDbName, "Test");
    strcpy(szFileName, "Test");
    Object.AppName =  szAppName;
    Object.DbName =   szDbName;
    Object.FileName =  szFileName;

    sts = EssOtlOpenOutline(hCtx, &Object, ESS_TRUE, ESS_TRUE, &hOutline);

    if (!sts)
    {
        sts = EssOtlFindMember(hOutline, "Year", &hDim);

        if (sts)
            printf("EssOtlFindMember failed sts %ld\n",sts);
    }

    /*Get the dimension's, Year, member-name uniqueness setting */
    if (!sts)
    {
        sts = EssOtlGetDimensionNameUniqueness (hOutline, hDim, &pbNameUnique);
    }
}
```

C Outline API Functions
if (sts)
    printf("EssOtlGetDimensionNameUniqueness failed sts %ld\n",sts);
else
    printf("Dimension Year has Member Name Uniqueness value: %ld\n", pbNameUnique);
}

if (!sts)
{
    sts = EssOtlFindMember(hOutline, "Product", &hDim);
    if (sts)
        printf("EssOtlFindMember failed sts %ld\n",sts);
}

if (!sts)
{
  /*set Product to prohibit duplicate (non-unique) member names*/
  pbNameUnique = ESS_TRUE;
  sts = EssOtlSetDimensionNameUniqueness (hOutline, hDim, pbNameUnique);
  if (sts)
      printf("EssOtlSetDimensionNameUniqueness failed sts %ld\n",sts);
  else
      printf("Dimension Product has Member Name Uniqueness value: %ld\n", pbNameUnique);
}

return sts;

See Also
- EssOtlGetDimensionNameUniqueness

EssOtlSetDimensionSolveOrder
Sets the solve order of a dimension.

Syntax

    ESS_FUNC_M EssOtlSetDimensionSolveOrder (hOutline, hMember, cOrder);

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline context handle (input).</td>
</tr>
<tr>
<td>hMember</td>
<td>ESS_HMEMBER_T</td>
<td>Dimension handle (input).</td>
</tr>
<tr>
<td>cOrder</td>
<td>ESS_UCHAR_T</td>
<td>Solve order (input). 0 - 127</td>
</tr>
</tbody>
</table>

Notes
- Solve order property on a member or dimension specifies its calculation order.
- Member solve order takes precedence over dimension solve order. Solve order can be between 0 and 127. The default is 0.
Members without formulas that do not have a specified solve order inherit the solve order of their dimension. Members with formulas that do not have a specified solve order have a solve order of zero.

**Return Value**

Returns 0 if successful.

**Example**

```c
ESS_STS_T sts = ESS_STS_NOERR;
ESS_HOUTLINE_T hOutline = ESS_NULL;
ESS_HMEMBER_T hMember = ESS_NULL;
ESS_UCHAR_T ucOrder = 0;

/* code to assign hOutline variable omitted */
/* code to assign hMember variable omitted */
/* code to assign ucOrder variable omitted */

if (hOutline && hMember)
{
    if (ucOrder > 127)
    {
        printf("Solve order must be less than 128\n");
    }
    else
    {
        sts = EssOtlSetDimensionSolveOrder(hOutline, hMember, ucOrder);
        if (sts)
            printf("Error [%ld] returned\n", sts);
        else
            printf("Solve Order: %d\n", ucOrder);
    }
}
else
    printf("Both hOutline and hMember must have values\n");
```

**See Also**

- EssOtlGetDimensionSolveOrder
- EssOtlSetMemberSolveOrder
- EssOtlGetMemberSolveOrder

### EssOtlSetDTSMemberAlias

Sets an alias name for a Dynamic Time Series (DTS) member.

**Syntax**

```c
ESS_STS_T EssOtlSetDTSMemberAlias (hOutline, pszDTSMember, pszAlias, pszAliasTable);
```

**Parameter** | **Data Type** | **Description**
--- | --- | ---
hOutline | ESS_HOUTLINE_T | The Essbase outline handle returned from the EssOtlOpenOutline call.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pszDTSMember</td>
<td>ESS_STR_T</td>
<td>Name of the DTS member which provides the alias.</td>
</tr>
<tr>
<td>pszAlias</td>
<td>ESS_STR_T</td>
<td>Pointer to a C string containing the alias name for the DTS member.</td>
</tr>
<tr>
<td>pzsAliasTable</td>
<td>ESS_STR_T</td>
<td>Name of the alias table which provides the alias. If NULL, the default alias table is used.</td>
</tr>
</tbody>
</table>

**Return Value**

If successful the return value is zero. Otherwise, one of the following is returned:

- OTLAPI_ERR_DTSMBRNOTDEFINED
- OTLAPI_BAD_ALIASTABLE
- OTLAPI_ERR_ILLEGALALIASSTRING
- OTLAPI_ERR_DUPLICATEALIAS

**Example**

```c
#include "essapi.h"
#include "essotl.h"
#include "esserror.h"

ESS_STS_T ESS_OtlSetDTSMemberAlias(ESS_HCTX_T hCtx, ESS_HINST_T hInst)
{
    ESS_STS_T       sts = ESS_STS_NOERR;
    ESS_OBJDEF_T    Object;
    ESS_HOUTLINE_T  hOutline;
    ESS_APPNAME_T   szAppName;
    ESS_DBNAME_T    szDbName;
    ESS_OBJNAME_T   szFileName;
    ESS_CHAR_T      pszAliasTable[ESS_ALIASNAMELEN];
    ESS_CHAR_T      pszAlias[ESS_ALIASNAMELEN];
    ESS_CHAR_T      pszDTSMember[ESS_MBRNAMELEN];
    ESS_PROCSTATE_T pState;
    ESS_ULONGLONG_T ulErrors;
    ESS_ULONGLONG_T ulCount;
    ESS_POUTERROR_T pMbrErrors = NULL;

    strcpy(szAppName, "sample");
    strcpy(szDbName, "Basic");
    strcpy(szFileName, "Basic");
    strcpy(pszDTSMember, "Q-T-D");
    strcpy(pszAliasTable, "Default");
    strcpy(pszAlias, "QuarterToDate");

    memset(&Object, '\0', sizeof(ESS_OBJDEF_T));
    Object.hCtx = hCtx;
    Object.ObjType = ESS_OBJTYPE_OUTLINE;
    Object.AppName = szAppName;
    Object.DbName = szDbName;
    Object.FileName = szFileName;

    sts = EssOtlOpenOutline(hCtx, &Object, ESS_TRUE, ESS_TRUE, &hOutline);
```
if (sts)
{
    printf("Could not open outline\n");
    return sts;
}
sts = EssOtlSetDTSMemberAlias(hOutline, pszDTSMember, pszAlias, pszAliasTable);
if (sts)
{
    printf("Could not set DTS member alias. Error is %d\n", sts);
}
sts = EssOtlWriteOutline(hOutline, &Object);
if (sts)
{
    printf("Could not write outline\n");
    return sts;
}
sts = EssOtlRestructure(hCtx, ESS_DOR_ALLDATA);
if (sts)
{
    printf("Could not restructure outline\n");
    return sts;
}
memset (&pState, 0, sizeof(ESS_PROCSTATE_T));
sts = EssGetProcessState(hCtx, &pState);
{
    while ((sts == ESS_STS_NOERR) && (pState.State != ESS_STATE_DONE))
    {
        memset (&pState, 0, sizeof(ESS_PROCSTATE_T));
        sts = EssGetProcessState(hCtx, &pState);
    }
}
sts = EssUnlockObject(hCtx, ESS_OBJTYPE_OUTLINE, szAppName, szDbName, szFileName);
if (sts)
{
    printf("Could not unlock outline\n");
    return sts;
}
EssOtlCloseOutline(hOutline);
return sts;
}

See Also

- EssOtlDeleteDTSMemberAlias
- EssOtlEnableDTSMember
- EssOtlGetEnabledDTSMembers
- EssOtlGetDTSMemberAlias
EssOtISetGenName

Sets the name for a specific generation within a dimension.

Syntax

```
ESS_FUNC_M EssOtISetGenName (hOutline, pszDimension, usGen, pszName);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline context handle.</td>
</tr>
<tr>
<td>pszDimension</td>
<td>ESS_STR_T</td>
<td>Name of dimension that contains the generation.</td>
</tr>
<tr>
<td>usGen</td>
<td>ESS_USHORT_T</td>
<td>Number of generation for which to set a name. The dimension itself is generation 1.</td>
</tr>
<tr>
<td>pszName</td>
<td>ESS_STR_T</td>
<td>Name to give the generation.</td>
</tr>
</tbody>
</table>

Notes

- The generation name must be unique across the entire member name space. It cannot duplicate any other generation, level, member name, or alias. Attempting to add a duplicate generation name generates an error.
- Each specific dimension and generation must have only one name.

Return Value

Returns 0 if successful; otherwise one of the following:

- OTLAPI_BAD_GENLEVELNAME
- OTLAPI_ERR_GENLEVELNAMEEXISTS
- OTLAPI_ERR_GENLEVELVALUE
- OTLAPI_ERR_NOTADIM
- OTLAPI_ERR_GENLEVELNAMEMBR

Example

```c
#include <essapi.h>
#include <essotl.h>

ESS_STS_T          sts = 0;
ESS_HOUTLINE_T     hOutline;
ESS_OBJDEF_T       Object;
ESS_APPNAME_T      szAppName;
ESS_DBNAME_T       szDbName;
ESS_OBJNAME_T      szFileName;
ESS_STR_T          Dimension;
ESS_USHORT_T       GenNum;
ESS_STR_T          GenName;

memset(&Object, '\0', sizeof(Object));
Object.hCtx = hCtx;
```
Object.ObjType = ESS_OBJTYPE_OUTLINE;
strcpy(szAppName, "Sample");
strcpy(szDbName, "Basic");
strcpy(szFileName, "Basic");
Object.AppName = szAppName;
Object.DbName = szDbName;
Object.FileName = szFileName;

sts = EssOtlOpenOutline(hCtx, &Object, ESS_TRUE,
                        ESS_TRUE, &hOutline);

/************  Set Generation Name ************/
Dimension = "Year";
GenNum = 2;
GenName = "Qtr123";

if (!sts)
{
    sts = EssOtlSetGenNameEx(hOutline, Dimension, GenNum, GenName);
}

See Also
● EssOtlDeleteGenName

**EssOtlSetGenNameEx**

Sets the generation name and member uniqueness setting for the specified generation number.

**Syntax**

```c
ESS_FUNC_M EssOtlSetGenNameEx (hOutline, pszDimension, usGen, pszName, bUniqueName);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline context handle.</td>
</tr>
<tr>
<td>pszDimension</td>
<td>ESS_STR_T</td>
<td>Dimension name.</td>
</tr>
<tr>
<td>usGen</td>
<td>ESS_USHORT_T</td>
<td>The number of the generation for which to set a name.</td>
</tr>
<tr>
<td>pszName</td>
<td>ESS_STR_T</td>
<td>The name to give the generation.</td>
</tr>
<tr>
<td>bUniqueName</td>
<td>ESS_BOOL_T</td>
<td>If TRUE, members at generation <code>usGen</code> in dimension <code>pszDimension</code> cannot have duplicate names.</td>
</tr>
</tbody>
</table>

**Notes**

● This function sets the name of a generation as well as the uniqueness property of a generation. If you only want to set the name, use EssOtlSetGenName.

● If you only want to set the uniqueness property, but not change the name, you must still pass in the name. To do so, call EssOtlSetGenName and pass its value to this function as `usGen`.

● Do not pass null for the `usGen` parameter.
Return Value

Returns 0 if successful; otherwise, returns an error code.

Example

void ESS_GetGenNameEx()
{

    ESS_STS_T          sts = 0;
    ESS_HOUTLINE_T     hOutline;
    ESS_OBJDEF_T       Object;
    ESS_APPNAME_T      szAppName;
    ESS_DBNAME_T       szDbName;
    ESS_OBJNAME_T      szFileName;
    ESS_STR_T          Dimension;
    ESS_USHORT_T       GenNum;
    ESS_STR_T          GenName;
    ESS_BOOL_T         bUnique= ESS_FALSE;

    memset(&Object, '\0', sizeof(Object));
    Object.hCtx = hCtx;
    Object.ObjType = ESS_OBJTYPE_OUTLINE;
    strcpy(szAppName, "Demo");
    strcpy(szDbName, "Test");
    strcpy(szFileName, "Test");
    Object.AppName = szAppName;
    Object.DbName = szDbName;
    Object.FileName = szFileName;

    sts = EssOtlOpenOutline(hCtx, &Object, ESS_TRUE, ESS_TRUE, &hOutline);
    printf("EssOtlOpenOutline sts: %ld\n",sts);

    /***************  Set and Get GenName **************/
    Dimension = "Year";
    GenNum = 1;
    GenName = "Gen 1 Year";

    //SetGenNameEx() so that Gen 1 members of Year cannot be non-unique
    if (!sts)
    {
        sts = EssOtlSetGenNameEx(hOutline, Dimension, GenNum, GenName, ESS_TRUE);
    }

    // GetGenNameEx() to see if the gen is able to be non-unique
    if (!sts)
    {
        sts = EssOtlGetGenNameEx(hOutline, Dimension, GenNum, &GenName, &bUnique);
        printf("Generation 1 members of Year have bUnique value of %ld\n", bUnique);
        printf("EssOtlGetGenNameEx sts: %ld\n",sts);
    }
}
if (!sts && GenName)
{
    printf("Gen Name: %s\n", GenName);
    EssFree(hInst, GenName);
}

See Also

- EssOtlSetGenName
- EssFree
- EssOtlDeleteGenName
- EssOtlGetGenNameEx

**EssOtlSetHierarchyType**

Sets the dimension's hierarchy type designation: Multiple hierarchies enabled, dynamic hierarchy, or stored hierarchy.

**Syntax**

```c
ESS_FUNC_M EssOtlSetHierarchyType(hOutline, hMember, cType);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline context handle (input).</td>
</tr>
<tr>
<td>hMember</td>
<td>ESS_HMEMBER_T</td>
<td>A dimension member (input).</td>
</tr>
<tr>
<td>cType</td>
<td>ESS_UCHAR_T</td>
<td>If <code>hMember</code> is a dimension member, one of the following values (input):</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_STORED_HIERARCHY—The dimension is a single, stored hierarchy.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_DYNAMIC_HIERARCHY—The dimension is a single, dynamic hierarchy.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_MULTIPLE_HIERARCHY_IS_ENABLED—The dimension is multiple-hierarchy enabled (same as using EssOtlSetAltHierarchyEnabled).</td>
</tr>
</tbody>
</table>

**Notes**

Once the dimension is multiple-hierarchy enabled, the hierarchy types are determined by the generation 2 members. If `hMember` is a generation 2 member, `cType` can have the following values:

- ESS_STORED_HIERARCHY—The hierarchy with `hMember` as top is a single, stored hierarchy.
- ESS_DYNAMIC_HIERARCHY—The hierarchy with `hMember` as top is a single, dynamic hierarchy.

**Return Value**

Returns 0 if successful; otherwise, returns an error.
EssOtlSetImpliedShare

Changes the Implied Share setting of an outline.

Syntax

ESS_FUNC_M EssOtlSetImpliedShare(hOutline, impliedShareSetting);

Parameter | Data Type | Description
---|---|---
hOutline | ESS_HOUTLINE_T | Outline context handle (input).

impliedShareSetting | ESS_USHORT | The implied share setting value. See "Implied Share Setting (C)" on page 103.

Return Value

- 0—If successful
- Error number—If unsuccessful

See Also

- EssOtlGetImpliedShare

EssOtlSetLevelName

Sets the name for a specific level within a dimension.

Syntax

ESS_FUNC_M EssOtlSetLevelName (hOutline, pszDimension, usLevel, pszName);

Parameter | Data Type | Description
---|---|---
hOutline | ESS_HOUTLINE_T | Outline context handle.

pszDimension | ESS_STR_T | Name of dimension that contains the level.

usLevel | ESS_USHORT_T | Number of level for which to set a name. Leaf members are level 0.

pszName | ESS_STR_T | Name to give the level.

Notes

- The level name follows the same rules as a member name and must be unique across the entire member name space. It cannot duplicate any other generation, level, member name, or alias. Attempting to add a duplicate name generates an error.
Each specific dimension and level must have only one name.

**Return Value**

Returns 0 if successful; otherwise one of the following:

- OTLAPI_BAD_GENLEVELNAME
- OTLAPI_ERR_GENLEVELNAMEEXISTS
- OTLAPI_ERR_GENLEVELNAMEMBR
- OTLAPI_ERR_GENLEVELNAMEEXISTS
- OTLAPI_ERR_NOTADIM
- OTLAPI_ERR_GENLEVELNAMEMBR

**Example**

```c
#include <essapi.h>
#include <essotl.h>

ESS_STS_T          sts = 0 ;
ESS_HOUTLINE_T     hOutline;
ESS_OBJDEF_T       Object;
ESS_APPNAME_T      szAppName;
ESS_DBNAME_T       szDbName;
ESS_OBJNAME_T      szFileName;
ESS_STR_T          Dimension;
ESS_USHORT_T       LevelNum;
ESS_STR_T          LevelName;

memset(&Object, '\0', sizeof(Object));
Object.hCtx = hCtx;
Object.ObjType = ESS_OBJTYPE_OUTLINE;
strcpy(szAppName, "Sample");
strcpy(szDbName, "Basic");
strcpy(szFileName, "Basic");
Object.AppName  = szAppName;
Object.DbName   = szDbName;
Object.FileName = szFileName;

sts = EssOtlOpenOutline(hCtx, &Object, ESS_TRUE, ESS_TRUE, &hOutline);

/********************  Set Level Name  ********************/
Dimension = "Year";
LevelNum = 1;
LevelName = "Qtr 1 2 3";

if (!sts)
{
    sts = EssOtlSetLevelName(hOutline, Dimension, LevelNum, LevelName);
}
```

**See Also**

- EssOtlDeleteLevelName
- EssOtlGetLevelName
**EssOtlSetLevelNameEx**

Sets whether members in a certain dimension at a certain level are prohibited from having duplicate names.

**Syntax**

```c
ESS_FUNC_M EssOtlSetLevelNameEx(hOutline, pszDimension, usLevel, pszName, bUniqueName);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline context handle.</td>
</tr>
<tr>
<td>pszDimension</td>
<td>ESS_STR_T</td>
<td>Name of dimension that contains the level.</td>
</tr>
<tr>
<td>usLevel</td>
<td>ESS_USHORT_T</td>
<td>Number of level for which to set a name. Leaf members are level 0.</td>
</tr>
<tr>
<td>pszName</td>
<td>ESS_STR_T</td>
<td>Name to give the level.</td>
</tr>
<tr>
<td>bUniqueName</td>
<td>ESS_BOOL_T</td>
<td>If TRUE, members at level <code>usLevel</code> in dimension <code>pszDimension</code> cannot have duplicate names. If FALSE, duplicate names are allowed.</td>
</tr>
</tbody>
</table>

**Notes**

- The level name must be unique across the entire member name space. It cannot duplicate any other generation, level, member name, or alias. Attempting to add a duplicate level name generates an error.
- This function sets the name of a level as well as the uniqueness property of a level. If you only want to set the name, use `EssOtlSetLevelName`.
- If you only want to set the uniqueness property, but not change the name, you must still pass in the name. To do so, call `EssOtlGetLevelName` and pass its value to this function as `usLevel`.
- Do not pass null for the `usLevel` parameter.

**Return Value**

Returns 0 if successful; otherwise, returns an error code.

**Example**

```c
ESS_FUNC_M
ESS_GetLevelNameEx()
{
    ESS_STS_T                 sts = 0;
    ESS_HOUTLINE_T            hOutline;
    ESS_OBJDEF_T              Object;
    ESS_APPNAME_T             szAppName;
    ESS_DBNAME_T              szDbName;
    ESS_OBJNAME_T             szFileName;
    ESS_STR_T                 Dimension;
    ESS_USHORT_T              LevelNum;
    ESS_STR_T                 LevelName;
    ESS_BOOL_T                bUnique= ESS_FALSE;
```
memset(&Object, '\0', sizeof(Object));
Object.hCtx = hCtx;
Object.ObjType = ESS_OBJTYPE_OUTLINE;
strcpy(szAppName, "Demo");
strcpy(szDbName, "Test");
strcpy(szFileName, "Test");
Object.AppName = szAppName;
Object.DbName = szDbName;
Object.FileName = szFileName;

sts = EssOtlOpenOutline(hCtx, &Object, ESS_TRUE,
    ESS_TRUE, &hOutline);

/***************  Set and Get Level Name **************/
    Dimension = "Year";
    LevelNum = 0;
    LevelName = "Level 0 Year";

    //SetLevelNameEx() so that level 0 member of Year cannot be non-unique
    if (!sts)
    {
        sts = EssOtlSetLevelNameEx(hOutline, Dimension,
        LevelNum, LevelName, ESS_TRUE);
    }

    // GetLevelNameEx() to see if the level is able to be non-unique
    if (!sts)
    {
        sts = EssOtlGetLevelNameEx(hOutline, Dimension,
        LevelNum, &LevelName, &bUnique);
        printf("Level 0 members of Year have bUnique value of %ld\n", bUnique);
    }

    if (!sts && LevelName)
    {
        printf("Level Name: %s\n",LevelName);
        EssFree(hInst, LevelName);
    }

    return (sts);
}

See Also
● EssOtlGetLevelNameEx

**EssOtlSetMemberAlias**

Sets the default member alias for the specified member in the specified alias table.

**Syntax**

```
ESS_FUNC_M EssOtlSetMemberAlias (hOutline, hMember, pszAliasTable, pszAlias);
```
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline context handle.</td>
</tr>
<tr>
<td>hMember</td>
<td>ESS_HMEMBER_T</td>
<td>Handle of member to set the alias for.</td>
</tr>
<tr>
<td>pszAliasTable</td>
<td>ESS_STR_T</td>
<td>Alias table to set the alias for. If this parameter is ESS_NULL, the default alias table is used.</td>
</tr>
<tr>
<td>pszAlias</td>
<td>ESS_STR_T</td>
<td>Alias.</td>
</tr>
</tbody>
</table>

**Notes**

Use EssOtlDeleteMemberAlias() to remove an alias.

**Return Value**

Returns 0 if successful; otherwise one of the following:

- OTLAPI_BAD_ALIAS_TABLE
- OTLAPI_ERR_ILLEGALDEFALIAS
- OTLAPI_ERR_ILLEGALCOMBOALIAS
- OTLAPI_ERR_ILLEGALALIASSTRING
- OTLAPI_ERR_DUPLICATEALIAS

**Example**

```c
#include <essapi.h>
#include <essotl.h>

ESS_STS_T          sts = 0;
ESS_OBJDEF_T       Object;
ESS_HOUTLINE_T     hOutline;
ESS_HMEMBER_T      hMember;
ESS_APPNAME_T      szAppName;
ESS_DBNAME_T       szDbName;
ESS_OBJNAME_T      szFileName;
memset(&Object, '\0', sizeof(Object));
Object.hCtx = hCtx;
Object.ObjType = ESS_OBJTYPE_OUTLINE;
strcpy(szAppName, "Sample");
strcpy(szDbName, "Basic");
strcpy(szFileName, "Basic");
Object.AppName = szAppName;
Object.DbName = szDbName;
Object.FileName = szFileName;

sts = EssOtlOpenOutline(hCtx, &Object, ESS_TRUE,
                        ESS_TRUE, &hOutline);

if (!sts)
{
    sts = EssOtlFindMember(hOutline, "Year",
                           &hMember);
}
```
if (!sts && hMember) {
    sts = EssOtlSetMemberAlias(hOutline, hMember, ESS_NULL, "Time Dimension");
}

See Also
- EssOtlGetMemberAlias
- EssOtlDeleteMemberAlias

**EssOtlSetMemberCommentEx**

Sets the extended comment for the specified member.

**Syntax**

```c
ESS_FUNC_M EssOtlSetMemberCommentEx (hOutline, hMember, pszCommentEx);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline context handle</td>
</tr>
<tr>
<td>hMember</td>
<td>ESS_HMEMBER_T</td>
<td>Member handle.</td>
</tr>
<tr>
<td>pszCommentEx</td>
<td>ESS_STR_T</td>
<td>Buffer containing the extended comment.</td>
</tr>
</tbody>
</table>

**Notes**

- To delete an extended comment, call this function with an empty string or a null pointer.

**Return Value**

Returns 0 if successful; OTLAPI_ERR_MBCOMPONENTEXLEN if the comment is too long.

**Example**

```c
#include <essapi.h>
#include <essotl.h>

ESS_STS_T         sts = 0;
ESS_HOUTLINE_T    hOutline;
ESS_OBJDEF_T      Object;
ESS_APPNAME_T     szAppName;
ESS_DBNAME_T      szDbName;
ESS_OBNAME_T      szFileName;
ESS_HMEMBER_T     hMember;
ESS_STR_T         pszCommentEx;

memset(&Object, '\0', sizeof(Object));
Object.hCtx = hCtx;
Object.ObjType = ESS_OBJTYPE_OUTLINE;
strcpy(szAppName, "Sample");
strcpy(szDbName, "Basic");
strcpy(szFileName, "Basic");
Object.AppName = szAppName;
```

932  C Outline API Functions
Object.DbName = szDbName;
Object.FileName = szFileName;

sts = EssOtlOpenOutline(hCtx, &Object, ESS_TRUE, ESS_TRUE, &hOutline);

/************ Set Extended Member Comment *************/
pszCommentEx = "EXTENDED MEMBER COMMENT";

if (!sts)
{
    sts = EssOtlFindMember(hOutline, "Variance", &hMember);
}

if (!sts && hMember)
{
    sts = EssOtlSetMemberCommentEx(hOutline, hMember, pszCommentEx);
}

See Also
- EssFree
- EssOtlGetMemberCommentEx
- EssOtlOpenOutline

---

**EssOtlSetMemberFormula**

Sets the formula for the specified member.

**Syntax**

```c
ESS_FUNC_M EssOtlSetMemberFormula (hOutline, hMember, pszFormula);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline context handle.</td>
</tr>
<tr>
<td>hMember</td>
<td>ESS_HMEMBER_T</td>
<td>Member handle.</td>
</tr>
<tr>
<td>pszFormula</td>
<td>ESS_STR_T</td>
<td>Buffer containing the member formula.</td>
</tr>
</tbody>
</table>

**Notes**
- Use EssOtlDeleteMemberFormula() to remove a member formula.

**Return Value**

Returns 0 if successful; otherwise one of the following:
- OTLAPI_ERR_SHAREDMEMBERFORMULA
- OTLAPI_ERR_MEMBERCALC

**Example**

```c
#include <essapi.h>
#include <essotl.h>

ESS_STS_T   sts = 0;
```
memset(&Object, '\0', sizeof(Object));
Object.hCtx = hCtx;
Object.ObjType = ESS_OBJTYPE_OUTLINE;
strcpy(szAppName, "Sample");
strcpy(szDbName, "Basic");
strcpy(szFileName, "Basic");
Object.AppName  = szAppName;
Object.DbName   = szDbName;
Object.FileName = szFileName;

sts = EssOtlOpenOutline(hCtx, &Object, ESS_TRUE, ESS_TRUE, &hOutline);

/************  Set Member Formula ************/
pszFormula = "@VAR(Budget, Actual);";
if (!sts)
{
    sts = EssOtlFindMember(hOutline, "Variance", &hMember);
}
if (!sts && hMember)
{
    sts = EssOtlSetMemberFormula(hOutline, hMember, pszFormula);
}

See Also
- EssOtlGetMemberFormula
- EssOtlDeleteMemberFormula

**EssOtlSetMemberInfo**

Sets member attribute information.

**Syntax**

ESS_FUNC_M EssOtlSetMemberInfo (hOutline, hMember, pInfo);

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline context handle</td>
</tr>
<tr>
<td>hMember</td>
<td>ESS_HMEMBER_T</td>
<td>Handle to member to set attributes for</td>
</tr>
<tr>
<td>pInfo</td>
<td>“ESS_MBRINFO_T” on page 699</td>
<td>Member information structure</td>
</tr>
</tbody>
</table>
### Notes

#### Attributes

- Three fields of the ESS_MBRINFO_T structure are for attributes only:

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_BOOL_T</td>
<td>fAttributed</td>
<td>Indicates whether a base member has attributes associated with it: either ESS_TRUE or ESS_FALSE.</td>
</tr>
</tbody>
</table>
| ESS_USHORT_T| Attribute.usDataType | For an attribute dimension or zero-level (leaf node) attribute member, one of the following data types:
|             |                   | - ESS_ATTRMBRDT_BOOL
|             |                   | - ESS_ATTRMBRDT_DATETIME
|             |                   | - ESS_ATTRMBRDT_DOUBLE
|             |                   | - ESS_ATTRMBRDT_STRING
|             |                   | For any attribute member, but not an attribute dimension:
|             |                   | - ESS_ATTRMBRDT_NONE
|             |                   | - ESS_ATTRMBRDT_AUTO
|             |                   | **Note:** Use ESS_ATTRMBRDT_AUTO only when adding a member. See Notes on Adding an Attribute Member. |

- Values for two fields of the ESS_MBRINFO_T structure are for attributes only:

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
</table>
| ESS_USHORT_T  | usCategory  | One of the following dimension categories:
|               |             | - ESS_CAT_ATTRIBUTE
|               |             | - ESS_CAT_ATTRCALC (for internal use only)            |
| ESS_USHORT_T  | usStorageCategory | One of the following dimension storage categories:
|               |             | - ESS_STORECAT_ATTRIBUTE
|               |             | - ESS_STORECAT_ATTRCALC (for internal use only)        |

#### Return Value

Returns 0 if successful; otherwise one of the following:

- OTLAPI_BAD_CONSOL
- OTLAPI_BAD_MBRNAME
- OTLAPI_BAD_MEMBER
Example

```c
#include <essapi.h>
#include <essotl.h>

ESS_STS_T          sts = 0;
ESS_HOUTLINE_T     hOutline;
ESS_HMEMBER_T      hMemberJan;
ESS_MBRINFO_T      MbrInfo;
ESS_PMBRINFO_T     pMbrInfo = ESS_NULL;
ESS_OBJDEF_T       Object;
ESS_APPNAME_T      szAppName;
ESS_DBNAME_T       szDbName;
ESS_OBJNAME_T      szFileName;
memset(&Object, '\0', sizeof(Object));
Object.hCtx = hCtx;
Object.ObjType = ESS_OBJTYPE_OUTLINE;
strcpy(szAppName, "Sample");
strcpy(szDbName, "Basic");
strcpy(szFileName, "Basic");
Object.AppName = szAppName;
Object.DbName = szDbName;
Object.FileName = szFileName;

sts = EssOtlOpenOutline(hCtx, &Object, ESS_TRUE,
                        ESS_TRUE, &hOutline);

if (!sts)
```

936  C Outline API Functions
{ sts = EssOtlFindMember(hOutline, "Jan", &hMemberJan); }

if (!sts && hMemberJan)
{
    sts = EssOtlGetMemberInfo(hOutline, hMemberJan, &pMbrInfo);
}

if (!sts && pMbrInfo)
{
    pMbrInfo->usConsolidation = ESS_UCALC_SUB;
    pMbrInfo->fTwoPass = ESS_TRUE;
    pMbrInfo->fExpense = ESS_TRUE;
    sts = EssOtlSetMemberInfo(hOutline, hMemberJan, pMbrInfo);
}

if (pMbrInfo)
{
    EssOtlFreeStructure(hOutline, count, structId, structPtr);
}

See Also

- EssOtlGetMemberInfo
- EssOtlFindMember

EssOtlSetMemberSolveOrder

Sets the solve order of a member.

Syntax

ESS_FUNC_M EssOtlSetMemberSolveOrder (hOutline, hMember, cOrder);

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline context handle (input).</td>
</tr>
<tr>
<td>hMember</td>
<td>ESS_HMEMBER_T</td>
<td>Dimension handle (input).</td>
</tr>
<tr>
<td>cOrder</td>
<td>ESS_UCHAR_T</td>
<td>Solve order (input). 0 - 127</td>
</tr>
</tbody>
</table>

Notes

- Solve order is applicable only to aggregate storage databases.
- Solve order property on a member or dimension specifies its calculation order.
- Member solve order takes precedence over dimension solve order. Solve order can be between 0 and 127. The default is 0.
- Members without formulas that do not have a specified solve order inherit the solve order of their dimension. Members with formulas that do not have a specified solve order have a solve order of zero.

**Return Value**

Returns 0 if successful.

**Example**

```c
ESS_STS_T        sts = ESS_STS_NOERR;
ESS_HOUTLINE_T   hOutline = ESS_NULL;
ESS_HMEMBER_T    hMember = ESS_NULL;
ESS_UCHAR_T      ucOrder = 0;

/* code to assign hOutline variable omitted */
/* code to assign hMember variable omitted */
/* code to assign ucOrder variable omitted */

if (hOutline && hMember)
{
    if (ucOrder > 127)
    {
        printf("Solveorder must be less than 128\n");
    }
    else
    {
        sts = EssOtlSetMemberSolveOrder(hOutline, hMember, ucOrder);

        if (sts)
            printf("Error [%ld] returned\n", sts);
        else
            printf("Solve Order: %d\n", ucOrder);
    }
}
else
    printf("Both hOutline and hMember must have values\n");

**See Also**

- EssOtlGetMemberSolveOrder
- EssOtlSetDimensionSolveOrder
- EssOtlGetDimensionSolveOrder

**EssOtlSetMemberType**

Sets the member type of the input outline member.

**Syntax**

```
ESS_FUNC_M EssOtlSetMemberType(hOutline, hMember, usType)
```

**Parameter**  **Data Type**  **Description**

<p>| hOutline | ESS_HOUTLINE_T | Outline handle (Edit mode only) |</p>
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hMember</td>
<td>ESS_HMEMBER_T</td>
<td>Outline member handle</td>
</tr>
<tr>
<td>usType</td>
<td>ESS_USHORT_T</td>
<td>Type of the outline member:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_MEMBERTYPE_NUMERIC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Member type is a numeric.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_MEMBERTYPE_DATE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Member type is date typed.</td>
</tr>
</tbody>
</table>

**Notes**

Does not allow setting the type to ESS_MEMBERTYPE_SMARTLIST. Instead, use `EssOtlSetMemberTypeToSmartList`.

**Return Value**

Returns:
- 0—If successful
- Error number—If unsuccessful

**Example**

```c
void TestGetSetMemberType()
{
    ESS_STS_T                    sts = ESS_STS_NOERR;
    ESS_HOUTLINE_T            hOutline = ESS_NULL;
    ESS_OBJDEF_T                Object;
    ESS_HMEMBER_T            hMember;
    ESS_USHORT_T                usMemberType;

    memset(&Object, '\0', sizeof(Object));
    Object.hCtx =             hCtx;
    Object.ObjType =         ESS_OBJTYPE_OUTLINE;
    Object.AppName =         szAppName;
    Object.DbName =         szDbName;
    Object.FileName =     szFileName;

    /* Open outline */
    sts = EssOtlOpenOutline(hCtx, &Object, ESS_TRUE, 
                            ESS_TRUE, &hOutline);

    /* Find a member */
    sts = EssOtlFindMember(hOutline, "Original Price", &hMember);

    /* Get Member Type of an outline that is not member 
     * type enabled */
    /* Get original type */
    sts = EssOtlGetMemberType(hOutline, hMember, &usMemberType);
    DisplayMemberType(usMemberType); /* a display function */

    /* Set type to NUMERIC */
    usMemberType = ESS_MEMBERTYPE_NUMERIC;
    sts = EssOtlSetMemberType(hOutline, hMember, usMemberType);
```
*/ Set type to SmartList */
usMemberType = ESS_MEMBERTYPE_SMARTLIST;
sts = EssOtlSetMemberType(hOutline, hMember, usMemberType);
printf("EssOtlSetMemberType sts: %d\n", sts);

/* Set type to DATE */
usMemberType = ESS_MEMBERTYPE_DATE;
sts = EssOtlSetMemberType(hOutline, hMember, usMemberType);
printf("EssOtlSetMemberType sts: %d\n", sts);

/* Get Member Type of an outline that is member type enabled */
EnableSmartList(hOutline);

/* Get original type */
sts = EssOtlGetMemberType(hOutline, hMember, &usMemberType);
printf("EssOtlGetMemberType sts: %d\n", sts);
DisplayMemberType(usMemberType);

/* Set type to DATE */
usMemberType = ESS_MEMBERTYPE_DATE;
sts = EssOtlSetMemberType(hOutline, hMember, usMemberType);
printf("EssOtlSetMemberType sts: %d\n", sts);

sts = EssOtlGetMemberType(hOutline, hMember, &usMemberType);
printf("EssOtlGetMemberType sts: %d\n", sts);
DisplayMemberType(usMemberType);

/* Set type to NUMERIC */
usMemberType = ESS_MEMBERTYPE_NUMERIC;
sts = EssOtlSetMemberType(hOutline, hMember, usMemberType);
printf("EssOtlSetMemberType sts: %d\n", sts);

sts = EssOtlGetMemberType(hOutline, hMember, &usMemberType);
printf("EssOtlGetMemberType sts: %d\n", sts);
DisplayMemberType(usMemberType);

/* Clean up */
sts = EssUnlockObject(hCtx, Object.ObjType,
                      Object.AppName, Object.DbName, Object.FileName);

/* Close outline */
sts = EssOtlCloseOutline(hOutline);
}

See Also

- EssOtlGetMemberSmartList
- EssOtlCreateObject
- EssOtlDeleteObject
- EssOtlGetSmartListInfo
- EssOtlFindObject
- EssOtlFreeObjectArray
EssOtlSetMemberTypeToSmartList

Sets the input outline member as ESS_MEMBERTYPE_SMARTLIST and associates the input Text List (SmartList object with it).

Syntax

ESS_FUNC_M EssOtlSetMemberTypeToSmartList(hOutline, hMember, hSmartList)

Parameter       Data Type          Description

hOutline        ESS_HOUTLINE_T    Outline handle (Edit mode only)

hMember         ESS_HMEMBER_T     Outline member handle

hSmartList      ESS_HSMARTLIST_T  SmartList handle to be associated with.

Return Value

Returns:

● 0—If successful

● Error number—If unsuccessful

Example

void TestSetMemberTypeToSmartList()
{
    ESS_STS_T                    sts = ESS_STS_NOERR;
    ESS_HOUTLINE_T            hOutline = ESS_NULL;
    ESS_OBJDEF_T                Object;
    ESS_OBJECT_TYPES        objType;
    ESS_STR_T                    objName;
    ESS_HOBJECT_T            hObjHandle1, hObjHandle2;
    ESS_HMEMBER_T            hMember;
    ESS_HSMARTLIST_T        hSmartList;
    //ESS_USHORT_T            usVerifyType;

    memset(&Object, '\0', sizeof(Object));
    Object.hCtx =             hCtx;
    Object.ObjType =         ESS_OBJTYPE_OUTLINE;
    Object.AppName =         szAppName;
    Object.DbName =         szDbName;
}
Object.FileName = szFileName;

/* Open outline */
sts = EssOtlOpenOutline(hCtx, &Object, ESS_TRUE,
    ESS_TRUE, &hOutline);

/* Find a member */
sts = EssOtlFindMember(hOutline, "Original Price",
    &hMember);

/* Get original SmartList association */
sts = EssOtlGetMemberSmartList(hOutline, hMember,
    &hSmartList);

/* Set member type to SmartList */
hSmartList = (ESS_HSMARTLIST_T)hObjHandle1;
sts = EssOtlSetMemberTypeToSmartList(hOutline,
    hMember, hSmartList);

/* Unlock */
sts = EssUnlockObject(hCtx, Object.ObjType,
    Object.AppName, Object.DbName, Object.FileName);

/* Close outline */
sts = EssOtlCloseOutline(hOutline);
}

See Also

- EssOtlGetMemberSmartList
- EssOtlCreateObject
- EssOtlDeleteObject
- EssOtlGetSmartListInfo
- EssOtlFindObject
- EssOtlFreeObjectArray
- EssOtlFreeSmartListInfo
- EssOtlGetMemberSmartList
- EssOtlGetMemberType
- EssOtlGetObjectReferenceCount
- EssOtlGetObjectReferences
- EssOtlImportExportObject
- EssOtlListObjects
- EssOtlQueryObjects
- EssOtlSetMemberType
- EssOtlSetMemberTypeToSmartList

**EssOtlSetOriginalMember**

Sets a member as an extended shared member.

**Syntax**

```
ESS_FUNC_M EssOtlSetOriginalMember (hOutline, hMember, pszOriginalMbr);
```
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline context handle (input).</td>
</tr>
<tr>
<td>hMember</td>
<td>ESS_HMEMBER_T</td>
<td>Member name (input). This member will be set as extended shared.</td>
</tr>
<tr>
<td>pszOriginalMbr</td>
<td>ESS_STR_T</td>
<td>The original member name intended to share with (input).</td>
</tr>
</tbody>
</table>

**Notes**

- If `hMember` is not shared already, it will be marked as extended shared.
- If you use this function on an outline in which all member names are unique, it will have no effect.
- Before you call this function, call `EssOt1OpenOutline` to open the outline in editing mode.
- Given the following hierarchy, if you pass to this function the member handle (`hMember`) corresponding to `[Diet].[100-10]`, and the original member (`pszOriginalMbr`) as `[200].[100-10]`, then `[Diet].[100-10]` becomes an extended shared member of `[200].[100-10]`.

```
100
  100-10
200
  100-10 (duplicate)
Diet
  100-10 (shared with [200.100-10])
```

**Return Value**

Returns 0 if successful; otherwise, returns an error.

**Example**

```c
ESS_FUNC_M ESS_SetOrigMember()
{
    ESS_STS_T    sts = 0;
    ESS_HOUTLINE_T   hOutline;
    ESS_OBJDEF_T   Object;
    ESS_APPNAME_T   szAppName;
    ESS_DBNAME_T   szDbName;
    ESS_OBJNAME_T   szFileName;
    ESS_HMEMBER_T   hMember = ESS_NULL;

    memset(&Object, '\0', sizeof(Object));
    Object.hCtx =   hCtx;
    Object.ObjType =  ESS_OBJTYPE_OUTLINE;
    strcpy(szAppName, "Sample");
    strcpy(szDbName, "Basic");
    strcpy(szFileName, "Basic");
    Object.AppName =  szAppName;
    Object.DbName =   szDbName;
    Object.FileName =  szFileName;
    ESS_HMEMBER_T   hMember = ESS_NULL;
```
sts = EssOtlOpenOutline(hCtx, &Object, ESS_TRUE, ESS_TRUE, &hOutline);

if (!sts)
{
    sts = EssOtlFindMember(hOutline, "[Diet].[100-10]", &hMember);
}

if (!sts && hMember)
{
    sts = EssOtlSetOriginalMember(hOutline, hMember, "[100].[100-10]");
}

return sts;
}

See Also
● EssOtlGetOriginalMember

EssOtlSetOutlineInfo
Sets outline information.

Syntax

ESS_FUNC_M EssOtlSetOutlineInfo (hOutline, pInfo);

Parameter Data Type Description
hOutline ESS_HOUTLINE_T Outline context handle.
pInfo "ESS_OUTLINEINFO_T" on page 706 Pointer to a structure allocated by the caller to store outline information.

Notes
● Only some of the fields of the "ESS_OUTLINEINFO_T" on page 706 structure are used to set information. See the structure description for more information.
● If the fCaseSensitive flag in the ESS_OUTLINEINFO_T structure is being changed from ESS_TRUE to ESS_FALSE, and this causes duplicate member names, the call will fail. If your outline is a duplicate member name outline, use EssOtlSetOutlineInfoEx instead of this function.

Return Value
Returns 0 if successful; otherwise, returns an error.

Example
#include <essapi.h>
#include <essotl.h>

ESS_STS_T sts = 0;
ESS_FALSE = 0;
ESS_TRUE = 1;
ESS_POUTLINEINFO_T pInfo = ESS_NULL;
ESS_HOUTLINE_T hOutline;
ESS_OBJDEF_T Object;
ESS_APPNAME_T szAppName;
ESS_DBNAME_T szDbName;
ESS_OBJNAME_T szFileName;
memset(&Object, '\0', sizeof(Object));
Object.hCtx = hCtx;
Object.ObjType = ESS_OBJTYPE_OUTLINE;
strcpy(szAppName, "Sample");
strcpy(szDbName, "Basic");
strcpy(szFileName, "Basic");
Object.AppName = szAppName;
Object.DbName = szDbName;
Object.FileName = szFileName;

sts = EssOtlOpenOutline(hCtx, &Object, ESS_TRUE, &hOutline);

if (!sts)
{
    sts = EssOtlGetOutlineInfo(hOutline, &pInfo);
}

if (!sts && Info)
{
    pInfo->fCaseSensitive = ESS_FALSE;
    sts = EssOtlSetOutlineInfo(hOutline, pInfo);
}

if (pInfo)
{
    EssFree(hInst, pInfo);
}

See Also
● EssOtlGetOutlineInfo
● EssOtlSetOutlineInfoEx

**EssOtlSetOutlineInfoEx**

Converts a unique-member-name outline to an outline that allows duplicate names.

When pInfo->fNonUniqueName is set to TRUE, this function converts a unique member name outline to an outline allowing duplicate member names. You cannot convert an outline allowing duplicate member names back to a unique member name outline.

**Syntax**

ESS_FUNC_M EssOtlSetOutlineInfoEx (hOutline, pInfo);
### Parameter | Data Type | Description
--- | --- | ---
**hOutline** | ESS_HOUTLINE_T | Outline context handle (input).  
**pInfo** | “ESS_OUTLINEINFO_T” on page 706 | Pointer to a structure allocated by the caller to store outline information (input).

### Notes
Only some of the fields of the “ESS_OUTLINEINFO_T” on page 706 structure are used to set information. See the structure description for more information.

### Return Value
Returns 0 if successful; otherwise, see the Outline API “C Outline API Error Return Values” on page 685.

### Example
```c
void SetOutlineInfoEx()
{
    ESS_STS_T                      sts = 0;
    ESS_POUTLINEINFO_T     pInfo = ESS_NULL;
    ESS_HOUTLINE_T             hOutline;
    ESS_OBJDEF_T                Object;
    ESS_APPNAME_T             szAppName;
    ESS_DBNAME_T               szDbName;
    ESS_OBJNAME_T              szFileName;
    memset(&Object, '\0', sizeof(Object));
    Object.hCtx = hCtx;
    Object.ObjType = ESS_OBJTYPE_OUTLINE;
    strcpy(szAppName, "Sample");
    strcpy(szDbName, "Basic");
    strcpy(szFileName, "Basic");
    Object.AppName = szAppName;
    Object.DbName = szDbName;
    Object.FileName = szFileName;
    sts = EssOtlOpenOutline(hCtx, &Object, ESS_TRUE, ESS_TRUE, &hOutline);

    if (!sts)
    {
        sts = EssOtlGetOutlineInfo(hOutline, &pInfo);
    }

    if (!sts && pInfo)
    {
        pInfo->fNonUniqueName = ESS_TRUE;
        sts = EssOtlSetOutlineInfoEx(hOutline, pInfo);
    }

    if (!sts)
    {
        sts = EssOtlWriteOutline(hOutline, &Object);
    }
}
```
if (!sts) {
    sts = EssOt1Restructure(hCtx, ESS_DOR_ALLDATA);
}

if (pInfo) {
    EssFree(hInst, pInfo);
}

See Also

- EssOt1GetOutlineInfo

**EssOt1SetQueryHint**

Changes the contents (pMemberArray) of an existing query hint; applies only to Release 9.3 or higher aggregate storage databases.

**Syntax**

```c
ESS_FUNC_M EssOt1SetQueryHint (hOutline, hintNum, numMembers, pMemberArray);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline context handle (input).</td>
</tr>
<tr>
<td>hintNum</td>
<td>ESS_SHORT_T</td>
<td>Query hint number (input).</td>
</tr>
<tr>
<td>numMembers</td>
<td>ESS_SHORT_T</td>
<td>Number of members in the array provided - usually the number of real dimensions in the outline (input)</td>
</tr>
<tr>
<td>pMemberArray</td>
<td>ESS_PHMEMBER_T</td>
<td>An array of members for the hint. Usually the array has one member per real dimension, with NULL used for dimensions that are not part of the hint. This array needs to be allocated. (Input)</td>
</tr>
</tbody>
</table>

**Notes**

- Level usage constraints override query hints whenever a conflict occurs (see SetAggLevelUsage).
- Hints may not contain dynamic, label-only, or shared members.
- Hints may become invalid when the outline changes. Invalid hints result in a warning message.
- Hints enable you to influence normal view selection by informing Essbase about the profile of common queries.
- Hints are written as MDX tuples, with no more than one member from each dimension specified.
- Each member used in the query hint is considered a representative member. Essbase Server interprets representative members as "this member or any member at the similar level of aggregation." For example, using a query hint of (Qtr1, Sales, 100, East, Actual) on Sample Basic means that quarterly, actual profit margin measures for level 1 products at level 1 markets is a common type of query.
- For any given dimension, Essbase interprets the omission of representative members to mean that any member from the dimension may be used in a query. For example, using a query hint of (Sales, 100, East) on Sample Basic means that profit margin measures for level 1 products at level 1 markets is a common type of query, regardless of Year and Scenario dimensions, which were omitted. The hint (Sales, 100, East) is treated as identical to (NULL, Sales, 100, East, NULL).

**Return Value**

Returns 0 if successful.

**Example**

```c
ESS_STS_T     sts = ESS_STS_NOERR;
ESS_HOUTLINE_T hOutline = ESS_NULL;
ESS_SHORT_T   nmHints = 0;
ESS_SHORT_T   i, j, hintNum;
ESS_HMEMBER_T hMember[10]; /* (nm real dimensions) < 10 */

/* clear array just to be safe */
memset(hMember, 0x00, 10*sizeof(ESS_HMEMBER_T));

/* Code to assign hOutline variable omitted */
/* Code to assign hintNum variable omitted */

sts = EssOtlGetNumQueryHints(hOutline, &nmHints);
if (sts) return sts; /* error out */

if (hintNum <= nmHints)
{
    sts = EssOtlGetQueryHint(hOutline, hintNum, 10, hMember);
    if (sts) return sts; /* error out */

    for (j = 0; j < 10; j++)
    {
        /* Code to inspect and change hMember[j] omitted */
    }

    sts = EssOtlSetQueryHint(hOutline, hintNum, 10, hMember);
    if (sts) return sts; /* error out */
    printf("Query-Hint number: (%d) updated\n", hintNum);
}
else
{
    printf("Query-Hint number: (%d) does not exist\n", hintNum);
}
```
EssOtlSetUserAttribute

Sets a user-defined attribute for a member.

Syntax

```c
ESS_FUNC_M EssOtlSetUserAttribute(hOutline, hMember, pszString);
```

Parameter | Data Type | Description |
--- | --- | --- |
```
hOutline  ESS_HOUTLINE_T  Outline context handle.
```
```
hMember  ESS_HMEMBER_T  Handle of member for which to set the user-defined attribute.
```
```
pszString  ESS_STR_T  User-defined attribute to set.
```

Notes

- A caller can set any number of user-defined attributes for a member. The string passed in uniquely defines each attribute and follows the same conventions as user names. See EssOtlGetUserAttributes.
- Attempting to set a user attribute for a shared member generates an error.

Return Value

Returns 0 if successful; otherwise one of the following:

- OTLAPI_BAD_USERATTR
- OTLAPI_ERR_SHAREUDA

Example

```c
#include <essapi.h>
#include <essotl.h>

ESS_STS_T          sts = 0;
ESS_HOUTLINE_T     hOutline;
ESS_OBJDEF_T       Object;
ESS_APPNAME_T      szAppName;
ESS_DBNAME_T       szDbName;
ESS_OBJNAME_T      szObjName;
ESS_HMEMBER_T      hMember;
ESS_STR_T          AttributeList;

memset(&Object, '\0', sizeof(Object));
Object.hCtx = hCtx;
Object.ObjType = ESS_OBJTYPE_OUTLINE;
```
strcpy(szAppName, "Sample");
strcpy(szDbName, "Basic");
strcpy(szFileName, "Basic");
Object.AppName = szAppName;
Object.DbName = szDbName;
Object.FileName = szFileName;

sts = EssOtlOpenOutline(hCtx, &Object, ESS_TRUE,
            ESS_TRUE, &hOutline);

/***********  Set User Attributes ***********/
AttributeList = "Read Write";

if (!sts)
{
    sts = EssOtlFindMember(hOutline, "Jan",
             &hMember);
}

if (!sts && hMember)
{
    sts = EssOtlSetUserAttribute(hOutline, hMember,
                        AttributeList);
}

See Also
● EssOtlDeleteUserAttribute
● EssOtlGetUserAttributes

**EssOtlSortChildren**
Sorts the children of an outline member.

**Syntax**

```c
ESS_FUNC_M EssOtlSortChildren (hOutline, hParent, usType, fpCompare, pUserData);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline context handle.</td>
</tr>
<tr>
<td>hParent</td>
<td>ESS_HMEMBER_T</td>
<td>Handle of parent of the children to sort. If this is ESS_NULL, the dimensions are sorted.</td>
</tr>
<tr>
<td>usType</td>
<td>ESS_USHORT_T</td>
<td>Sort type. This can be one of the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESS_SORT_ASCENDING</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESS_SORT_DESCENDING</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESS_SORT_USERDEFINED</td>
</tr>
</tbody>
</table>
**Parameter** | **Data Type** | **Description**
---|---|---
fpCompare | ESS_POTLSORTFUNC_T | Pointer to user-defined comparison function. This is only used if the `usType` parameter is ESS_SORT_USERDEFINED. It points to a function defined as follows:

```c
(ESS_INTFUNC_M Compare
ESS_HMEMBER_T mbr1,
ESS_HMEMBER_T mbr2,
ESS_PVOID_T pUserData);
```

The function accepts handles for two members and should return the following:

- `< 0` if `mbr1` goes before `mbr2`.
- `= 0` if `mbr1` is equivalent to `mbr2`.
- `> 0` if `mbr1` goes after `mbr2`.

pUserData | ESS_PVOID_T | Pointer to any user-specified data. This is only used if the `usType` parameter is ESS_SORT_USERDEFINED. Each time the comparison function is called, the value of this parameter is passed into the comparison function.

**Notes**

During the callback function, you should not call any outline functions that might change the outline. Only `EssOtlGetMemberInfo()`, `EssOtlGetMemberFormula()`, and `EssOtlGetMemberAlias()` can be called.

**Return Value**

Returns 0 if successful; otherwise one of the following:

- OTLAPI_BAD_SORTTYPE
- OTLAPI_BAD_SORTCOMPAREFUNC
- OTLAPI_SORT_TOOMANY

**Example**

```c
#include <essapi.h>
#include <essotl.h>

ESS_STS_T          sts = 0;
ESS_HOUTLINE_T     hOutline;
ESS_HMEMBER_T      hMeasures;
FARPROC            pfnSort;
ESS_OBJDEF_T       Object;
ESS_APPNAME_T      szAppName;
ESS_DBNAME_T       szDbName;
ESS_OBJNAME_T      szFileName;
memset(&Object, '\0', sizeof(Object));
Object.hCtx = hCtx;
Object.ObjType = ESS_OBJTYPE_OUTLINE;
strcpy(szAppName, "Sample");
strcpy(szDbName, "Basic");
strcpy(szFileName, "Basic");
Object.AppName = szAppName;
```

Object.DbName = szDbName;
Object.FileName = szFileName;

sts = EssOtlOpenOutline(hCtx, &Object, ESS_TRUE, 
ESS_TRUE, &hOutline);

if (!sts)
{
    sts = EssOtlFindMember(hOutline, "Measures", 
&hMeasures);
}

if (!sts)
{
    sts = EssOtlSortChildren(hOutline, hMeasures, 
ESS_SORT_USERDEFINED, 
(ESS_POTLSORTFUNC_T)pfnSort, 
(ESS_PVOID_T)hOutline);
}

/***********************************************/
int  ESS_INTFUNCT_M SortCompare ( 
ESS_HMEMBER_T hMember1, 
ESS_HMEMBER_T hMember2, 
ESS_PVOID_T pData)
{
    int nRet = 0;
    int nLen1;
    int nLen2;
    ESS_STS_T sts = 0;
    ESS_PMBRINFO_T pMbrInfo1 = ESS_NULL;
    ESS_PMBRINFO_T pMbrInfo2 = ESS_NULL;
    ESS_HOUTLINE_T hOutline = 
(ESS_HOUTLINE_T)pData;

    sts = EssOtlGetMemberInfo(hOutline, hMember1, 
&pMbrInfo1);
    if (!sts &amp; pMbrInfo1)
    sts = EssOtlGetMemberInfo(hOutline, 
    hMember2, &pMbrInfo2);

    if (!sts &amp; pMbrInfo2)
    {
        nLen1 = strlen(pMbrInfo1-&gt;szMember);
        nLen2 = strlen(pMbrInfo2-&gt;szMember);
        if (nLen1 &lt; nLen2)
            nRet = -1;
        else if (nLen1 &gt; nLen2)
            nRet = 1;
    }

    if (pMbrInfo1)
    {
        EssFree(hInst, pMbrInfo1);
    }
if (pMbrInfo2)
{
    EssFree(hInst, pMbrInfo2);
}
return (nRet);

See Also

- EssOtlFindMember

**EssOtlVaryingAssociateAttribute**

Associates an attribute member to a base member, with the validity of the associations specified by the given validity set.

**Syntax**

```c
ESS_FUNC_M EssOtlVaryingAssociateAttribute (hOutline, hBaseMember, hAttrMember, mode, pValiditySet);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline context handle</td>
</tr>
<tr>
<td>hBaseMember</td>
<td>ESS_HMEMBER_T</td>
<td>Handle to the base member</td>
</tr>
<tr>
<td>hAttrMember</td>
<td>ESS_HMEMBER_T</td>
<td>Handle to the attribute member</td>
</tr>
<tr>
<td>mode</td>
<td>ESS_INT_T</td>
<td>Association mode. Possible values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- MODE_OVERWRITE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- MODE_NOOVERWRITE</td>
</tr>
<tr>
<td>pValiditySet</td>
<td>“ESS_VALIDITYSET_T”</td>
<td>Pointer to the validity set that defines the independent dimension for which the association is valid</td>
</tr>
</tbody>
</table>

**Notes**

- When a full range is specified, the association is made as specified.
- Association mode determines how to handle open-ended situations when only the starting tuple is specified instead of a full range. (For these description examples, assume the following associations of the Ounces attribute members associated with product 100-10 prior to association):

<table>
<thead>
<tr>
<th></th>
<th>Jan</th>
<th>Feb</th>
<th>Mar</th>
<th>Apr</th>
<th>May</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>12</td>
<td>12</td>
<td>16</td>
<td>16</td>
<td>20</td>
</tr>
</tbody>
</table>

- MODE_OVERWRITE—Association starts from the specified member through all members that follow it. Example: Associating 12 starting with Mar.

Result: Mar and all members after it are associated with 12. Conflicting associations are overwritten:

<table>
<thead>
<tr>
<th></th>
<th>Jan</th>
<th>Feb</th>
<th>Mar</th>
<th>Apr</th>
<th>May</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
</tr>
</tbody>
</table>
- **MODE_NOOVERWRITE**—Association starts at the specified tuple and continues until the existing associated attribute member is different. Example: Associating 12 starting with Mar.

Result: Both Mar and Apr had the same attribute so association continues until May where it was different:

<table>
<thead>
<tr>
<th>Jan</th>
<th>Feb</th>
<th>Mar</th>
<th>Apr</th>
<th>May</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>20</td>
</tr>
</tbody>
</table>

**Return Value**

Returns 0 if successful.

**Example**

```c
void TestEssOtlVaryingAssociateAttribute()
{
    ESS_STS_T    sts = ESS_STS_NOERR;
    ESS_HOUTLINE_T    hOutline = ESS_NULL;
    ESS_OBJDEF_T    Object;
    ESS_HMEMBER_T    hBaseMbr, hAttrMbr, hBaseDim, hAttrDim, hIndDim = ESS_NULL;
    ESS_INT_T    mode;
    ESS_VALIDITYSET_T ValiditySet;
    ESS_HMEMBER_T IndDimsArray[2];
    ESS_STR_T    IndepMbrsArray[4];
    ESS_INT32_T    countOfIndDims;
    ESS_USHORT_T    usValiditySetType;
    ESS_UCHAR_T    pucIndependentTypes[2];

    memset(&Object, '\0', sizeof(Object));
    Object.hCtx = hCtx;
    Object.ObjType = ESS_OBJTYPE_OUTLINE;
    Object.AppName = szAppName;
    Object.DbName = szDbName;
    Object.FileName = szDbName;
    printf("\n");
    sts = EssOtlOpenOutline(hCtx, &Object, ESS_TRUE, ESS_TRUE, &hOutline);
    printf("EssOtlOpenOutline sts: %ld", sts);

    /* Disassociate base dimension from attribute dimension before test.*/
    printf("\nDisassociate base dimension from attribute dimension before test: \n");
    sts = EssOtlFindMember(hOutline, "Entities", &hBaseDim);
    printf("\nEssOtlFindMember hBaseDim: %ld
", sts);
    sts = EssOtlFindMember(hOutline, "Type", &hAttrDim);
    printf("\nEssOtlFindMember hAttrDim: %ld
", sts);
    sts = EssOtlDisassociateAttributeDimension(hOutline, hBaseDim, hAttrDim);
    printf("EssOtlDisassociateAttributeDimension: %ld
", sts);

    /* Get handle for base member*/
    printf("\nGet handle for base member:\n");
    sts = EssOtlFindMember(hOutline, "Doe, Jane", &hBaseMbr);
    printf("EssOtlFindMember hBaseMbr: %ld
", sts);

    /* Get handle for indep dimensions*/
    printf("\nGet handle for indep dimensions:\n");
}
```
sts = EssOtlFindMember(hOutline, "Time Periods", &IndDimsArray[0]);
printf("EssOtlFindMember sts: %d\n", sts);
st = EssOtlFindMember(hOutline, "Years", &IndDimsArray[1]);
printf("EssOtlFindMember sts: %d\n", sts);

/* Associate the dimension Entities and Type*/
printf("\nAssociate the dimensions:\n");
countOfIndDims = 2;
pucIndependentTypes[0] = ESS_ASSOCIATE_TYPE_DISCRETE;
pucIndependentTypes[1] = ESS_ASSOCIATE_TYPE_CONTINUOUS;
st = EssOtlVaryingAssociateAttributeDimension(hOutline, hBaseDim, hAttrDim,
countOfIndDims, IndDimsArray, pucIndependentTypes);
printf("EssOtlVaryingAssociateAttributeDimension sts: %d\n", sts);

/* Initial valid case with ValiditySetType of member handles*/
printf("\n*** Initial valid case with ValiditySetType of member handles ***\n");
printf("\nGet handle for attribute member: \n");
st = EssOtlFindMember(hOutline, "Regular", &hAttrMbr);
printf("EssOtlFindMember sts: %d\n", sts);

sts = EssOtlFindMember(hOutline, "Jan", &hIndepMbrsArray[0]);
st = EssOtlFindMember(hOutline, "FY03", &hIndepMbrsArray[1]);
st = EssOtlFindMember(hOutline, "Jan", &hIndepMbrsArray[2]);
st = EssOtlFindMember(hOutline, "FY06", &hIndepMbrsArray[3]);
memset(&ValiditySet, '\0', sizeof(ValiditySet));
ValiditySet.countOfIndepDims = 2;
ValiditySet.usValiditySetType = ESS_VALIDITYSET_TYPE_MBRHDLS;
ValiditySet.countOfIndepRanges = 1;
ValiditySet.pIndepMbrs = hIndepMbrsArray;

printf("\nBefore association: \n");
usValiditySetType = ESS_VALIDITYSET_TYPE_MBRHDLS;
DisplayVaryingAttributes(hOutline, hBaseMbr, hAttrDim, ESS_NULL, usValiditySetType);
mode = ESS_ASSOCIATE_MODE_NOOVERWRITE;
st = EssOtlVaryingAssociateAttribute(hOutline, hBaseMbr, hAttrMbr, mode,
&ValiditySet);
printf("EssOtlVaryingAssociateAttribute sts: %d\n", sts);

/* Restructure and save outline */
SaveOutline(hOutline);

sts = EssUnlockObject(hCtx, ESS_OBJTYPE_OUTLINE, Object.AppName, Object.DbName,
Object.FileName);
printf("\nEssUnlockObject sts: %d\n", sts);

sts = EssOtlCloseOutline(hOutline);
printf("EssOtlCloseOutline sts: %d\n", sts);
}

See Also
- EssOtlQueryVaryingAttributes
- EssOtlVaryingAssociateAttributeDimension
- EssOtlVaryingDisassociateAttribute
- EssOtlVaryingGetAssociatedAttributes
EssOt1VaryingAssociateAttributeDimension

Associates a base dimension with an attribute dimension and defines the base dimension as having varying attributes. Member attribute associations vary depending on the level-0 members of the given independent dimensions, of types specified in `pucIndependentTypes`. Continuous independent dimensions must be specified last.

**Syntax**

```c
ESS_FUNC_M EssOt1VaryingAssociateAttributeDimension (hOutline, hBaseDim, hAttrDim, countOfIndepDims, *pIndepDims, *pucIndependentTypes);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline context handle</td>
</tr>
<tr>
<td>hBaseDim</td>
<td>ESS_HMEMBER_T</td>
<td>Base dimension handle</td>
</tr>
<tr>
<td>hAttrDim</td>
<td>ESS_HMEMBER_T</td>
<td>Attribute dimension handle</td>
</tr>
<tr>
<td>countOfIndepDims</td>
<td>ESS_INT32_T</td>
<td>The number of independent dimensions that control the varying attribute</td>
</tr>
<tr>
<td>*pIndepDims</td>
<td>ESS_HMEMBER_T</td>
<td>Pointer to an array of member handles for the independent dimensions</td>
</tr>
<tr>
<td>*pucIndependentTypes</td>
<td>ESS_UCHAR_T</td>
<td>Pointer to an array of independent types contained in *pIndepDims. The independent types supported are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESS_ASSOCIATE_TYPE_DISCRETE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESS_ASSOCIATE_TYPE_CONTINUOUS</td>
</tr>
</tbody>
</table>

**Return Value**

Returns 0 if successful.

**Example**

```c
void TestEssOt1VaryingAssociateAttributeDimension()
{
    ESS_STS_T     sts = ESS_STS_NOERR;
    ESS_HOUTLINE_T hOutline = ESS_NULL;
    ESS_OBJDEF_T  Object;
    ESS_HMEMBER_T hBaseDim = ESS_NULL;
    ESS_HMEMBER_T IndDimsArray[2];
    ESS_HMEMBER_T hAttrDimArray[9];
    ESS_INT32_T   countOfIndDims;
    ESS_UCHAR_T   pucIndependentTypes[2];
    int           i;

    memset(&Object, '\0', sizeof(Object));
    Object.hCtx = hCtx;
    Object.ObjType = ESS_OBJTYPE_OUTLINE;
    Object.AppName = szAppName;
```
Object.DbName = szDbName;
Object.FileName = szDbName;

printf("\n");
sts = EssOtlOpenOutline(hCtx, &Object, ESS_TRUE, ESS_TRUE, &hOutline);
printf("EssOtlOpenOutline sts: %ld\n",sts);

SetupTest(hOutline);

/* Assign independent dimension array*/
sts = EssOtlFindMember(hOutline, "Time Periods", &IndDimsArray[0]);
printf("EssOtlFindMember sts: %d\n", sts);
sts = EssOtlFindMember(hOutline, "Years", &IndDimsArray[1]);
printf("EssOtlFindMember sts: %d\n", sts);

/* Get handles to base and attribute dimensions for test.*/
sts = EssOtlFindMember(hOutline, "Entities", &hBaseDim);
printf("\nEssOtlFindMember sts: %d\n", sts);
sts = EssOtlFindMember(hOutline, "Type", &hAttrDimArray[0]);
printf("EssOtlFindMember sts: %d\n", sts);
sts = EssOtlFindMember(hOutline, "FT/PT", &hAttrDimArray[1]);
printf("EssOtlFindMember sts: %d\n", sts);

/* Disassociate current association before tests.*/
for (i = 0; i < 2; i++)
{
    sts = EssOtlDisassociateAttributeDimension(hOutline, hBaseDim,
    hAttrDimArray[i]);
    printf("EssOtlDisassociateAttributeDimension sts: %d\n", sts);
}

/* Associate the dimension Entities to Type*/
printf("\nValid case: Associate the dimension Entities and Type:\n");
countOfIndDims = 2;
pucIndependentTypes[0] = ESS_ASSOCIATE_TYPE_DISCRETE;
pucIndependentTypes[1] = ESS_ASSOCIATE_TYPE_CONTINUOUS;
sts = EssOtlVaryingAssociateAttributeDimension(hOutline, hBaseDim, hAttrDimArray[0],
countOfIndDims, IndDimsArray, pucIndependentTypes);
printf("EssOtlVaryingAssociateAttributeDimension sts: %d\n", sts);

sts = EssUnlockObject(hCtx, ESS_OBJTYPE_OUTLINE, Object.AppName, Object.DbName,
Object.FileName);
printf("\nEssUnlockObject sts: %d\n", sts);

sts = EssOtlCloseOutline(hOutline);
printf("EssOtlCloseOutline sts: %d\n", sts);

See Also

- EssOtlQueryVaryingAttributes
- EssOtlVaryingAssociateAttribute
- EssOtlVaryingDisassociateAttribute
- EssOtlVaryingGetAssociatedAttributes
- EssOtlVaryingGetAttributeIndepDims
**EssOtI Varying Disassociate Attribute**

Disassociates the attribute members in the given attribute dimension from the specified base member. The given validity set specifies where the disassociations should occur.

**Syntax**

```c
ESS_FUNC_M EssOtI VaryingDisassociateAttribute (hOutline, hBaseMember, hAttrDim, mode, pValiditySet);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline context handle</td>
</tr>
<tr>
<td>hBaseMember</td>
<td>ESS_HMEMBER_T</td>
<td>Base member handle</td>
</tr>
<tr>
<td>hAttrDim</td>
<td>ESS_HMEMBER_T</td>
<td>Attribute member handle</td>
</tr>
<tr>
<td>mode</td>
<td>ESS_INT_T</td>
<td>Association mode. Possible values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• MODE_OVERWRITE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• MODE_NOOVERWRITE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• MODE_EXTEND</td>
</tr>
<tr>
<td>pValiditySet</td>
<td>&quot;ESS_VALIDITYSET_T&quot; on page 710</td>
<td>Pointer to the set of independent members where the disassociation occurs</td>
</tr>
</tbody>
</table>

**Notes**

- This function removes attribute associations from the specified base members.
- When a full range is specified, the disassociation is made as specified.
- Association mode determines how to handle situations when only the starting tuple is specified instead of a full range. (For these description examples, assume the following associations of members of the Ounces attribute dimension with product 100-10 prior to disassociation):

<table>
<thead>
<tr>
<th></th>
<th>Jan</th>
<th>Feb</th>
<th>Mar</th>
<th>Apr</th>
<th>May</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>12</td>
<td>12</td>
<td>16</td>
<td>16</td>
<td>12</td>
</tr>
</tbody>
</table>

  - **MODE_OVERWRITE**: Disassociation starts from the specified member through all members that follow it. Example: Disassociation starting with Feb.
    
    Result: Associations are removed from Feb and all members after it:

    |   | Jan | Feb | Mar | Apr | May |
    |---|-----|-----|-----|-----|-----|
    |   | 12  | 12  | 16  | 16  | 12  |

  - **MODE_NOOVERWRITE**: Disassociation starts at the specified tuple and continues until the existing associated attribute member is different. Example: Disassociation starting with Mar.
    
    Result: Since Mar also has attribute 16, both the Apr and Mar associations are removed:

    |   | Jan | Feb | Mar | Apr | May |
    |---|-----|-----|-----|-----|-----|
    |   | 12  | 12  | 12  | 12  | 12  |
MODE_EXTEND: Similar to MODE_NOOVERWRITE except that the association immediately ahead of the start tuple is extended over the disassociated member.
Example: Disassociation starting with Mar

Result: The Mar and Apr associations are removed; they assume the association of the previous month, Feb.

<table>
<thead>
<tr>
<th>Jan</th>
<th>Feb</th>
<th>Mar</th>
<th>Apr</th>
<th>May</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
</tr>
</tbody>
</table>

Return Value

Returns 0 if successful.

Example

```c
void TestEssOtlVaryingDisassociateAttribute()
{
    ESS_STS_T   sts = ESS_STS_NOERR;
    ESS_HOUTLINE_T  hOutline = ESS_NULL;
    ESS_OBJDEF_T   Object;
    ESS_HMEMBER_T  hBaseMbr, hAttrMbr, hBaseDim, hAttrDim, hIndDim = ESS_NULL;
    ESS_INT_T     mode;
    ESS_VALIDITYSET_T ValiditySet;
    ESS_HMEMBER_T  IndDimsArray[2];
    ESS_HMEMBER_T  hIndepMbrsArray[4];
    ESS_STR_T      IndepMbrsArray[4];
    ESS_INT32_T    countOfIndDims;
    ESS_USHORT_T   usValiditySetType;
    ESS_UCHAR_T    pucIndependentTypes[2];

    memset(&Object, '\0', sizeof(Object));
    Object.hCtx = hCtx;
    Object.ObjType = ESS_OBJTYPE_OUTLINE;
    Object.AppName = szAppName;
    Object.DbName = szDbName;
    Object.FileName = szDbName;

    printf("\n");
    sts = EssOtlOpenOutline(hCtx, &Object, ESS_TRUE, ESS_TRUE, &hOutline);
    printf("EssOtlOpenOutline sts: %ld\n",sts);

    /* Disassociate base dimension from attribute dimension before test.*/
    printf("\nDisassociate base dimension from attribute dimension before test: ");
    sts = EssOtlFindMember(hOutline, "Entities", &hBaseDim);
    printf("\nEssOtlFindMember sts: %d\n",sts);
    sts = EssOtlFindMember(hOutline, "Type", &hAttrDim);
    printf("\nEssOtlFindMember sts: %d\n",sts);
    sts = EssOtlDisassociateAttributeDimension(hOutline, hBaseDim, hAttrDim);
    printf("EssOtlDisassociateAttributeDimension sts: %d\n",sts);

    /* Get handle for base member*/
    printf("\nGet handle for base member: \n");
    sts = EssOtlFindMember(hOutline, "Doe,Jane", &hBaseMbr);
    printf("EssOtlFindMember sts: %d\n",sts);

    /* Get handle for indep dimensions*/
```
printf(\"\nGet handle for indep dimensions:\n\");  
sts = EssOtlFindMember(hOutline, "Time Periods", &IndDimsArray[0]);  
printf(\"EssOtlFindMember sts: %d\n", sts);  
sts = EssOtlFindMember(hOutline, "Years", &IndDimsArray[1]);  
printf(\"EssOtlFindMember sts: %d\n", sts);  
/* Associate the dimension Entities and Type*/  
printf(\"\nAssociate the dimensions:\n\");  
countOfIndDims = 2;  
pucIndependentTypes[0] = ESS_ASSOCIATE_TYPE_DISCRETE;  
pucIndependentTypes[1] = ESS_ASSOCIATE_TYPE_CONTINUOUS;  
sts = EssOtlVaryingAssociateAttributeDimension(hOutline, hBaseDim, hAttrDim,  
countOfIndDims, IndDimsArray, pucIndependentTypes);  
printf(\"EssOtlVaryingAssociateAttributeDimension sts: %d\n", sts);  
/* Initial valid case with ValiditySetType of member handles*/  
printf(\"\n*** Initial valid case with ValiditySetType of member handles ***\n\");  
sts = EssOtlFindMember(hOutline, "Regular", &hAttrMbr);  
printf(\"EssOtlFindMember sts: %d\n", sts);  
sts = EssOtlFindMember(hOutline, "Jan", &hIndepMbrsArray[0]);  
sts = EssOtlFindMember(hOutline, "FY03", &hIndepMbrsArray[1]);  
sts = EssOtlFindMember(hOutline, "Jan", &hIndepMbrsArray[2]);  
sts = EssOtlFindMember(hOutline, "FY06", &hIndepMbrsArray[3]);  
memset(&ValiditySet, '\0', sizeof(ValiditySet));  
ValiditySet.countOfIndepDims = 2;  
ValiditySet.usValiditySetType = ESS_VALIDITYSET_TYPE_MBRHDLS;  
ValiditySet.countOfIndepRanges = 1;  
ValiditySet.pIndepMbrs = hIndepMbrsArray;  
printf(\"\nBefore association:\n\");  
usValiditySetType = ESS_VALIDITYSET_TYPE_MBRHDLS;  
DisplayVaryingAttributes(hOutline, hBaseMbr, hAttrDim, ESS_NULL, usValiditySetType);  
mode = ESS_ASSOCIATE_MODE_NOOVERWRITE;  
sts = EssOtlVaryingAssociateAttribute(hOutline, hBaseMbr, hAttrDim, mode,  
&ValiditySet);  
printf(\"EssOtlVaryingAssociateAttribute sts: %d\n", sts);  
/* Disassociation */  
IndepMbrsArray[0] = \"\";  
IndepMbrsArray[1] = \"\";  
IndepMbrsArray[2] = \"\";  
IndepMbrsArray[3] = \"\";  
memset(&ValiditySet, '\0', sizeof(ValiditySet));  
ValiditySet.countOfIndepDims = 2;  
ValiditySet.usValiditySetType = ESS_VALIDITYSET_TYPE_MBRNAMS;  
ValiditySet.countOfIndepRanges = 1;  
ValiditySet.pIndepMbrs = IndepMbrsArray;  
mode = ESS_DISASSOCIATE_MODE_NOOVERWRITE;  
sts = EssOtlVaryingDisassociateAttribute(hOutline, hBaseMbr, hAttrDim, mode,  
&ValiditySet);  
printf(\"EssOtlVaryingDisassociateAttribute sts: %d\n", sts);  

displayVaryingAttributes(\"\");  
exit(0);
/* Restructure and save outline */
SaveOutline(hOutline);

sts = EssUnlockObject(hCtx, ESS_OBJTYPE_OUTLINE, Object.AppName, Object.DbName,
Object.FileName);
printf("\nEssUnlockObject sts: %d\n", sts);

sts = EssOtlCloseOutline(hOutline);
printf("EssOtlCloseOutline sts: %d\n",sts);
}

See Also
- EssOtlQueryVaryingAttributes
- EssOtlVaryingAssociateAttribute
- EssOtlVaryingAssociateAttributeDimension
- EssOtlVaryingGetAssociatedAttributes
- EssOtlVaryingGetAttributeIndepDims

**EssOtlVaryingGetAssociatedAttributes**

Returns the attribute members in the specified attribute dimension associated with the given base member where the association validity includes at least one of the tuples in the given perspective.

For each qualifying attribute member, the full validity set of its association to the base member can be optionally returned (by specifying a non-null value for *pppValiditySets*).

*pphMembers* and *pppValiditySets* will contain the array of member handles and array of validity set pointers, respectively.

The type of validity set desired is indicated using *usValiditySetType*.

Note that the perspective must specify discrete independent members individually.

If no perspective is specified, or if the perspective specifies a NULL set of independent members, the routine will consider all associations that exist for any combination of independent members. In this case, the returned validity sets may contain ranges for discrete independent members, and it is the responsibility of the client to split these accordingly.

**Syntax**

```c
ESS_FUNC_M EssOtlVaryingGetAssociatedAttributes (hOutline, hBaseMember, hAttrDim,
pPerspective, pusCount, pphMembers, usValiditySetType, **pppValiditySets);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline context handle</td>
</tr>
<tr>
<td>hBaseMember</td>
<td>ESS_HMEMBER_T</td>
<td>Handle to the member of the base dimension</td>
</tr>
<tr>
<td>hAttrDim</td>
<td>ESS_HMEMBER_T</td>
<td>Attribute member handle</td>
</tr>
<tr>
<td>Parameter</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>pPerspective</td>
<td>“ESS_PERSPECTIVE_T” on page 708</td>
<td>Pointer to the collection of independent members used when querying the client or server for associations</td>
</tr>
<tr>
<td>pusCount</td>
<td>ESS_PUSHORT_T</td>
<td>Pointer to the number of varying attribute members returned</td>
</tr>
<tr>
<td>pphMembers</td>
<td>ESS_PPHMEMBER_T</td>
<td>Pointer to the array of attribute member handles</td>
</tr>
<tr>
<td>usValiditySetType</td>
<td>ESS_USHORT_T</td>
<td>Type of validity set assigned to independent members:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_VALIDITYSET_TYPE_MBRHDLS—As an XRange of member handles. For example, Mar 2003-Feb 2004 consists of ten months of 2003 (starting with March) and the first two months of 2004 (ending with February).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_VALIDITYSET_TYPE_MBRNAMS—As an XRange of member names</td>
</tr>
</tbody>
</table>

**pppValiditySets “ESS_VALIDITYSET_T” on page 710**  
A collection of independent members for which an association is true

**Return Value**

Returns 0 if successful.

**Example**

```c
void DisplayVaryingAttributes(ESS_HOUTLINE_T hOutline, ESS_HMEMBER_T hBaseMbr,
ESS_HMEMBER_T hAttrDim, ESS_PERSPECTIVE_T *pPerspective,
ESS_USHORT_T usValiditySetType)
{
    ESS_STS_T                sts = ESS_STS_NOERR;
    ESS_USHORT_T            Count, i, j, totalIndMbrs;
    ESS_PPHMEMBER_T            phAttrMbrs;
    ESS_PVALIDITYSET_T  *ppValiditySets;
    ESS_PMBRINFO_T            pMemberInfo1, pMemberInfo2;
    sts = EssOtlVaryingGetAssociatedAttributes(hOutline,
                                             hBaseMbr,
                                             hAttrDim,
                                             pPerspective,
                                             &Count,
                                             &phAttrMbrs,
                                             usValiditySetType,
                                             &ppValiditySets);
    printf("\n\nEssOtlVaryingGetAssociatedAttributes sts: %d", sts);
    if(!sts)
    {
        if(Count)
        {
            for (i = 0; i < Count ;++i)
            {
                sts = EssOtlGetMemberInfo(hOutline, phAttrMbrs[i], &pMemberInfo1);
                printf("\n\t%s", pMemberInfo1->szMember);
                EssOtlFreeStructure(hOutline, ESS_DT_STRUCT_MBRINFO, 1, pMemberInfo1);
                if(ppValiditySets[i])
```
totalIndMbrs = (ESS_SHORT_T)(ppValiditySets[i]->countOfIndepRanges) * 4;
printf("\n\t\tValidity Type: %d - ", ppValiditySets[i]->usValiditySetType);
switch(ppValiditySets[i]->usValiditySetType)
{
    case ESS_VALIDITYSET_TYPE_MBRHDLS:
        printf("Member Handles\n");
        for(j = 0; j < totalIndMbrs; j++)
        {
            if(j >= 3)
                if(j%4 == 0)
                    printf("\n");
            sts = EssOtlGetMemberInfo(hOutline, ppValiditySets[i]->pIndepMbrs[j], &pMemberInfo2);
            printf("\n\t\tValidity independent member: %s", pMemberInfo2->szMember);
            EssOtlFreeStructure(hOutline, ESS_DT_STRUCT_MBRINFO, 1, pMemberInfo2);
        }
        break;
    case ESS_VALIDITYSET_TYPE_MBRNAMS:
        printf("Member Names\n");
        for(j = 0; j < totalIndMbrs; j++)
        {
            if(j >= 3)
                if(j%4 == 0)
                    printf("\n");
            printf("\n\t\tValidity independent member: %s", ppValiditySets[i]->pIndepMbrs[j]);
        }
        break;
    default:
        printf("Unrecognized\n");
}
printf("\n\t\tValidity count of Indep Dims: %d", ppValiditySets[i]->countOfIndepDims);
printf("\n\t\tValidity count of Indep Ranges: %d", ppValiditySets[i]->countOfIndepRanges);
printf("\n");
EssFree(hInst, ppValiditySets[i]);
}
else
    printf("\nNo member returned.\n");
printf("\n");
EssFree(hInst, ppValiditySets);
See Also

- EssOtlQueryVaryingAttributes
- EssOtlVaryingAssociateAttribute
- EssOtlVaryingAssociateAttributeDimension
- EssOtlVaryingDisassociateAttribute
- EssOtlVaryingGetAttributeIndepDims

**EssOtlVaryingGetAttributeIndepDims**

Returns the independent dimensions, if any, for the given attribute member.

**Syntax**

```c
ESS_FUNC_M EssOtlVaryingGetAttributeIndepDims (hOutline, hAttrMember, *pCountOfIndepDims, **ppIndepDims, **ppucIndependentTypes);
```

**Parameter**

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T Outline context handle</td>
</tr>
<tr>
<td>hAttrMember</td>
<td>ESS_HMEMBER_T Attribute member handle</td>
</tr>
<tr>
<td>*pCountOfIndepDims</td>
<td>ESS_INT32_T Pointer to the number of independent dimensions that control the varying attributes</td>
</tr>
<tr>
<td>**ppIndepDims</td>
<td>ESS_HMEMBER_T Pointer to an array of member handles for the independent dimensions</td>
</tr>
<tr>
<td>**ppucIndependentTypes</td>
<td>ESS_UCHAR_T Pointer to an array of independent types contained in *pIndepDims</td>
</tr>
</tbody>
</table>

**Notes**

An independent dimension is a dimension the values of which identify where the attribute associations may change. Independent dimensions are selected when attribute dimensions are associated with a base dimension. VaryingGetAttributeIndepDims returns a list of independent dimensions associated with the dimension of the specified attribute.

**Return Value**

Returns 0 if successful.

**Example**

```c
void TestEssOtlVaryingGetAttributeIndepDims()
{
    ESS_STS_T    sts = ESS_STS_NOERR;
    ESS_HOUTLINE_T    hOutline = ESS_NULL;
    ESS_OBJDEF_T    Object;
    ESS_STR_T    attrMbr, attrDim, baseMbr, baseDim;
    ESS_USHORT_T    i;
    ESS_HMEMBER_T    hAttrMbr;
    ESS_HMEMBER_T    hAttrDim;
    ESS_HMEMBER_T    hBaseMbr;
    ESS_HMEMBER_T    hBaseDim;
```
memset(&Object, '0', sizeof(Object));
Object.hCtx = hCtx;
Object.ObjType = ESS_OBJTYPE_OUTLINE;
Object.AppName = szAppName;
Object.DbName = szDbName;
Object.FileName = szDbName;

printf("\n");
sts = EssOtlOpenOutline(hCtx, &Object, ESS_TRUE, ESS_TRUE, &hOutline);
printf("EssOtlOpenOutline sts: %ld\n", sts);

printf("\nGet handles of members for tests:\n");
attrMbr = "Contractor";
sts = EssOtlFindMember(hOutline, attrMbr, &hAttrMbr);
printf("EssOtlFindMember sts: %d\n", sts);
attrDim = "Type";
sts = EssOtlFindMember(hOutline, attrDim, &hAttrDim);
printf("EssOtlFindMember sts: %d\n", sts);
baseMbr = "Doe,Jane";
sts = EssOtlFindMember(hOutline, baseMbr, &hBaseMbr);
printf("EssOtlFindMember sts: %d\n", sts);
baseDim = "Entities";
sts = EssOtlFindMember(hOutline, baseDim, &hBaseDim);
printf("EssOtlFindMember sts: %d\n", sts);

/* Valid case with a valid attribute member handle. */
printf("\nValid case with a valid attribute member handle:\n");
sts = EssOtlVaryingGetAttributeIndepDims(hOutline, hAttrMbr, &pCountOfIndepDims,
&ppIndepDims, &pucIndependentTypes);
printf("EssOtlVaryingGetAttributeIndepDims sts: %d\n", sts);
if(pCountOfIndepDims)
{
    printf("Independent dimension(s) for attribute member %s: ", attrMbr);
    for (i = 0; i < pCountOfIndepDims; i++)
    {
        sts = EssOtlGetMemberInfo(hOutline, ppIndepDims[i], &pMemberInfo);
        printf("\n\n%s", pMemberInfo->szMember);
        switch(pucIndependentTypes[i])
        {
            case ESS_ASSOCIATE_TYPE_CONTINUOUS:
                printf(" - (Continuous)\n");
                break;
            case ESS_ASSOCIATE_TYPE_DISCRETE:
                printf(" - (Discrete)\n");
                break;
        }
    }
    printf("\n");
}
else
    printf("\tAttribute member %s has no independent dimension.\n", attrMbr);
sts = EssUnlockObject(hCtx, ESS_OBJTYPE_OUTLINE, Object.AppName, Object.DbName, Object.FileName);
printf("\nEssUnlockObject sts: %d\n", sts);

sts = EssOtlCloseOutline(hOutline);
printf("EssOtlCloseOutline sts: %d",sts);
}

See Also

- EssOtlQueryVaryingAttributes
- EssOtlVaryingAssociateAttribute
- EssOtlVaryingAssociateAttributeDimension
- EssOtlVaryingDisassociateAttribute
- EssOtlVaryingGetAssociatedAttributes

EssOtlVerifyFormula

Verifies that an outline is correct. This function returns both global outline errors and errors for each incorrect member. This function is called by EssOtlVerifyOutlineEx, but can also be called directly from a client program.

Syntax

ESS_FUNC_M EssOtlVerifyFormula (hOutline, hCtx, FormulaString, pErrorNumber, pErrorLine, MemberName, ErrorBufferLength, ErrorMessage);

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline context handle.</td>
</tr>
<tr>
<td>hCtx</td>
<td>ESS_HCTX_T</td>
<td>API context handle. If the outline is a local outline, it is necessary to also provide this separate hCtx to a running server, because formula checking only is done on the server, and outlines from the file system do not have a server connected to them. This parameter should normally be NULL.</td>
</tr>
<tr>
<td>FormulaString</td>
<td>ESS_STR_T</td>
<td>The syntactic formula expression.</td>
</tr>
<tr>
<td>pErrorNumber</td>
<td>ESS_PULONG_T</td>
<td>Pointer to the count of errors.</td>
</tr>
<tr>
<td>pErrorLine</td>
<td>ESS_PULONG_T</td>
<td>Pointer to the error line number.</td>
</tr>
<tr>
<td>MemberName</td>
<td>ESS_STR_T</td>
<td>Name of member that has the formula. This is an optional field. Supplying it will enhance the error message, especially if EssOtlVerifyFormula() is called within a loop.</td>
</tr>
<tr>
<td>ErrorBufferLength</td>
<td>ESS_ULONG_T</td>
<td>The size of the error buffer.</td>
</tr>
<tr>
<td>ErrorMessage</td>
<td>ESS_STR_T</td>
<td>The error message contained in the error buffer. This is a pre-allocated string which contains a descriptive message of any error (including error number, line number, and member name). It should be set to a length of at least 400 bytes.</td>
</tr>
</tbody>
</table>
Notes

- The return value is normally zero, even if the formula has errors. A non-zero return value means serious code-level error.
- This function is called by `EssOtlVerifyOutlineEx`, but can also be called directly from a client program.

Return Value

This function returns zero if successful, otherwise it returns an error code of either OTLAPI_ERR_HOUTLINE or OTLAPI_NULL_ARG. The return value can be zero even in the case of minor errors in the formula. A non-zero return value indicates a serious code-level error.

Any formula error is returned in the `pErrorNumber` and `pErrorLine` variables.

A non-zero return value indicates a serious code-level error in which case the error checking has been interrupted and `pErrorNumber` and `pErrorLine` both are set to zero.

See Also

- `EssOtlVerifyOutlineEx`
- `EssVerifyFilter`
- `EssVerifyRulesFile`

**EssOtlVerifyOutline**

Verifies that an outline is correct. The function returns both global outline errors and errors for each incorrect member.

Syntax

```c
ESS_FUNC_M EssOtlVerifyOutline (hOutline, pulErrors, pulCount, pMbrErrors);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline context handle.</td>
</tr>
<tr>
<td>pulErrors</td>
<td>ESS_PULONG_T</td>
<td>Pointer to bitmask destination for global outline errors. Currently, this field has only one value: ESS_OUTERROR_CURTOOMANYDIMS</td>
</tr>
<tr>
<td>pulCount</td>
<td>ESS_PULONG_T</td>
<td>Count of members with errors. This defines the number of elements of the <code>pMbrErrors</code> array.</td>
</tr>
<tr>
<td>pMbrErrors</td>
<td>“ESS_OUTERROR_T” on page 704</td>
<td>Pointer to an array with <code>*pulCount</code> members. Each element of the array contains the errors for a single member.</td>
</tr>
</tbody>
</table>

Notes

- This function checks for:
  - Duplicate user attributes in shared members
  - Duplicate level or generation names or aliases
  - Restrictions on adding and associating attributes
Saving the outline to the server succeeds only when the outline is free of errors (*pulErrors == 0 and *pulCount == 0).

Use **EssFree()** to free the **pMbrErrors** array.

**Return Value**

Returns 0 if successful; otherwise one of the following:

- **ESS_OUTERROR_SHAREUDA**
- **ESS_OUTERROR_DUPGENLEVNAME**

**Example**

```c
#include <essapi.h>
#include <essotl.h>

ESS_STS_T          sts = 0;
ESS_OBJDEF_T       Object;
ESS_HOUTLINE_T     hOutline;
ESS_ULONG_T        ulErrors;
ESS_ULONG_T        ulCount;
ESS_POUTERROR_T    pMbrErrors = ESS_NULL;
ESS_APPNAME_T      szAppName;
ESS_DBNAME_T       szDbName;
ESS_OBJNAME_T      szFileName;
memset(&Object, '\0', sizeof(Object));
Object.hCtx = hCtx;
Object.ObjType = ESS_OBJTYPE_OUTLINE;
strcpy(szAppName, "Sample");
strcpy(szDbName, "Basic");
strcpy(szFileName, "Basic");
Object.AppName = szAppName;
Object.DbName = szDbName;
Object.FileName = szFileName;

sts = EssOtlOpenOutline(hCtx, &Object, ESS_TRUE,
                       ESS_TRUE, &hOutline);

if (!sts)
{
    sts = EssOtlVerifyOutline(hOutline, &ulErrors,
                              &ulCount, &pMbrErrors);
}

if (pMbrErrors)
{
    EssFree(hInst, pMbrErrors);
}
```

**See Also**

- **EssOtlNewOutline**
- **EssOtlOpenOutline**
- **EssOtlWriteOutline**
**EssOtlVerifyOutlineEx**

Verifies that the specified outline is correct and builds an array of the errors found in that outline. The function returns both global outline errors and errors for each incorrect member.

**Syntax**

```c
ESS_FUNC_M EssOtlVerifyOutlineEx (hOutline, pulErrors, pulCount, pMbrErrors);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline context handle.</td>
</tr>
<tr>
<td>pulErrors</td>
<td>ESS_PULONG_T</td>
<td>Pointer to bitmask destination for global outline errors. If the outline had formula errors the only field with a value is: ESS_OUTERROREX_OUTLINEHASFORMULAERROR</td>
</tr>
<tr>
<td>pulCount</td>
<td>ESS_PULONG_T</td>
<td>Count of members with errors. This defines the number of members of the pMbrErrors array. The errors will be bitmasks if the outline had errors. If the outline had only formula errors the pMbrError fields comprise error numbers (ulErrors) and line numbers (ulErrors2). In that case pulErrors is set to ESS_OUTERROREX_OUTLINEHASFORMULAERROR.</td>
</tr>
<tr>
<td>pMbrErrors</td>
<td>“ESS_OUTERROR_T” on page 704</td>
<td>Pointer to an array with *pulCount members. Each element of the array contains the errors for a single member.</td>
</tr>
</tbody>
</table>

**Notes**

- This function calls **EssOtlVerifyOutline**. If that call is successful, this function then calls **EssOtlVerifyFormula** for each member that has a formula and includes any formula errors in the output error array. If the call to EssOtlVerifyOutline() in not successful, this function operates exactly like EssOtlVerifyOutline().

- This function checks for:
  - Duplicate user attributes in shared members.
  - Duplicate level or generation names or aliases.
  - Restrictions on adding and associating attributes.

- Use **EssFree()** to free the pMbrErrors array.

**Return Value**

Returns 0 if successful; otherwise one of the following:

- OTLAPI_ERR_OPENMODE
- OTLAPI_BAD_HOUTLINE OTLAPI_NULL_ARG

**Example**

```c
ESS_STS_T TestVerifyOtlEx(ADT_CMDCTX_T *cmdctxp)
{
    ESS_STS_T         sts = ESS_STS_NOERR;
    ESS_STS_T         sts2 = ESS_STS_NOERR;
    ESS_SHORT_T       hOutline;
    ESS_ULONG_T       ulErrors;
    ESS_ULONG_T       ulCount;
```
ESS_POUTERROR_T pMbrErrors;
ESS_ULONG_T ind;
ESS_PMBRINFO_T *ppMbrInfo;

if (cmdctxp->cmdbuf.argvn < 2)
{
    hOutlineChoice = ishOutlineMenu(cmdctxp);
} else
{
    hOutlineChoice = atoi(*(cmdctxp->cmdbuf.argv + 1));
}

sts = EssOtlVerifyOutlineEx(cmdctxp->hOutline[hOutlineChoice], &ulErrors,
&ulCount, &pMbrErrors);

if (sts == ESS_STS_NOERR)
{
    fprintf(cmdctxp->output, "\n------Global Errors------\n");

    if (ulErrors & ESS_OUTERROR_CURTOOMANYDIMS)
    {
        fprintf(cmdctxp->output, "Too many dimensions in currency outline\n");
    } else if(ulErrors & ESS_OUTERROR2_ATTRCALCABSENT)
    {
        fprintf(cmdctxp->output, "Attribute calculations dimension is absent\n");
    } else if(ulErrors & ESS_OUTERROREX_OUTLINEHASFORMULAERROR)
    {
        fprintf(cmdctxp->output, "Outline has formula error\n");
    } else if (ulErrors == 0)
    {
        fprintf(cmdctxp->output, "No errors\n");
    } else
    {
        fprintf(cmdctxp->output, "Unknown error\n");
    }

    fprintf(cmdctxp->output, "\n------Member Errors------\n");

    if (ulErrors != ESS_OUTERROREX_OUTLINEHASFORMULAERROR)
    {
        for (ind = 0; ind < ulCount; ind++)
        {
            sts2 = EssOtlGetMemberInfo(cmdctxp->hOutline[hOutlineChoice],
pMbrErrors[ind].hMember, &ppMbrInfo);

            if (sts2 == ESS_STS_NOERR)
            {
                fprintf(cmdctxp->output, "Member: %s\n",
                ppMbrInfo->szMember);
                EssFree(cmdctxp->hInst, ppMbrInfo);
            }
        }
    }
} else {
    fprintf(cmdctxp->output, "Member: Unknown member \n");
}

if (pMbrErrors[ind].ulErrors & ESS_OUTERROR_ILLEGALNAME) {
    fprintf(cmdctxp->output, "ESS_OUTERROREX_OUTLINEHASFORMULAERROR\n");
}

if (pMbrErrors[ind].ulErrors & ESS_OUTERROR_DUPLICATENAME) {
    fprintf(cmdctxp->output, "ESS_OUTERROR_DUPLICATENAME\n");
}

if (pMbrErrors[ind].ulErrors & ESS_OUTERROR_ILLEGALCURRENCY) {
    fprintf(cmdctxp->output, "ESS_OUTERROR_ILLEGALCURRENCY\n");
}

if (pMbrErrors[ind].ulErrors & ESS_OUTERROR_ILLEGALDEFALIAS) {
    fprintf(cmdctxp->output, "ESS_OUTERROR_ILLEGALDEFALIAS\n");
}

if (pMbrErrors[ind].ulErrors & ESS_OUTERROR_ILLEGALCOMBOALIAS) {
    fprintf(cmdctxp->output, "ESS_OUTERROR_ILLEGALCOMBOALIAS\n");
}

if (pMbrErrors[ind].ulErrors & ESS_OUTERROR_ILLEGALALIASSTRING) {
    fprintf(cmdctxp->output, "ESS_OUTERROR_ILLEGALALIASSTRING\n");
}

if (pMbrErrors[ind].ulErrors & ESS_OUTERROR_ILLEGALTAG) {
    fprintf(cmdctxp->output, "ESS_OUTERROR_ILLEGALTAG\n");
}

if (pMbrErrors[ind].ulErrors & ESS_OUTERROR_NOTIMEDIM) {
ESS_OUTERROR_NOTIMEDIM
"
);

if (pMbrErrors[ind].ulErrors &

ESS_OUTERROR_DUPLICATEALIAS)
{
    fprintf(cmdctxp->output,

ESS_OUTERROR_DUPLICATEALIAS"
";
}

if (pMbrErrors[ind].ulErrors & ESS_OUTERROR_MEMBERCALC)
{
    fprintf(cmdctxp->output,

ESS_OUTERROR_MEMBERCALC"
";
}

if (pMbrErrors[ind].ulErrors &

ESS_OUTERROR_SHARENOTLEVEL0)
{
    fprintf(cmdctxp->output,

ESS_OUTERROR_SHARENOTLEVEL0"
";
}

if (pMbrErrors[ind].ulErrors &

ESS_OUTERROR_NOSHAREPROTO)
{
    fprintf(cmdctxp->output,

ESS_OUTERROR_NOSHAREPROTO"
";
}

if (pMbrErrors[ind].ulErrors & ESS_OUTERROR_TIMESPARSE)
{
    fprintf(cmdctxp->output,

ESS_OUTERROR_TIMESPARSE"
";
}

if (pMbrErrors[ind].ulErrors & ESS_OUTERROR_LEAFLABEL)
{
    fprintf(cmdctxp->output,

ESS_OUTERROR_LEAFLABEL"
";
}

if (pMbrErrors[ind].ulErrors &

ESS_OUTERROR_ALIASSHARED)
{
    fprintf(cmdctxp->output,

ESS_OUTERROR_ALIASSHARED"
";
}

if (pMbrErrors[ind].ulErrors & ESS_OUTERROR_BADTIMEBAL)
{
    fprintf(cmdctxp->output,

ESS_OUTERROR_BADTIMEBAL"
";
}

if (pMbrErrors[ind].ulErrors & ESS_OUTERROR_BADSKIP)
if (pMbrErrors[ind].ulErrors & ESS_OUTERROR_BADSHARE) {
    fprintf(cmdctxp->output,"   ESS_OUTERROR_BADSHARE
\n");
}

if (pMbrErrors[ind].ulErrors & ESS_OUTERROR_BADSTORAGE) {
    fprintf(cmdctxp->output," ESS_OUTERROR_BADSTORAGE\n");
}

if (pMbrErrors[ind].ulErrors & ESS_OUTERROR_BADCATEGORY) {
    fprintf(cmdctxp->output," ESS_OUTERROR_BADCATEGORY\n");
}

if (pMbrErrors[ind].ulErrors & ESS_OUTERROR_BADSTORAGECATEGORY) {
    fprintf(cmdctxp->output," ESS_OUTERROR_BADSTORAGECATEGORY\n");
}

if (pMbrErrors[ind].ulErrors & ESS_OUTERROR_SHAREDMEMBERFORMULA) {
    fprintf(cmdctxp->output," ESS_OUTERROR_SHAREDMEMBERFORMULA\n");
}

if (pMbrErrors[ind].ulErrors & ESS_OUTERROR_SHAREUDA) {
    fprintf(cmdctxp->output," ESS_OUTERROR_SHAREUDA\n");
}

if (pMbrErrors[ind].ulErrors & ESS_OUTERROR_VIRTLEV0NOFORMULA) {
    fprintf(cmdctxp->output," ESS_OUTERROR_VIRTLEV0NOFORMULA\n");
}
if (pMbrErrors[ind].ulErrors &
ESS_OUTERROR_VIRTBADPARENT)
{
    fprintf(cmdctxp->output,"ESS_OUTERROR_VIRTBADPARENT\n");
}

if (pMbrErrors[ind].ulErrors &
ESS_OUTERROR_VIRTBADCHILD)
{
    fprintf(cmdctxp->output,"ESS_OUTERROR_VIRTBADCHILD\n");
}

if (pMbrErrors[ind].ulErrors &
ESS_OUTERROR_VIRTWHOLEDIMVIRTUAL)
{
    fprintf(cmdctxp->output,"ESS_OUTERROR_VIRTWHOLEDIMVIRTUAL\n");
}

if (pMbrErrors[ind].ulErrors2 &
ESS_OUTERROR2_NOTLEVEL0)
{
    fprintf(cmdctxp->output,"ESS_OUTERROR2_NOTLEVEL0\n");
}

if (pMbrErrors[ind].ulErrors2 &
ESS_OUTERROR2_LEVELMISMATCH)
{
    fprintf(cmdctxp->output,"ESS_OUTERROR2_LEVELMISMATCH\n");
}

if (pMbrErrors[ind].ulErrors2 &
ESS_OUTERROR2_ILLEGALORDER)
{
    fprintf(cmdctxp->output,"ESS_OUTERROR2_ILLEGALORDER\n");
}

if (pMbrErrors[ind].ulErrors2 &
ESS_OUTERROR2_ILLEGALDATATYPE)
{
    fprintf(cmdctxp->output,"ESS_OUTERROR2_ILLEGALDATATYPE\n");
}

if (pMbrErrors[ind].ulErrors2 &
ESS_OUTERROR2_DATATYPEMISMATCH)
{
    fprintf(cmdctxp->output,"ESS_OUTERROR2_DATATYPEMISMATCH\n");
}

if (pMbrErrors[ind].ulErrors2 &
ESS_OUTERROR2_ILLEGALATTRIBUTEPARENT)
{
    fprintf(cmdctxp->output,"ESS_OUTERROR2_ILLEGALATTRIBUTEPARENT\n");
}
if (pMbrErrors[ind].ulErrors2 & ESS_OUTERROR2_ATTRDIMNOTASSOCIATED) {
    fprintf(cmdctxp->output," ESS_OUTERROR2_ATTRDIMNOTASSOCIATED\n");
} if (pMbrErrors[ind].ulErrors2 & ESS_OUTERROR2_ILLEGALUDA) {
    fprintf(cmdctxp->output," ESS_OUTERROR2_ILLEGALUDA\n");
} if (pMbrErrors[ind].ulErrors2 & ESS_OUTERROR2_CHILDCCOUNT) {
    fprintf(cmdctxp->output," ESS_OUTERROR2_CHILDCCOUNT\n");
} if (pMbrErrors[ind].ulErrors2 & ESS_OUTERROR2_ILLEGALATTRCALC) {
    fprintf(cmdctxp->output," ESS_OUTERROR2_ILLEGALATTRCALC\n");
} if (pMbrErrors[ind].ulErrors2 & ESS_OUTERROR2_DUPLICATEATTRCALC) {
    fprintf(cmdctxp->output," ESS_OUTERROR2_DUPLICATEATTRCALC\n");
} if (pMbrErrors[ind].ulErrors2 & ESS_OUTERROR2_ILLEGALATTRSET) {
    fprintf(cmdctxp->output," ESS_OUTERROR2_ILLEGALATTRSET\n");
} if (pMbrErrors[ind].ulErrors2 & ESS_OUTERROR2_ILLEGALATTRCALCSET) {
    fprintf(cmdctxp->output," ESS_OUTERROR2_ILLEGALATTRCALCSET\n");
} if (pMbrErrors[ind].ulErrors2 & ESS_OUTERROR2_NOTATTRIBUTE) {
    fprintf(cmdctxp->output," ESS_OUTERROR2_NOTATTRIBUTE\n");
} if (pMbrErrors[ind].ulErrors2 & ESS_OUTERROR2_ATTRCALCABSENT) {
    fprintf(cmdctxp->output," ESS_OUTERROR2_ATTRCALCABSENT\n");
} if (pMbrErrors[ind].ulErrors2 & ESS_OUTERROR2_ILLEGALATTRVALUE)
if(ulErrors != ESS_OUTERROREX_OUTLINEHASFORMULAERROR)
{
    for (ind = 0; ind < ulCount; ind++)
    {
        sts2 = EssOtlGetMemberInfo(cmdctxp->hOutline[hOutlineChoice],
                                   pMbrErrors[ind].hMember,
                                   &ppMbrInfo);

        if (sts2 == ESS_STS_NOERR)
        {
            fprintf(cmdctxp->output, "Member: %s\n", ppMbrInfo->szMember);
            EssFree(cmdctxp->hInst, ppMbrInfo);
        }
        else
        {
            fprintf(cmdctxp->output, "Member: Unknown member\n");
        }

        fprintf(cmdctxp->output, "Error %d at line %d\n",
                pMbrErrors[ind].ulErrors, pMbrErrors[ind].ulErrors2);
    }

    if (ulCount == 0)
    {
        fprintf(cmdctxp->output, "No errors\n");
    }
    EssFree(cmdctxp->hInst, pMbrErrors);
}

fprintf(cmdctxp->output, "\nsts: %ld\n
", sts);
return(sts);
}

See Also

- EssOtlNewOutline
- EssOtlOpenOutline
- EssOtlWriteOutline
- EssOtlVerifyOutline
- EssOtlVerifyFormula

EssOtlWriteOutline

 Writes the existing outline information to disk.
Syntax

```c
ESS_FUNC_M EssOtWriteOutline (hOutline, pObject);
```

Parameter  Data Type  Description

| hOutline | ESS_HOUTLINE_T | Outline context handle.
| pObject  | ESS_POBJDEF_T  | Outline object to write.

Notes

- If you are saving the outline as a server object, the outline is initially saved as a .OTN file. You should then call `EssOtRestructure()` to create the actual .OTL file.
- If you are saving the outline as a server object, the object name must be the same as the database name.
- The database must already exist if you are saving a server outline object, or a client outline object to a local database.
- This call fails if the outline is not currently locked by the specified user (`hCtx` parameter in the ESS_OBJDEF_T structure).

Return Value

Returns 0 if successful; otherwise one of the following:
- OTLAPI_BAD_OBJTYPE
- OTLAPI_ERR_NOTVERIFIED

Access

This function requires you to have the appropriate level of access to the specified application and/or database to contain the outline object. To write the outline object, you must have Application Designer or Database Designer privilege (ESS_PRIV_APPDESIGN or ESS_PRIV_DBDESIGN) for the specified application or database containing the outline.

Example

```c
#include <essapi.h>
#include <essotl.h>

ESS_STS_T          sts = 0;
ESS_HOUTLINE_T     hOutline;
ESS_OBJDEF_T       Object;
ESS_APPNAME_T      szAppName;
ESS_DBNAME_T       szDbName;
ESS_OBJNAME_T      szFileName;
memset(&Object, '\0', sizeof(Object));
Object.hCtx = hCtx;
Object.ObjType = ESS_OBJTYPE_OUTLINE;
strcpy(szAppName, "Sample");
strcpy(szDbName, "Basic");
strcpy(szFileName, "Basic");
Object.AppName = szAppName;
Object.DbName = szDbName;
```
Object.FileName = szFileName;

sts = EssOtlOpenOutline(hCtx, &Object, ESS_TRUE,
                    ESS_TRUE, &hOutline);

/* body of code */
if (!sts)
    { sts = EssOtlWriteOutline(hOutline, &Object); }

/* restructure db using EssOtlRestructure() */

See Also
- EssOtlWriteOutlineEx
- EssOtlOpenOutline
- EssOtlNewOutline
- EssOtlVerifyOutline
- EssOtlRestructure
- EssOtlCloseOutline

EssOtlWriteOutlineEx

Writes the existing outline information to disk, specifying whether to save in UTF-8 encoding or in non-Unicode encoding.

Syntax

```c
ESS_FUNC_M EssOtlWriteOutlineEx(hOutline, pObject, iOtlType);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>ESS_HOUTLINE_T</td>
<td>Outline context handle.</td>
</tr>
<tr>
<td>pObject</td>
<td>ESS_POBJDEF_T</td>
<td>Outline object to write.</td>
</tr>
<tr>
<td>iOtlType</td>
<td></td>
<td>Whether the outline is saved in Unicode mode or non-Unicode mode. The valid values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_OUTLINE_UTF8 0x0002—Encoded in UTF-8.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_OUTLINE_NONUNICODE 0x0003—Not Unicode-encoded.</td>
</tr>
</tbody>
</table>

Notes

- If you are saving the outline as a server object, the outline is initially saved as a .OTN file. You should then call EssOtlRestructure() to create the actual .OTL file.
- If you are saving the outline as a server object, the object name must be the same as the database name.
- The database must already exist if you are saving a server outline object, or a client outline object to a local database.
- This call fails if the outline is not currently locked by the specified user (hCtx parameter in the ESS_OBJDEF_T structure).

**Return Value**

Returns 0 if successful; otherwise one of the following:

- OTLAPI_BAD_OBJTYPE
- OTLAPI_ERR_NOTVERIFIED

**Access**

This function requires you to have the appropriate level of access to the specified application and/or database to contain the outline object. To write the outline object, you must have Application Designer or Database Designer privilege (ESS_PRIV_APPDESIGN or ESS_PRIV_DBDESIGN) for the specified application or database containing the outline.

**See Also**

- EssOtlOpenOutlineEx
- EssOtlCloseOutline
- EssOtlGetMemberCommentEx
- EssOtlNewOutline
- EssOtlRestructure
- EssOtlSetMemberCommentEx
- EssOtlVerifyOutline
Example of Traversing an Outline

This example demonstrates the use of the outline tree. TraverseTree is a recursive algorithm that traverses the outline tree to provide access to all outline members. It selects each member in turn, allowing processing on each, until it reaches the last member. A comment in the code notes the opportunity for added processing.

This algorithm incorporates several C Outline API commands.

- **EssOtlGetFirstMember()** returns a member handle to the first member (the first dimension defined) in the outline.
- **EssOtlGetMemberInfo()** gets information for the specified member.
- **EssOtlGetChild()** returns the child of a member.
- **EssOtlGetNextSibling()** returns the next sibling of a member.

Before executing this code, initialize the API and open the outline. Following this code, close the outline and terminate the API.

```c
TraverseTree (ESS_HOUTLINE_T)
{
    ESS_HMEMBER_T hMember;
    ESS_STS_T sts = 0;

    sts = EssOtlGetFirstMember(hOutline, &hMember);
    if (!sts && hMember)
        sts = TraverseTreeRecurse(hOutline, hMember);
}

TraverseTreeRecurse(ESS_HOUTLINE_T hOutline, ESS_HMEMBER_T hMember)
{
    ESS_MEMBERINFO_T MbrInfo;
    ESS_HMEMBER_T, hChild;
    ESS_STS_T sts = 0;

    while (!sts && hMember)
    {
```
sts = EssOtlGetMemberInfo (hOutline, hMember, &MbrInfo);

/* ADD THE PROCESSING FOR EACH MEMBER HERE. */

if (!sts)
{
    sts = EssOtlGetChild(hOutline, hMember, &hChild);
    if (!sts && hChild)
    {
        sts = TraverseTreeRecurse(hOutline, hChild);
    }
}
sts = EssOtlGetNextSibling(hOutline, hMember, &hMember);
return (sts);

---

Extended Member Query Code Example

```c
#include <windows.h>
#include <essapi.h>
#include <essotl.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#define AD_CHK_PRINTF_1(ARG1, ARG2)
do
    { printf(ARG1, (ARG2) ? (ARG2) : "NullValue"); }
while (0)

void PrintResult(ESS_HCTX_T      hCtx,
             ESS_HINST_T     hInst,
             ESS_HOUTLINE_T  hOutline,
             ESS_HMEMBER_T   hMbr)
{
    ESS_PMBRINFO_T pMbrInfo = NULL;
    ESS_STS_T      sts;
    int            size;
    ESS_STR_T      pszFormula = NULL;
    ESS_STR_T      pszLastFormula = NULL;
    ESS_STR_T      pszCommentEx = NULL;
    ESS_STR_T      pszAlias = NULL;
    ESS_STR_T      pszAliasCombo = NULL;
    ESS_PMBRNAME_T pUDAList = NULL;
    ESS_USHORT_T   iCount = 0;
    ESS_STR_T      pszPrev = NULL;
    ESS_USHORT_T   iIndex;

    ESS_ULONG_T* pMemNum;
    ESS_ULONG_T* pDimNum;
    ESS_STR_T      pDimName = NULL;
    ESS_STR_T      pAliasName = NULL;
```
ESS_STR_T   pNextName = NULL;
ESS_STR_T   pPrevName = NULL;
ESS_STR_T   pParentName = NULL;
ESS_STR_T   pChildName = NULL;
NESS_BOOL_T*  pCurrConv = NULL;
NESS_ULONG_T* pStatus = NULL;

sts = EssOtlGetMemberInfo(hOutline, hMbr, &pMbrInfo);
if (sts != 0) goto Error;

size = sizeof(ESS_MBRINFO_T);

printf("MbrInfo\n");
AD_CHK_PRINTF_1("    szMember --------------->(%s)\n",  pMbrInfo->szMember);
printf("    usLevel  --------------->(%hd)\n", pMbrInfo->usLevel);
printf("    usGen  ----------------->(%hd)\n", pMbrInfo->usGen);
printf("    usConsolidation  ------->(%hd)\n", pMbrInfo->usConsolidation);
printf("    fTwoPass  -------------->(%hd)\n", pMbrInfo->fTwoPass);
printf("    fExpense  -------------->(%hd)\n", pMbrInfo->fExpense);
printf("    usConversion ----------->(%hd)\n", pMbrInfo->usConversion);
AD_CHK_PRINTF_1("    szCurMember ------------>(%s)\n",  pMbrInfo->szCurMember);
printf("    usTimeBalance ---------->(%hd)\n", pMbrInfo->usTimeBalance);
printf("    usSkip ----------------->(%hd)\n", pMbrInfo->usSkip);
printf("    usShare ---------------->(%hd)\n", pMbrInfo->usShare);
printf("    usStorage -------------->(%hd)\n", pMbrInfo->usStorage);
printf("    usCategory ------------->(%hd)\n", pMbrInfo->usCategory);
printf("    usStorageCategory ------>(%hd)\n", pMbrInfo->usStorageCategory);
AD_CHK_PRINTF_1("    szComment ---------------->(%s)\n",  pMbrInfo->szComment);
printf("    ulChildCount ----------->(%ld)\n", pMbrInfo->ulChildCount);

sts = EssOtlGetMemberFormula(hOutline, hMbr, &pszFormula);
if (sts) printf("sts=%d ", sts);
AD_CHK_PRINTF_1("szFormula ------------------>(%s)\n", pszFormula);

sts = EssOtlGetMemberLastFormula(hOutline, hMbr, &pszLastFormula);
if (sts) printf("sts=%d ", sts);
AD_CHK_PRINTF_1("szLastFormula -------------->(%s)\n", pszLastFormula);

sts = EssOtlGetMemberCommentEx(hOutline, hMbr, &pszCommentEx);
if (sts) printf("sts=%d ", sts);
AD_CHK_PRINTF_1("szCommentEx ---------------->(%s)\n", pszCommentEx);

sts = EssOtlGetMemberAlias(hOutline, hMbr, ESS_NULL, &pszAlias);
if (sts) printf("sts=%d ", sts);
AD_CHK_PRINTF_1("szAlias (Default)----------->(%s)\n", pszAlias);

sts = EssOtlGetNextAliasCombination(hOutline, hMbr, ESS_NULL, "\0", &pszAliasCombo);
if (sts) printf("sts=%d ", sts);
printf("szAliasCombo :\n");
pszPrev = pszAliasCombo;
while (sts && pszAliasCombo)
{
    AD_CHK_PRINTF_1("\t(%s)\n", pszAliasCombo);
    sts = EssOtlGetNextAliasCombination(hOutline, hMbr, ESS_NULL, pszPrev, &pszAliasCombo);
    EssFree(hInst, pszPrev);
}
pszPrev = pszAliasCombo;
}

sts = EssOtlGetUserAttributes(hOutline, hMbr, &iCount, &pUDAList);
if (sts) printf("sts=%d ", sts);

printf("User Defined Attributes ::\n");
for(iIndex = 0; iIndex < iCount; iIndex++)
    AD_CHK_PRINTF_1("\t(%s)\n", pUDAList[iIndex]);

sts = EssOtlGetMemberField(hOutline, hMbr, ESS_OTLQRYMBR_NUMBER, (ESS_PPVOID_T)&pMemNum);
if (sts)
    {printf("sts=%d ", sts);
    }
else
    {
        printf("Member Number ------------------>(%ld)\n", *pMemNum);
        EssFree(hInst, pMemNum);
    }

sts = EssOtlGetMemberField(hOutline, hMbr, ESS_OTLQRYMBR_DIMNUMBER, (ESS_PPVOID_T)&pDimNum);
if (sts)
    {printf("sts=%d ", sts);
    }
else
    {
        printf("Dimension Number ------------------>(%ld)\n", *pDimNum);
        EssFree(hInst, pDimNum);
    }

sts = EssOtlGetMemberField(hOutline, hMbr, ESS_OTLQRYMBR_DIMNAME, (ESS_PPVOID_T)&pDimName);
if (sts)
    {printf("sts=%d ", sts);
    }
else
    {
        AD_CHK_PRINTF_1("Dimension Name ------------------>(%s)\n", pDimName);
        EssFree(hInst, pDimName);
    }

sts = EssOtlGetMemberField(hOutline, hMbr, ESS_OTLQRYMBR_ALIASNAME, (ESS_PPVOID_T)&pAliasName);
if (sts)
    {printf("sts=%d ", sts);
    }
else
    {
        AD_CHK_PRINTF_1("Alias Name ------------------>(%s)\n", pAliasName);
        EssFree(hInst, pAliasName);
    }
sts = EssOtlGetMemberField(hOutline, hMbr, ESS_OTLQRYMBR_NEXTNAME, (ESS_PPVOID_T) &pNextName);
if (sts)
{
  printf("sts=%d ", sts);
}
else
{
  AD_CHK_PRINTF_1("Next Mbr Name ------------------>(%s)\n", pNextName);
  EssFree(hInst, pNextName);
}

sts = EssOtlGetMemberField(hOutline, hMbr, ESS_OTLQRYMBR_PREVNAME, (ESS_PPVOID_T) &pPrevName);
if (sts)
{
  printf("sts=%d ", sts);
}
else
{
  AD_CHK_PRINTF_1("Prev Mbr Name ------------------>(%s)\n", pPrevName);
  EssFree(hInst, pPrevName);
}

sts = EssOtlGetMemberField(hOutline, hMbr, ESS_OTLQRYMBR_PARENNTNAME, (ESS_PPVOID_T) &pParentName);
if (sts)
{
  printf("sts=%d ", sts);
}
else
{
  AD_CHK_PRINTF_1("Parent MbrName ------------------>(%s)\n", pParentName);
  EssFree(hInst, pParentName);
}

sts = EssOtlGetMemberField(hOutline, hMbr, ESS_OTLQRYMBR_CHILDNAME, (ESS_PPVOID_T) &pChildName);
if (sts)
{
  printf("sts=%d ", sts);
}
else
{
  AD_CHK_PRINTF_1("Child Mbr Name ------------------>(%s)\n", pChildName);
  EssFree(hInst, pChildName);
}

sts = EssOtlGetMemberField(hOutline, hMbr, ESS_OTLQRYMBR_CURRENCYCONVDB, (ESS_PPVOID_T) &pCurrConv);
if (sts)
{
  printf("sts=%d ", sts); printf("Curr Conv Type ------------------->\n");
}
else
{

Extended Member Query Code Example 985
AD_CHK_PRINTF_1("Curr Conv Type ------------------>(%ld)\n", *pCurrConv);
EssFree(hInst, pCurrConv);
}

sts = EssOtlGetMemberField(hOutline, hMbr, ESS_OTLQRYMBR_STATUS, (ESS_PPVOID_T)
&pStatus);
if (sts)
{  printf("sts=%d ", sts); printf("Status ------------------>
"); }
else
{
  printf("Status ------------------>(%hd)\n", *pStatus);
  EssFree(hInst, pStatus);
}

EssFree(hInst, pMbrInfo);
EssFree(hInst, pszFormula);
EssFree(hInst, pszLastFormula);
EssFree(hInst, pszCommentEx);
EssFree(hInst, pszAlias);
EssFree(hInst, pszAliasCombo);
return;

Error:
  printf("************************* Error *************************");
}

int TestCode_EssOtlQueryMembersEx(ESS_HCTX_T hCtx,
      ESS_HINST_T  hInst)
{
  ESS_STS_T sts = ESS_STS_NOERR;
  ESS_HOUTLINE_T hOutline;
  ESS_OBJDEF_T Object;
  ESS_HMEMBER_T hMember = 0;
  ESS_PHMEMBER_T phMemberArray = ESS_NULL;
  ESS_ULONG_T i;
  unsigned long MaxCount = -1;
  ESS_STR_T member_fields;
  ESS_STR_T member_selection;

  /* query string to get level numbers of all markets members */
  member_fields = "<SelectMbrInfo ( MemberName, MemberLevel,Consolidation,
             MemberFormula ) ");
  member_selection = "@ichild(Product), @ichild(Market)";
  memset(&Object, '\0', sizeof(Object));
  Object.hCtx = hCtx;
  Object.ObjType = ESS_OBJTYPE_OUTLINE;
  Object.AppName = "Basic";
  Object.DbName = "Demo";
  Object.FileName = "Demo";

  sts = EssOtlOpenOutlineQuery(hCtx, &Object, &hOutline);
  if (!sts) goto exit;

  if (!sts)
  {
    ESS_POTLQUERYERRORLIST_T pqryErrorList;

sts = EssOtlQueryMembersEx(hOutline, member_fields, member_selection, &MaxCount, &phMemberArray, &pqryErrorList);
    if (sts) goto exit;

    if (phMemberArray)
        for (i = 0; i< MaxCount; i++)
            PrintResult(hOutline, phMemberArray[i]);
    
    if(MaxCount && phMemberArray)
    {
        sts = EssOtlFreeMembers(hOutline, MaxCount, phMemberArray);
        if (sts)
            printf("EssOtlFreeMembers    sts = %d\n",sts);
    }

    sts = EssOtlCloseOutline(hOutline);

exit:
    return sts;
}

Return to EssOtlQueryMembersEx or EssOtlGetMemberField.
Part IV

C Grid API

In C Grid API:

- Using the C Grid API
- C Grid API Declarations
- C Grid API Function Reference
- C Grid API Examples
- C Grid API Error Codes
Using the C Grid API

General Information on the C Grid API

Use the Essbase Application Programming Interface (API) to create custom interfaces to Essbase Server.

The Grid API functions interact with the Essbase Server in a grid paradigm. Use the Grid API functions to extract data from an Essbase database in order to display the data in a grid-based reporting interface or a chart.

The Grid API functions contain all the functionality of Spreadsheet Add-in. These functions include querying, drill-down, keep-only, and pivots. The Grid API can also launch report specifications and display the resulting data in grid form. Grid API functions can be tested by performing the same actions in Microsoft Excel or other automated grid tools.

The Essbase Grid API offers significant advantages for developers currently building reporting applications using the Essbase API and report script commands. Some of the benefits are as follows:

- The Grid API is optimized for grid-based data retrieval. It is significantly faster than report script retrieval, providing improved query performance for your application.
- The Grid API makes it easy to add robust interactive update capabilities to your existing applications. Coupled with the intelligent calculator in Essbase, the Grid API has the potential to provide tremendous additional functionality with relatively little effort on your part.
With the Grid API programs do not need to parse the returned data from Essbase. The Grid API automatically places it in a two dimensional binary form that informs you of the type and value of each individual cell member.

Most Spreadsheet Add-in commands, such as Zoom In, Zoom Out, Pivot, and so on, are available through the Grid API. You can build custom user interfaces using your own or third-party grid controls.

The Grid API also provides access to member attributes, such as Shared (implicit or explicit), Parent, or Child. You can customize the look and feel of Essbase data using these attributes.

**Overview of C Grid API Architecture**

The Essbase Grid API functions use the Common Spreadsheet Layer (CSL) to communicate with the Essbase Server.

The Grid API functions provide functionality similar to the Essbase extended spreadsheet macros.

The Grid API functions perform their actions on a grid. The result of any action is also a grid. It is the responsibility of the caller to supply the Grid API functions with the two-dimensional data for each call, and to render any returned data. It is also the responsibility of the caller to handle notifications, such as an error message dialog box, in a manner that ensures that the API application’s connection does not time out.

**C Grid API Supported Platforms and Compilers**

For a list of platforms on which the Essbase API is supported, see the *Oracle Hyperion Enterprise Performance Management System Certification Matrix* ([http://www.oracle.com/technology/software/products/ias/files/fusion_certification.html](http://www.oracle.com/technology/software/products/ias/files/fusion_certification.html)). For a list of specific compiler releases which are supported by the Essbase API, see “Supported Compilers” on page 29.

Function names and parameter order are the same for all platforms. However, you must link different files for each platform. See “API Libraries” on page 42.

- If you are using an integrated C development environment, such as Microsoft Visual C++, you should check the compiler and linker options carefully to ensure that the Essbase API will work correctly. In particular, you must ensure that structure fields are one byte-aligned, and that the correct libraries are used (using the large memory model on Intel X86 platforms). In addition, don’t forget to include the appropriate Essbase API library in your link process. See “API Libraries” on page 42.

- You must compile all Essbase API functions using single-byte structure alignment. If you are using a Microsoft compiler, you can use a pragma:

  ```c
  #pragma pack(push, id, 1)
  #include "essgapi.h"
  #pragma pack(pop, id)
  ```
Files to Include in C Grid API Programs

In order to use the Essbase API functions in your program, you must include the file that contains Essbase API definitions.

To use the Grid API functions in your C program, you must include the Grid API header definitions file (ESSGAPI.H) in the appropriate source modules. If, in addition to Grid API functions, you are using regular API functions, you must also include the Main API definitions file (ESSAPI.H) in the appropriate source modules.

Always include these definition files after any C run-time library header files. If you are programming in the Windows environment, place ESSGAPI.H, and optionally ESSAPI.H, after the Windows include file WINDOWS.H.

C Grid API Initialization and Setup

When you use Grid API functions, you must call the initialization function, EssGInit. This function performs the following tasks:

- Passes information about your environment to the Grid API functions.
- Provides you with an instance handle that you use for future communication with the Grid API functions.

Notes:

- When you call EssGInit, you get the Grid instance handle. If you want to use regular API functions, you can use EssGGetAPIInstance to get the API instance handle and EssGGetAPIContext to get the login context handle. See “Using the C Main API Functions” on page 994.
- You cannot use Main API instance handles and login contexts in Grid API calls. See “Using the C Main API Functions” on page 994.

C Grid API Memory Management

You must free any memory you allocate and any memory allocated by Grid API functions for your use. There are Grid API functions that free memory where necessary.

C Grid API Versioning

When you use EssGInit to initialize the API, you need to pass in the version number of the API libraries that you used to compile the application. This allows older applications to use new versions of the Grid API DLL and CSL DLL without your having to redistribute the applications.

A Grid API function reports the current release of the Grid API. You do not need to do any initialization before you make this call.
Using the C Grid API Functions

Many of the operations require a call to begin the operation. Other calls need to be made to complete the operation and retrieve data. The following list shows the order in which you should make these operational calls:

1. Call `EssGBeginXxx` to begin the operation.
2. Call `EssGSendRows` to send the rows. You can call this multiple times to send more data.
3. Call `EssGPerformOperation` to tell the API that all information has been passed in.
4. Call `EssGGetResults` to return information on the number of rows and columns that will be returned.
5. Call `EssGetRows` until all data is retrieved.
6. Call `EssGEndOperation` to clean up any internal resources.
7. Optionally call `EssGCancelOperation` at any stage of this process to cancel the operation.

After `EssGEndOperation`, `EssGCancelOperation`, or another `EssGBeginXxxx` operation is called, all information from the previous operation is lost.

Using the C Main API Functions

The Grid API functions are specific to the grid paradigm, and do not replace any of the functionality of the main Essbase API functions. Because of this operational separation, cases can arise where it is necessary to call Main API functions from within a Grid API program.

To call the Main API you need two pieces of information:

- Essbase instance handle
- Valid login context

If you have not called `EssGConnect` and `EssGNewGrid`, but have called `EssGInit`, you can get the Essbase instance handle by calling `EssGGetAPIInstance`. This gives you access to the memory calls `EssAlloc`, `EssFree`, `EssRealloc`, and to the login calls `EssLogin` and `EssAutoLogin`.

After you have a valid grid handle and have connected, you can call `EssGGetAPIContext` to get a valid login context. You can then use this login context handle with any Essbase function that takes a login context handle. Be careful NOT to use the login context from the Grid API in any Essbase API functions that would change the login context. The functions that change the login context are `EssLogin`, `EssAutoLogin`, `EssSetActive`, and `EssClearActive`.

Handles and login contexts acquired through the main Essbase API cannot be used in the Grid API calls. If you want to use both the main Essbase API and the Grid API, you need to initialize and connect through the Grid API and use the handles and login contexts from the Grid API for the other Essbase functions.
C Grid API Coordinate Systems

Assume a zero-based column and row numbering scheme in the range structure that you pass into functions that expect a two-dimensional array of data. The input and output data ranges will be in the same relative coordinate system, while the data arrays are always zero-based.

For example, assume that your first data cell is in the third row and fourth column, and you have three rows of five columns each. If you pass in the structure ESSG_RANGE_T, it would contain $ulStartRow = 2$, $ulStartColumn = 3$, $ulNumRows = 3$, and $ulNumColumns = 5$.

The two-dimensional array of ESSG_DATA_T items would start at index [0][0] and end at index [2][4].
C Grid API Constants

The following constants are defined in the Essbase Grid API.
Returned on successful API call.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESSG_STS_NOERR</td>
<td>0</td>
</tr>
</tbody>
</table>

Defines the version of the EGAPI API. Changes each time the API is modified.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESSG_VERSION</td>
<td>0x00040000</td>
</tr>
</tbody>
</table>

Define the maximum number of rows and columns supported

<table>
<thead>
<tr>
<th>Constant</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>WINODWS</td>
<td></td>
</tr>
<tr>
<td>ESSG_MAXROWS</td>
<td>0xFFFFFFFF / sizeof(ESSG_PDATA_T)</td>
</tr>
<tr>
<td>ESSG_MAXCOLUMNS</td>
<td>0xFFFFFFFF / sizeof(ESSG_DATA_T)</td>
</tr>
<tr>
<td>OTHER</td>
<td></td>
</tr>
<tr>
<td>ESSG_MAXROWS</td>
<td>0xFFFF / sizeof(ESSG_PDATA_T)</td>
</tr>
<tr>
<td>ESSG_MAXCOLUMNS</td>
<td>0xFFFF / sizeof(ESSG_DATA_T)</td>
</tr>
</tbody>
</table>

Used by the pAttributes member of the “ESSG_DATA_T” on page 1006 structure.
<table>
<thead>
<tr>
<th>Constant</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESSG_CA_READONLY</td>
<td>0x00000001</td>
</tr>
<tr>
<td>ESSG_CA_READWRITE</td>
<td>0x00000002</td>
</tr>
<tr>
<td>ESSG_CA_LINKEDOBJ</td>
<td>0x00000004</td>
</tr>
<tr>
<td>ESSG_CA_LINKPARTITION</td>
<td>0x00000008</td>
</tr>
<tr>
<td>ESSG_CA_LINKCELLNOTE</td>
<td>0x00000010</td>
</tr>
<tr>
<td>ESSG_CA_LINKWINAPP</td>
<td>0x00000020</td>
</tr>
<tr>
<td>ESSG_CA_LINKURL</td>
<td>0x00000040</td>
</tr>
<tr>
<td>ESSG_CA_AISDT</td>
<td>0x00000080</td>
</tr>
<tr>
<td>ESSG_CA_GLDT</td>
<td>0x00000040</td>
</tr>
</tbody>
</table>

Used for the `pAttributes` for members in the “ESSG_DATA_T” on page 1006 structure.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESSG_MA_DIMTOP</td>
<td>0x00000001</td>
</tr>
<tr>
<td>ESSG_MA_ZOOMINABLE</td>
<td>0x00000002</td>
</tr>
<tr>
<td>ESSG_MA_NEVERSHARE</td>
<td>0x00000004</td>
</tr>
<tr>
<td>ESSG_MA_LABELONLY</td>
<td>0x00000008</td>
</tr>
<tr>
<td>ESSG_MASTOREDATA</td>
<td>0x00000010</td>
</tr>
<tr>
<td>ESSG_MA_EXPSHARE</td>
<td>0x00000020</td>
</tr>
<tr>
<td>ESSG_MA_IMPSHARE</td>
<td>0x00000040</td>
</tr>
<tr>
<td>ESSG_MA_DYNICALC</td>
<td>0x00000080</td>
</tr>
<tr>
<td>ESSG_MA_FORMULA</td>
<td>0x00000100</td>
</tr>
<tr>
<td>ESSG_MA_ATTRIBUTE</td>
<td>0x00000200</td>
</tr>
<tr>
<td>ESSG_MA_DIMNUMBITS</td>
<td>0xF80000000 (the last 5 bits contain the dimension number)</td>
</tr>
</tbody>
</table>

Used by the `usType` member in the “ESSG_DATA_T” on page 1006 structure.
<table>
<thead>
<tr>
<th>Constant</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESSG_DT_DOUBLE</td>
<td>3</td>
</tr>
<tr>
<td>ESSG_DT_BLANK</td>
<td>4</td>
</tr>
<tr>
<td>ESSG_DT_RESERVED</td>
<td>5</td>
</tr>
<tr>
<td>ESSG_DT_ERROR</td>
<td>6</td>
</tr>
<tr>
<td>ESSG_DT_MISSING</td>
<td>7</td>
</tr>
<tr>
<td>ESSG_DT_ZERO</td>
<td>8</td>
</tr>
<tr>
<td>ESSG_DT_NOACCESS</td>
<td>9</td>
</tr>
<tr>
<td>ESSG_DT_MEMBER</td>
<td>10</td>
</tr>
<tr>
<td>ESSG_DT_FORMULA</td>
<td>11</td>
</tr>
<tr>
<td>ESSG_DT_ZEROwFORMULA</td>
<td>12</td>
</tr>
<tr>
<td>ESSG_DT_DOUBLEwFORMULA</td>
<td>13</td>
</tr>
<tr>
<td>ESSG_DT_BLANKwFORMULA</td>
<td>14</td>
</tr>
<tr>
<td>ESSG_DT_STRINGwFORMULA</td>
<td>15</td>
</tr>
<tr>
<td>ESSG_DT_MISSINGwFORMULA</td>
<td>16</td>
</tr>
<tr>
<td>ESSG_DT_NOACCESSwFORMULA</td>
<td>17</td>
</tr>
<tr>
<td>ESSG_DT_STRINGEX</td>
<td>18</td>
</tr>
<tr>
<td>ESSG_DT_MEMBEREX</td>
<td>19</td>
</tr>
<tr>
<td>ESSG_DT_STRINGEXwFORMULA</td>
<td>20</td>
</tr>
<tr>
<td>ESSG_DT_FORMULAEX</td>
<td>21</td>
</tr>
<tr>
<td>ESSG_DT_MEMBERwKEY</td>
<td>23</td>
</tr>
<tr>
<td>ESSG_DT_SMARTLIST</td>
<td>24</td>
</tr>
<tr>
<td>ESSG_DT_MNGLESS</td>
<td>25</td>
</tr>
<tr>
<td>ESSG_DT_DATE</td>
<td>26</td>
</tr>
</tbody>
</table>

**Note:** When the `usType` field of the ESSG_DATA_T structure is set to ESSG_DT_MEMBERwKEY, the `pszStr` field of Value(ESSG_DATA_VALUE) field is interpreted as follows: `<length of member name><the member-name><length of key><the key>` where the length elements are 2 bytes in size. Note that `<the member-name>` is null-terminated.

Used by the `ulOptions` parameter of `EssGBeginRetrieve`
<table>
<thead>
<tr>
<th>Constant</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESSG_RET_RETRIEVE</td>
<td>0</td>
</tr>
<tr>
<td>ESSG_RET_RETRIEVELOCK</td>
<td>1</td>
</tr>
<tr>
<td>ESSG_RET_LOCKONLY</td>
<td>2</td>
</tr>
</tbody>
</table>

Used by the ulOptions parameter of `EssGBeginUpdate`.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESSG_RET_REQUIRELOCK</td>
<td>0</td>
</tr>
<tr>
<td>ESSG_RET_LOCKIFNEEDED</td>
<td>1</td>
</tr>
</tbody>
</table>

This bitmask constant is used by the ulOptions parameter of `EssGBeginConditionalRetrieve`, `EssGBeginConditionalZoomIn`, `EssGBeginReport`, and `EssGBeginReportFile`.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESSG_NOATTRIBUTES</td>
<td>0x00001000</td>
</tr>
</tbody>
</table>

These bitmask constants are used by the ulOptions parameter of `EssGBeginZoomIn` and `EssGBeginConditionalZoomIn`.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESSG_ZOOM_DOWN</td>
<td>0x00000080</td>
</tr>
<tr>
<td>ESSG_ZOOM_ACROSS</td>
<td>0x00000100</td>
</tr>
</tbody>
</table>

Describe the connect options.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESSG_CONNECT_DEFAULT</td>
<td>0</td>
</tr>
<tr>
<td>ESSG_CONNECT_NODIALOG</td>
<td>1</td>
</tr>
</tbody>
</table>

Describe the various zoom levels.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESSG_OPTIONS</td>
<td>0</td>
</tr>
<tr>
<td>ESSG_NEXTLEVEL</td>
<td>1</td>
</tr>
<tr>
<td>ESSG_ALLLEVELS</td>
<td>2</td>
</tr>
<tr>
<td>ESSG_BOTTOMLEVEL</td>
<td>3</td>
</tr>
<tr>
<td>Constant</td>
<td>Definition</td>
</tr>
<tr>
<td>------------------------------</td>
<td>------------</td>
</tr>
<tr>
<td>ESSG_SIBLEVEL</td>
<td>4</td>
</tr>
<tr>
<td>ESSGSAMELEVEL</td>
<td>5</td>
</tr>
<tr>
<td>ESSG_SAMEGENERATION</td>
<td>6</td>
</tr>
<tr>
<td>ESSG_CALCLEVEL</td>
<td>7</td>
</tr>
<tr>
<td>ESSG_PARENTLEVEL</td>
<td>8</td>
</tr>
<tr>
<td>ESSG_TOPLEVEL</td>
<td>9</td>
</tr>
</tbody>
</table>

Used for setting and retrieving grid options

<table>
<thead>
<tr>
<th>Constant</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESSG_OP_DRILLLEVEL</td>
<td>1</td>
</tr>
<tr>
<td>ESSG_OP_INCSEL</td>
<td>2</td>
</tr>
<tr>
<td>ESSG_OP_SELONLY</td>
<td>3</td>
</tr>
<tr>
<td>ESSG_OP_SELGROUP</td>
<td>4</td>
</tr>
<tr>
<td>ESSG_OP_INDENT</td>
<td>5</td>
</tr>
<tr>
<td>ESSG_OP_SUPMISSING</td>
<td>6</td>
</tr>
<tr>
<td>ESSG_OP_SUPZEROS</td>
<td>7</td>
</tr>
<tr>
<td>ESSG_OP_SUPUNDER</td>
<td>8</td>
</tr>
<tr>
<td>ESSG_OP_UPDATEMODE</td>
<td>9</td>
</tr>
<tr>
<td>ESSG_OP_ALIASNAMES</td>
<td>10</td>
</tr>
<tr>
<td>ESSG_OP_ALIASTABLE</td>
<td>11</td>
</tr>
<tr>
<td>ESSG_OP_USERGRIDDATA</td>
<td>12</td>
</tr>
<tr>
<td>ESSG_OP_RETAINTHREAD</td>
<td>20</td>
</tr>
<tr>
<td>ESSG_OP_EMPTYGRIDERROR</td>
<td>21</td>
</tr>
<tr>
<td>ESSG_OP_DRILLONLEAF</td>
<td>22</td>
</tr>
<tr>
<td>ESSG_OP_DATALESS</td>
<td>23</td>
</tr>
<tr>
<td>ESSG_OP_SPANHYBRIDANALYSIS</td>
<td>24</td>
</tr>
<tr>
<td>ESSG_OP_UNIQUENAMEONLY</td>
<td>32</td>
</tr>
<tr>
<td>ESSG_OP_MEMBERANDUNIQUENAME</td>
<td>33</td>
</tr>
</tbody>
</table>

(see "ESSG_OP_MEMBERANDUNIQUENAME Example" on page 1107)
<table>
<thead>
<tr>
<th>Constant</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESSG_OP_GET_ME.Cells</td>
<td>36</td>
</tr>
<tr>
<td>Return #ME (meaningless) value for cells</td>
<td>with no base member-attribute combination</td>
</tr>
<tr>
<td>Default: off</td>
<td></td>
</tr>
<tr>
<td>ESSG_OP_GET_FORMATTED.VALUE</td>
<td>38</td>
</tr>
<tr>
<td>Include formatting values for formatted cells</td>
<td></td>
</tr>
<tr>
<td>Default: Return only cell values</td>
<td></td>
</tr>
<tr>
<td>ESSG_OP_GET_VALUE</td>
<td>39</td>
</tr>
<tr>
<td>Requests original values for cells with non-</td>
<td>numeric types</td>
</tr>
<tr>
<td>Default: on</td>
<td></td>
</tr>
<tr>
<td>ESSG_OP_GET_FORMATTED.MISSING</td>
<td>40</td>
</tr>
<tr>
<td>Include formatting values for cells with</td>
<td>missing values</td>
</tr>
<tr>
<td>Default: off</td>
<td></td>
</tr>
<tr>
<td>ESSG_OP_GET_DRILLTHRU.URLS</td>
<td>41</td>
</tr>
<tr>
<td>Populates the drill through flag for cells</td>
<td></td>
</tr>
</tbody>
</table>

Describe the various indent styles

<table>
<thead>
<tr>
<th>Constant</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESSG_INDENTNONE</td>
<td>1</td>
</tr>
<tr>
<td>ESSG_INDENTSUBITEMS</td>
<td>2</td>
</tr>
<tr>
<td>ESSG_INDENTTOTALS</td>
<td>3</td>
</tr>
</tbody>
</table>

Used by get results calls to determine the process state

<table>
<thead>
<tr>
<th>Constant</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESSG_STATE_DONE</td>
<td>1</td>
</tr>
<tr>
<td>ESSG_STATE_INPROGRESS</td>
<td>2</td>
</tr>
</tbody>
</table>

Buffer length constants (including terminating null)

<table>
<thead>
<tr>
<th>Constant</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESSG_USERNAMELEN</td>
<td>31</td>
</tr>
<tr>
<td>ESSG_PASSWORDLEN</td>
<td>101</td>
</tr>
</tbody>
</table>
### Grid API Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESSG_SERVERLEN</td>
<td>31</td>
</tr>
<tr>
<td>ESSG_APPLICATIONLEN</td>
<td>9</td>
</tr>
<tr>
<td>ESSG_DATABASELEN</td>
<td>9</td>
</tr>
</tbody>
</table>

### Constants used by Grid API Drill-Through functions (EssGDTxxx())

<table>
<thead>
<tr>
<th>Constant</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESSG_DESCRIPTION_LEN</td>
<td>Maximum buffer length (255) used for report data</td>
</tr>
<tr>
<td>ESSG_DTINPUTOPTION_PROMPT_HISNAME</td>
<td><em>ulnputOption</em> value in <em>ESSG_DTINFO_T</em>, meaning that users have all the default values needed to connect to Essbase Studio and start a drill-through session</td>
</tr>
<tr>
<td>ESSG_DTINPUTOPTION_PROMPT_LOGIN</td>
<td><em>ulnputOption</em> value in <em>ESSG_DTINFO_T</em>, meaning that users must set the password to connect to Essbase Studio and start a drill-through session</td>
</tr>
<tr>
<td>ESSG_DTREPORT_NAME</td>
<td>Maximum string length (80) used for drill-through</td>
</tr>
<tr>
<td>ESSG_ERR_INVALIDDTHANDLE</td>
<td>Error message constant returned if the given drill-through instance handle is invalid</td>
</tr>
<tr>
<td>ESSG_ERR_NODTREPORTS</td>
<td>Error message constant returned if no drill-through report is defined for the given drill-through instance handle</td>
</tr>
<tr>
<td>ESSG_FIELDLEN</td>
<td>Maximum string length (30) used for drill-through</td>
</tr>
<tr>
<td>ESSG_HISDT</td>
<td>Value (5) used for drill-through entry</td>
</tr>
</tbody>
</table>

### Used by LRO API calls in the structure ESSG_LRODESC_T

<table>
<thead>
<tr>
<th>Constant</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESSG_PARTITIONTYPE</td>
<td>1</td>
</tr>
<tr>
<td>ESSG_CELLNOTETYPE</td>
<td>2</td>
</tr>
<tr>
<td>ESSG_WINAPPTYPE</td>
<td>3</td>
</tr>
<tr>
<td>ESSG_URLTYPE</td>
<td>4</td>
</tr>
</tbody>
</table>

### Grid Perspective Types

**Used by EssGGetGridPerspective and EssGSetGridPerspective.**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESSG_PERSP_EXPLICIT</td>
<td>0</td>
</tr>
</tbody>
</table>

Requires tuple specification
<table>
<thead>
<tr>
<th>Constant</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESSG_PERSP_REALITY</td>
<td>Uses reality context for attribute dimension</td>
</tr>
</tbody>
</table>

**Text List (SmartList) Types**

Text List (SmartList) attributes.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESSG_CA_MISSINGCELL</td>
<td>0x00000100</td>
</tr>
<tr>
<td></td>
<td>Set for cells of type SmartList when the cell has a #Missing value. This occurs for SmartList cells where #Missing values map to text values.</td>
</tr>
<tr>
<td>ESSG_CA_OUTOFRANGE</td>
<td>0x00000200</td>
</tr>
<tr>
<td></td>
<td>Set when a SmartList-type cell with a numeric value is out of range in the context of that text list</td>
</tr>
</tbody>
</table>

**Unicode Mode Types**

Used as values for `usApiType` field of ESSG_INIT_T for Unicode Mode.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESSG_API_UTF8</td>
<td>0x0003</td>
</tr>
<tr>
<td></td>
<td>This value enables Essbase Server to create or migrate Unicode-mode applications.</td>
</tr>
<tr>
<td>ESSG_API_NONUNICODE</td>
<td>0x0002</td>
</tr>
<tr>
<td></td>
<td>This value disables the creation and migration of Unicode-mode applications on Essbase Server.</td>
</tr>
</tbody>
</table>

**C Grid API Data Types**

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Essbase Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>typedef char ESSG_APPLICATION_T[ESSG_APPLICATIONLEN];</td>
<td>ESSG_APPLICATION_T</td>
</tr>
<tr>
<td>typedef unsigned char ESSG_BOOL_T;</td>
<td>ESSG_BOOL_T</td>
</tr>
<tr>
<td>typedef char ESSG_CHAR_T;</td>
<td>ESSG_CHAR_T</td>
</tr>
<tr>
<td>typedef char ESSG_DATABASE_T[ESSG_DATABASELEN];</td>
<td>ESSG_DATABASE_T</td>
</tr>
<tr>
<td>typedef double ESSG_DOUBLE_T;</td>
<td>ESSG_DOUBLE_T</td>
</tr>
<tr>
<td>typedef ESSG_PVOID_T ESSG_DTHINST_T, *ESSG_PDTHINST_T</td>
<td>ESSG_DTHINST_T, ESSG_PDTHINST_T</td>
</tr>
<tr>
<td>typedef float ESSG_FLOAT_T;</td>
<td>ESSG_FLOAT_T</td>
</tr>
<tr>
<td>typedef ESSG_PVOID_T ESSG_HANDLE_T, *ESSG_PHANDLE_T;</td>
<td>ESSG_HANDLE_T, ESSG_PHANDLE_T</td>
</tr>
<tr>
<td>Data Type</td>
<td>Essbase Type</td>
</tr>
<tr>
<td>-------------------------------------------------------------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>typedef ESSG_PVOID_T ESSG_HGRID_T,*ESSG_PHGRID_T;</td>
<td>ESSG_HGRID_T, ESSG_PHGRID_T</td>
</tr>
<tr>
<td>typedef long ESSG_LONG_T;</td>
<td>ESSG_LONG_T</td>
</tr>
<tr>
<td>typedef char ESSG_PASSWORD_T[ESSG_PASSWORDLEN];</td>
<td>ESSG_PASSWORD_T</td>
</tr>
<tr>
<td>typedef char *ESSG_PSTR_T;</td>
<td>ESSG_PSTR_T</td>
</tr>
<tr>
<td>typedef ESSG_VOID_T *ESSG_PVOID_T;</td>
<td>ESSG_PVOID_T</td>
</tr>
<tr>
<td>typedef char ESSG_SERVER_T[ESSG_SERVERLEN];</td>
<td>ESSG_SERVER_T</td>
</tr>
<tr>
<td>typedef short ESSG_SHORT_T;</td>
<td>ESSG_SHORT_T</td>
</tr>
<tr>
<td>typedef char *ESSG_STR_T;</td>
<td>ESSG_STR_T</td>
</tr>
<tr>
<td>typedef long ESSG_STS_T;</td>
<td>ESSG_STS_T</td>
</tr>
<tr>
<td>typedef unsigned char ESSG_UCHAR_T;</td>
<td>ESSG_UCHAR_T</td>
</tr>
<tr>
<td>typedef unsigned long ESSG_ULONG_T;</td>
<td>ESSG_ULONG_T</td>
</tr>
<tr>
<td>typedef char ESSG_USERNAME_T[ESSG_USERNAMELEN];</td>
<td>ESSG_USERNAME_T</td>
</tr>
<tr>
<td>typedef unsigned short ESSG_USHORT_T;</td>
<td>ESSG_USHORT_T</td>
</tr>
<tr>
<td>typedef void ESSG_VOID_T;</td>
<td>ESSG_VOID_T</td>
</tr>
<tr>
<td>typedef unsigned short ESSG_WORD_T;</td>
<td>ESSG_WORD_T</td>
</tr>
</tbody>
</table>

**ESSG_PFUNC_T, ESSG_PFUNC_M**

These types define the prototype for a user’s message callback function.

```c
#ifdef WIN32
#define ESSG_CALLBACK _export
#define ESSG_FUNC_M   ESSG_STS_T ESSG_CALLBACK /* for Win32 */
#else
#define ESSG_CALLBACK _export
#define ESSG_FUNC_M   ESSG_STS_T ESSG_CALLBACK /* for other platforms */
#endif

#ifdef WIN32
// function pointer (Win32) */
typedef
ESSG_STS_T (ESSG_CALLBACK *ESSG_PFUNC_T)(ESSG_PVOID_T, ESSG_LONG_T,
    ESSG_USHORT_T, ESSG_STR_T, ESSG_STR_T);
#else
// function pointer (other) */
typedef
ESSG_STS_T (ESSG_CALLBACK *ESSG_PFUNC_T)(ESSG_PVOID_T, ESSG_LONG_T,
    ESSG_USHORT_T, ESSG_STR_T, ESSG_STR_T);
#endif
```
C Grid API Structures

This section describes the structures used by the Grid API. Click on one of the structure names below to navigate to the description.

- “ESSG_CONNECTINFO_T” on page 1006
- “ESSG_DATA_T” on page 1006
- “ESSG_DRILLDATA_T” on page 1008
- “ESSG_DTDATA_T” on page 1009
- “ESSG_DTHEADER_T” on page 1009
- “ESSG_DTINFO_T” on page 1010
- “ESSG_DTREPORT_T” on page 1010
- “ESSG_INIT_T” on page 1011
- “ESSG_LRODESC_T” on page 1011
- “ESSG_LROINFO_T” on page 1012
- “ESSG_RANGE_T” on page 1012

**ESSG_CONNECTINFO_T**

Contains information about database connection for each linked partition. The fields are described as follows:

```c
typedef struct ESSG_CONNECTINFO_T
{
    ESSG_SERVER_T       Server;
    ESSG_APPLICATION_T  Application;
    ESSG_DATABASE_T     Database;
    ESSG_USERNAME_T     Username;
    ESSG_PASSWORD_T     Password;
} ESSG_CONNECTINFO_T, * ESSG_PCONNECTINFO_T, ** ESSG_PPCONNECTINFO_T;
```

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESSG_SERVER_T</td>
<td>Server</td>
<td>Name of the server</td>
</tr>
<tr>
<td>ESSG_APPLICATION_T</td>
<td>Application</td>
<td>Name of the application</td>
</tr>
<tr>
<td>ESSG_DATABASE_T</td>
<td>Database</td>
<td>Name of the Essbase database</td>
</tr>
<tr>
<td>ESSG_USERNAME_T</td>
<td>Username</td>
<td>User's name</td>
</tr>
<tr>
<td>ESSG_PASSWORD_T</td>
<td>Password</td>
<td>User's password</td>
</tr>
</tbody>
</table>

**ESSG_DATA_T**

Describes the format of the data to be sent and received by the Essbase Grid API. Note that calls returning this structure will return member names in the Member structure. The caller can pass
in the same structure back to the API using the Member structure instead of the pszStr field if the type is ESSG_DT_MEMBER.

The ESSG_DATA_T data structure defines each cell sent or returned via the grid API. If this structure is being returned to the caller, pszStr contains string data and dblData contains numeric data. Use the usType field to determine whether the cell is a member, a number, or text. Similarly, if the structure is being passed into the API, pszStr should contain a member name or text and dblData should contain numeric data. Set the usType field to correspond to the data type of the cell. If the cell data type is unknown, set it to text (ESSG_DT_STRING), and the server determines whether it is a member.

```c
typedef struct ESSG_DATA_T
{
    ESSG_PVOID_T     pAttributes;
    ESSG_DATA_VALUE Value;
    ESSG_USHORT_T    usType;
    ESSG_PVOID_T     pCellProps;
} ESSG_DATA_T;

ESS_TSA_API_typedef(ESSG_DATA_T *,  ESSG_PDATA_T);
ESS_TSA_API_typedef(ESSG_DATA_T **, ESSG_PPDATA_T);
```

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESSG_PVOID_T</td>
<td>pAttributes</td>
<td>One of the long integer constants listed below indicating the cell type or member type (OUT)</td>
</tr>
<tr>
<td>ESSG_DATA_VALUE_T</td>
<td>Value</td>
<td>The value of the returned grid string</td>
</tr>
<tr>
<td>ESSG_USHORT_T</td>
<td>usType</td>
<td>One of the tag constants listed below indicating the data type (IN/OUT)</td>
</tr>
<tr>
<td>ESSG_PVOID_T</td>
<td>pCellProps</td>
<td>Stores cell properties; for example, whether or not cell is associated with a drill-through URL</td>
</tr>
</tbody>
</table>

**Constants for ESSG_DATA_T**

The following constants are used by the pAttributes field of the ESSG_DATA_T structure for cell data types:

- ESSG_CA_READONLY
- ESSG_CA_READWRITE
- ESSG_CA_LINKEDOBJ
- ESSG_CA_LINKPARTITION
- ESSG_CA_LINKCELLNOTE
- ESSG_CA_LINKWINAPP
- ESSG_CA_LINKURL
- ESSG_CA_AISDT
- ESSG_CA_GLDT

The following constants are used by the pAttributes field of the ESSG_DATA_T structure for member data types:

- ESSG_MA_DIMTOP
- ESSG_MA_ZOOMINABLE
- ESSG_MA_NEVERSHARE
- ESSG_MA_LABELONLY
- ESSG_MASTOREDATA
- ESSG_MA_EXPSHARE
The following constants are used by the usType field of the ESSG_DATA_T structure:

- ESSG_DT_UNUSED
- ESSG_DT_STRING
- ESSG_DT_LONG
- ESSG_DT_DOUBLE
- ESSG_DT_BLANK
- ESSG_DT_RESERVED
- ESSG_DT_ERROR
- ESSG_DT_MISSING
- ESSG_DT_ZERO
- ESSG_DT_NOACCESS
- ESSG_DT_MEMBER
- ESSG_DT_FORMULA
- ESSG_DT_ZEROwFORMULA
- ESSG_DT_DOUBLEwFORMULA
- ESSG_DT_BLANKwFORMULA
- ESSG_DT_STRINGwFORMULA
- ESSG_DT_MISSINGwFORMULA
- ESSG_DT_NOACCESSwFORMULA
- ESSG_DT_STRINGEX
- ESSG_DT_FORMULASEX
- ESSG_DT_MEMBEREX
- ESSG_DT_STRINGEXwFORMULA
- ESSG_DT_FORMULASEXwFORMULA
- ESSG_DT_MEMBERwKEY

The following constants are additional values for the usType field, to work in Unicode mode.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESSG_DT_STRINGEX</td>
<td>0x0018</td>
<td>This value specifies a string extended for Unicode mode.</td>
</tr>
<tr>
<td>ESSG_DT_MEMBEREX</td>
<td>0x0019</td>
<td>This value specifies a member name extended for Unicode mode.</td>
</tr>
<tr>
<td>ESSG_DT_STRINGEXwFORMULA</td>
<td>0x0020</td>
<td>This value specifies a formula string extended for Unicode mode.</td>
</tr>
<tr>
<td>ESSG_DT_FORMULASEX</td>
<td>0x0021</td>
<td>This value specifies a formula extended for Unicode mode.</td>
</tr>
</tbody>
</table>

**ESSG_DRILLDATA_T**

Contains information associating linked objects with specific cell addresses. The fields are described as follows:

```c
typedef struct ESSG_DRILLDATA_T
{
    ESSG_HLRO_T       hLRO;
    ESSG_USHORT_T     usLinkObjType;
    ESSG_LINKOBJDESC  Description;
    ESSG_PSTR_T       pMbrCombos;
} ESSG_DRILLDATA_T;
```
```c
typedef struct ESSG_DTDATA_T
{
    ESSG_ULONG_T row;
    ESSG_ULONG_T column;
    ESSG_CHAR_T  data[ESSG_DESCRIPTION_LEN + 1];
} ESSG_DTDATA_T, *ESSG_PDTDATA_T, **ESSG_PPDTDATA_T;
```

### ESSG_DTDATA_T

Defines a report data cell.

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESSG_ULONG_T</td>
<td>row</td>
<td>0-indexed row number for the given data block</td>
</tr>
<tr>
<td>ESSG_ULONG_T</td>
<td>column</td>
<td>0-indexed column number for the given data block</td>
</tr>
<tr>
<td>ESSG_CHAR_T</td>
<td>data</td>
<td>Data value for the given data block</td>
</tr>
</tbody>
</table>

```c
typedef struct ESSG_DTHEADER_T
{
    ESSG_ULONG_T         colIndex;
    ESSG_CHAR_T          viewName[ESSG_DESCLEN + 1];
    ESSG_CHAR_T          data[ESSG_DESCLEN + 1];
    ESSGDTREPORTDATATYPE dataType;
} ESSG_DTHEADER_T, *ESSG_PDTHEADER_T, **ESSG_PPDTHEADER_T;
```

### ESSG_DTHEADER_T

Defines header information for a specific column.

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESSG_ULONG_T</td>
<td>colIndex</td>
<td>0-based index of the column position</td>
</tr>
<tr>
<td>ESSG_CHAR_T</td>
<td>viewName</td>
<td>Heading text for the given column of data</td>
</tr>
<tr>
<td>ESSG_CHAR_T</td>
<td>data</td>
<td>Data value for the given data block</td>
</tr>
<tr>
<td>Data Type</td>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ESSGDTREPORTDATATYPE</td>
<td>dataType</td>
<td>One of the constants listed below indicating the data type of the given column of data</td>
</tr>
</tbody>
</table>

### Constants for ESSG_DTHEADER_T

The following constants are used by the `dataType` field of the ESSG_DTHEADER_T structure:

- ESSGDTINT
- ESSGDTFLOAT
- ESSGDTSTRING

**ESSG_DTINFO_T**

Defines the connection information for a range of data cells.

```c
typedef struct ESSG_DTINFO_T
{
    ESSG_CHAR_T   hisName[ESSG_FIELDLEN + 1];
    ESSG_CHAR_T   dataSource[ESSG_FIELDLEN + 1];
    ESSG_CHAR_T   username[ESSG_FIELDLEN + 1];
    ESSG_CHAR_T   password[ESSG_FIELDLEN + 1];
    ESGG_USHORT_T inputOption;
} ESSG_DTINFO_T, *ESSG_PDTINFO_T, **ESSG_PPDTINFO_T;
```

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESSG_CHAR_T</td>
<td>hisName</td>
<td>[ESSG_FIELDLEN + 1]</td>
</tr>
<tr>
<td>ESSG_CHAR_T</td>
<td>dataSource</td>
<td>(read only)</td>
</tr>
<tr>
<td>ESSG_CHAR_T</td>
<td>username</td>
<td></td>
</tr>
<tr>
<td>ESSG_CHAR_T</td>
<td>password</td>
<td>(write only)</td>
</tr>
<tr>
<td>ESSG_USHORT_T</td>
<td>inputOption</td>
<td>(read only)</td>
</tr>
</tbody>
</table>

**ESSG_DTREPORT_T**

Defines a report definition.

```c
typedef struct ESSG_DTREPORT_T
{
    ESSG_LONG_T reportId;
    ESSG_CHAR_T name[ESSG_DESCLEN + 1];
    ESSG_LONG_T customize;
    ESSG_LONG_T rowGoverner;
    ESSG_LONG_T timeGoverner;
} ESSG_DTREPORT_T, *ESSG_PDTREPORT_T, **ESSG_PPDTREPORT_T;
```
<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESSG_LONG_T</td>
<td>reportId</td>
<td></td>
</tr>
<tr>
<td>ESSG_CHAR_T</td>
<td>name</td>
<td>[ESSG_DESCLEN + 1]</td>
</tr>
<tr>
<td>ESSG_LONG_T</td>
<td>customize</td>
<td></td>
</tr>
<tr>
<td>ESSG_LONG_T</td>
<td>rowGoverner</td>
<td></td>
</tr>
<tr>
<td>ESSG_LONG_T</td>
<td>timeGoverner</td>
<td></td>
</tr>
</tbody>
</table>

**ESSG_INIT_T**

Describes the information to be passed into the call to `EssGInit`.

typedef struct
{
    ESSG_ULONG_T ulVersion;
    ESSG_ULONG_T ulMaxRows;
    ESSG_ULONG_T ulMaxColumns;
    ESSG_PFUNC_T pfnMessageFunc;
    ESSG_PVOID_T pUserdata;
    ESSG_USHORT_T usApiType;
} ESSG_INIT_T, *ESSG_PINIT_T;

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESSG_ULONG_T</td>
<td>ulVersion</td>
<td>This should be set to ESSG_VERSION</td>
</tr>
<tr>
<td>ESSG_ULONG_T</td>
<td>ulMaxRows</td>
<td>Maximum number of rows for the grid</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Limit: 65535 rows</td>
</tr>
<tr>
<td>ESSG_ULONG_T</td>
<td>ulMaxColumns</td>
<td>Maximum number of columns for the grid</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Limit: 256 columns</td>
</tr>
<tr>
<td>ESSG_PFUNC_T</td>
<td>pfnMessageFunc</td>
<td>Pointer to the user-defined message callback function</td>
</tr>
<tr>
<td>ESSG_PVOID_T</td>
<td>pUserdata</td>
<td>Pointer to user data passed to message callback</td>
</tr>
<tr>
<td>ESSG_USHORT_T</td>
<td>usApiType</td>
<td>Encoding type of the Grid API. For valid values, see &quot;Unicode Mode Types&quot; on page 1004.</td>
</tr>
</tbody>
</table>

**ESSG_LRODESC_T**

Contains information describing a specific object linked to a data cell in an Essbase database.
The fields are described as follows:

typedef struct ESSG_LRODESC_T
{
    ESSG_USHORT_T usLinkObjType;
    ESSG_USERNAME_T Username;
    ESSG_TIME_T LastUpdate;
    union


```c
{
    ESSG_LROINFO_T lroInfo;
    ESSG_CHAR_T     Note[ESSG_LRONOTELEN];
} lro;
} ESSG_LRODESC_T, *ESSG_LPLRODESC_T;
```

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESSG_ULONG_T</td>
<td>usLinkObjType</td>
<td>Object type</td>
</tr>
<tr>
<td>ESSG_USERNAME_T</td>
<td>userName</td>
<td>Name of the last user to modify the object</td>
</tr>
<tr>
<td>ESSG_TIME_T</td>
<td>LastUpdate</td>
<td>Last date the object was modified</td>
</tr>
<tr>
<td>ESSG_LROINFO_T</td>
<td>lroInfo</td>
<td>LRO information structure, associated by union</td>
</tr>
<tr>
<td>ESSG_CHAR_T</td>
<td>Note[ESSG_LRONOTELEN]</td>
<td>A cell note, associated by union</td>
</tr>
</tbody>
</table>

The default note length specified by ESSG_LRONOTELEN is 599.

### ESSG_LROINFO_T

Contains information about a specific object linked to a data cell in an Essbase database. The fields are described as follows:

```c
typedef struct ESSG_LROINFO_T
{
    ESSG_CHAR_T  ObjName[ESSG_ONAMELEN];
    ESSG_CHAR_T  Desc[ESS_DESCLEN];
} ESSG_LROINFO_T, *ESSG_LPLROINFO_T;
```

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESSG_CHAR_T</td>
<td>objName[ESSG_ONAMELEN]</td>
<td>Source file name of object linked to a data cell</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ESSG_ONAMELEN specifies the maximum length of an object name; the default value is 511.</td>
</tr>
<tr>
<td>ESSG_CHAR_T</td>
<td>Desc[ESS_DESCLEN]</td>
<td>Description of an object linked to a data cell</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ESS_DESCLEN specifies the maximum length of the description; the default value is 79.</td>
</tr>
</tbody>
</table>

### ESSG_RANGE_T

Describes the extent of the data being sent or received.

```c
typedef struct
{
    ESSG_ULONG_T ulRowStart;
    ESSG_ULONG_T ulColumnStart;
    ESSG_ULONG_T ulNumRows;
    ESSG_ULONG_T ulNumColumns;
} ESSG_RANGE_T, *ESSG_P RANGE_T;
```

---

1012 C Grid API Declarations
<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESSG_ULONG_T</td>
<td>ulRowStart</td>
<td>First Row in the report (zero based)</td>
</tr>
<tr>
<td>ESSG_ULONG_T</td>
<td>ulColumnStart</td>
<td>First Column in the report (zero based)</td>
</tr>
<tr>
<td>ESSG_ULONG_T</td>
<td>ulNumRows</td>
<td>Number of rows in the report (maximum 16370)</td>
</tr>
<tr>
<td>ESSG_ULONG_T</td>
<td>ulNumColumns</td>
<td>Number of columns in the report (maximum 256)</td>
</tr>
</tbody>
</table>
Consult the Contents pane for the alphabetical list of C Grid API functions, which are prefaced with EssG.

**EssGBeginConditionalRetrieve**

Begins a conditional retrieval operation.

**Syntax**

```c
ESSG_FUNC_M EssGBeginConditionalRetrieve (hGrid, pszConditions, ulOptions);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hGrid</td>
<td>ESSG_HGRID_T</td>
<td>Handle passed back from EssGNewGrid.</td>
</tr>
<tr>
<td>pszConditions</td>
<td>ESSG_STR_T</td>
<td>String (no greater than 64K) containing Essbase report specification commands relating to the conditions for the retrieval. Do not use Report Writer member/alias/unique name handling formatting commands for the pszConditions parameter. Use the options available in the EssGSetGridOption function.</td>
</tr>
<tr>
<td>ulOptions</td>
<td>ESSG_ULONG_T</td>
<td>A constant which describes the type of retrieval. One of the following values must be used:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESSG_RET_RETRIEVE Retrieve Only</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESSG_RET_RETRIEVELOCK Retrieve and Lock</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESSG_RET_LOCKONLY Lock Only (No data is to be retrieved)</td>
</tr>
</tbody>
</table>

The following value may be added into `ulOptions` using bitwise OR (|): ESSG_NOATTRIBUTES returns grid without `pAttributes` values.

**Notes**

- Conditions, as defined in a partial report specification, are applied to the provided grid.
- Attributes for returned cell values are obtained using a second server request. Passing ESSG_NOATTRIBUTES in the `ulOptions` parameter will issue one less request of the server, and could, in large resulting grids, be faster.
- In case of Type-enabled applications, such as applications with SmartList, Date, or Format strings, you will get textual encoded data but without type information if you specify ESSG_NOATTRIBUTES. Type information works like member attributes, so do not use ESSG_NOATTRIBUTES if type information is required.
Return Value

If successful, returns ESS_STS_NOERR.

Access

None.

Example

```c
ESSG_VOID_T ESSG_BeginConditionalRetrieve(ESSG_HGRID_T hGrid)
{
    ESSG_STS_T     sts = ESS_STS_NOERR;
    ESSG_PPDATA_T   ppDataIn;
    ESSG_PPDATA_T   ppDataOut;
    ESSG_RANGE_T   rDataRangeIn, rDataRangeOut;
    ESSG_ULONG_T   ulOptions;
    ESSG_USHORT_T   usState;
    ESSG_STR_T   pszConditions;

    /* connect the grid to a database on the server */
    sts = EssGConnect(hGrid, "Rainbow", "Admin",
                      "Password", "Demo", "Basic", ESSG_CONNECT_DEFAULT);

    if(sts == 0)
    {
        ppDataIn = BuildTable(&rDataRangeIn);

        ulOptions = ESSG_RET_RETRIEVE;
        pszConditions = "<TOP(Scenario,3,@Datacol(3))";
        /* start the conditional retrieve operation */
        sts = EssGBeginConditionalRetrieve(hGrid,
                                            pszConditions, ulOptions);
    }

    if(sts == 0)
    {
        /* send the entire grid to define the query */
        sts = EssGSendRows(hGrid, &rDataRangeIn, pDataIn);
    }

    if(sts == 0)
    {
        /* perform the retrieval */
        sts = EssGPerformOperation(hGrid, 0);

        /* free the built data */
        FreeTwoDim(ppDataIn, rDataRangeIn.ulNumRows);
    }

    if(sts == 0)
    {
        /* determine the results of the retrieve */
        sts = EssGGetResults(hGrid, 0, &rDataRangeOut,
                              &usState);
    }

    if(sts ==0)
    {
        /* get all the data */
    }
}```
sts = EssGGetRows(hGrid, 0, &rDataRangeOut, &DataRangeOut, &ppDataOut);
}

if(sts == 0)
{
    /* display the results */
    DisplayOutput(ppDataOut, rDataRangeOut);
    /* free the returned data */
    EssGFreeRows(hGrid, &rDataRangeOut, ppDataOut);
}

if(!sts)
{
    EssGEndOperation(hGrid, 0);
    EssGDisconnect(hGrid, 0);
}

See Also

- “Using the C Grid API Functions” on page 994
- “C Grid API Structures” on page 1006

**EssGBeginConditionalZoomIn**

Begins a conditional zoom-in.

**Syntax**

```c
ESSG_FUNC_M EssGBeginConditionalZoomIn (hGrid, pZoomCell, pszConditions, ulOptions);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hGrid</td>
<td>ESSG_HGRID_T</td>
<td>Handle passed back from EssGNewGrid.</td>
</tr>
<tr>
<td>pZoomCell</td>
<td>&quot;ESSG_RANGE_T&quot; on page 1012</td>
<td>Describes the cell to be zoomed in upon. This must be a single cell for</td>
</tr>
<tr>
<td></td>
<td></td>
<td>conditional zoomin.</td>
</tr>
<tr>
<td>pszConditions</td>
<td>ESSG_STR_T</td>
<td>String (no greater than 64K) containing Essbase report specification commands</td>
</tr>
<tr>
<td></td>
<td></td>
<td>relating to the conditions for the zoom-in. Do not use Report Writer member/</td>
</tr>
<tr>
<td></td>
<td></td>
<td>alias/unique name handling formatting commands for the pszConditions</td>
</tr>
<tr>
<td></td>
<td></td>
<td>parameter. Use the options available in EssGSetGridOption.</td>
</tr>
</tbody>
</table>

EssGBeginConditionalZoomln 1017
### Parameter
- **ulOptions**: ESSG_ULONG_T
  
  A bitmask which describes the type of zoom-in (across or down). The following two values are mutually exclusive:
  - ESSG_ZOOM_DOWN: Any page/title dimensions selected will be zoomed down.
  - ESSG_ZOOM_ACROSS: Any page dimensions selected will be zoomed across.

  The following option may be added into `ulOptions` using bitwise OR (|):
  - ESSG_NOATTRIBUTES: returns grid without `pAttributes` values.

### Notes
- The cell to be zoomed in upon is described by a single range, the conditions to be applied are passed as a string containing Essbase report specification commands.
- Attributes for returned cell values are obtained using a second server request. Passing ESSG_NOATTRIBUTES for the `ulOptions` parameter will issue one less request of the server, and could, in large resulting grids, be faster.
- Conditional zoom-in will only work on one zoom cell at a time.
- There are only three valid zoom levels when doing a conditional ZoomIn - ESSG_NEXTLEVEL, ESSG_BOTTOMLEVEL, or ESSG_ALLLEVELS. The Zoom level for conditional ZoomIn must be set via `EssGSetGridOption` to one of the three valid levels. If a non-valid level is set when performing a conditional ZoomIn the API will default to ESSG_NEXTLEVEL.
- If the zoom level is ESSG_BOTTOMLEVEL the resulting members are selected based on the conditions from all Level zero members in the dimension being zoomed on. For example, if the zoom cell contains East, from the Market dimension, and the zoom level is ESS_BOTTOMLEVEL, the resulting members could be any of the leaf members of Market, not just descendents of East.
- In case of Type-enabled applications, such as applications with SmartList, Date, or Format strings, you will get textual encoded data but without type information if you specify ESSG_NOATTRIBUTES. Type information works like member attributes, so do not use ESSG_NOATTRIBUTES if type information is required.

### Return Value
If successful, returns ESSG_STS_NOERR.

### Access
None.

### Example

```
ESSG_VOID_T ESSG_BeginConditionalZoomIn(ESSG_HGRID_T hGrid)
{
    ESSG_FUNC_M     sts = ESS_STS_NOERR;
    ESSG_PPDATA_T         ppDataIn;
    ESSG_PPDATA_T         ppDataOut;
```

---

1018 C Grid API Function Reference
ESSG_RANGE_T     rDataRangeIn, rDataRangeOut;
ESSG_ULONG_T      ulOptions;
ESSG_RANGE_T      pZoomCells;
ESSG_USHORT_T     usState;
ESSG_STR_T        pszConditions;

/* connect the grid to a database on the server */
sts = EssGConnect(hGrid, "Rainbow", "Admin",  
                 "Password", "Demo", "Basic", ESSG_CONNECT_DEFAULT);

if(sts == 0)
{
    ppDataIn = BuildTable(&rDataRangeIn);

    ulOptions = ESSG_ZOOM_DOWN | ESSG_ALLLEVELS;

    pZoomCells.ulRowStart = 0;
    pZoomCells.ulColumnStart = 2;
    pZoomCells.ulNumRows = 1;
    pZoomCells.ulNumColumns = 1;
    pszConditions = "<TOP("Scenario",3,@Datacol(3))"

    /* start the conditional zoom-in operation */
    sts = EssGBeginConditionalZoomIn(hGrid,  
                                    &pZoomCells, pszConditions, ulOptions);
}

if(sts == 0)
{
    /* send the entire grid to define the query */
    sts = EssGSendRows(hGrid, &rDataRangeIn,  
                       ppDataIn);
}

if(sts == 0)
{
    /* perform the conditional zoom-in */
    sts = EssGPerformOperation(hGrid, 0);

    /* Free the built data */
    FreeTwoDim(ppDataIn, rDataRangeIn.ulNumRows);
}

if(sts == 0)
{
    /* determine the results of conditional zoom-in */
    sts = EssGGetResults(hGrid, 0, &rDataRangeOut, &usState);
}

if(sts == 0)
{
    /* get all the data */
    sts = EssGGetRows(hGrid, 0, &rDataRangeOut,  
                       &rDataRangeOut, &ppDataOut);
}

if(sts == 0)
{
    DisplayOutput(ppDataOut, rDataRangeOut);
See Also

- “Using the C Grid API Functions” on page 994
- “C Grid API Structures” on page 1006

## EssGBeginCreateLRO

Begins the operation of creating a linked object for a data cell in an Essbase database.

### Syntax

```c
ESSG_FUNC_M EssGBeginCreateLRO (hGrid, usCells, pCells, pLroDesc, ulOption);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hGrid;</td>
<td>ESSG_HGRID_T</td>
<td>Grid handle returned by EssGNewGrid().</td>
</tr>
<tr>
<td>usCells;</td>
<td>ESSG_USHORT_T</td>
<td>The number of cell ranges specified in pCells.</td>
</tr>
<tr>
<td>pCells;</td>
<td>“ESSG_RANGE_T” on page 1012</td>
<td>Array of cell ranges for which to create the link.</td>
</tr>
<tr>
<td>pLroDesc;</td>
<td>“ESSG_LRODESC_T” on page 1011</td>
<td>LRO description information for the new object.</td>
</tr>
<tr>
<td>ulOption;</td>
<td>ESSG_ULONG_T</td>
<td>Option specifying whether to store the object on the server. Use ESS_STORE_OBJECT_API to store winapp and URL objects on the server. Use ESS_NOSTORE_OBJECT_API to store cell notes off the server (and in the index file).</td>
</tr>
</tbody>
</table>

### Return Value

If successful, returns ESSG_STS_NOERR.

### See Also

- “Using the C Grid API Functions” on page 994
- “C Grid API Structures” on page 1006
- EssGBeginDeleteLROs
- EssGBeginDrillOrLink
- EssGDeleteLRO
- EssGFreeCellLinkResults
EssGBeginDataPoint

Begins a data point operation.

**Syntax**

```c
ESSG_FUNC_M EssGBeginDataPoint (hGrid, ulRow, ulColumn, ulOptions);
```

**Parameter** | **Data Type** | **Description**
--- | --- | ---
 hGrid | ESSG_HGRID_T | Handle passed back from EssGNewGrid.
 ulRow | ESSG_ULONG_T | Row of the data point.
 ulColumn | ESSG_ULONG_T | Column of the data point.
 ulOptions | ESSG_ULONG_T | Reserved for future use. Should be set to zero.

**Notes**

- This function returns one member from each dimension describing the combination of members for a particular cell in the grid.
- The caller should pass in (EssGSendRows) enough information for Essbase to determine the members for the cell. It is safest to pass in all rows less than or equal to the `ulRow` parameter and all columns. The `ulRow` and `ulColumn` values are zero-based.

**Return Value**

If successful, returns ESSG_STS_NOERR.

**Access**

None.

**Example**

```c
ESSG_VOID_T EssGBeginDataPoint(ESSG_HGRID_T hGrid)
{
    ESSG_FUNC_M    sts = ESS_STS_NOERR;
    ESSG_ULONG_T         ulRow;
    ESSG_ULONG_T         ulColumn;
    ESSG_ULONG_T         ulOptions;
    ESSG_PPDATA_T        ppDataIn;
    ESSG_RANGE_T         rDataRangeIn;
    ESSG_ULONG_T         ulMembers, i;
    ESSG_PSTR_T          ppszMembers;
    ESSG_USHORT_T        usState;

    /* connect the grid to a database on the server */
```
sts = EssGConnect(hGrid, "Rainbow", "Admin", 
                     "Password", "Demo", "Basic", ESSG_CONNECT_DEFAULT);

if(sts == 0)
{
    ppDataIn = BuildTable(&rDataRangeIn);

    ulRow = 1;
    ulColumn = 2;
    ulOptions = 0;

    /* start the data point operation */
    sts = EssGBeginDataPoint(hGrid, ulRow, ulColumn, ulOptions);
}

if(sts == 0)
{
    /* send the entire grid to define the query */
    sts = EssGSendRows(hGrid, &rDataRangeIn,
                        ppDataIn);
}

if(sts == 0)
{
    /* perform the data point operation */
    sts = EssGPerformOperation(hGrid, 0);

    /* free the built data */
    FreeTwoDim(ppDataIn, rDataRangeIn.ulNumRows);
}

if(sts == 0)
{
    /* determine the results of the data point operation */
    sts = EssGGetDataPointResults(hGrid, &ulMembers, 
                                  &ppszMembers, &usState);
}

if(!sts && ulMembers)
{
    printf("\nMembers: (*nMembers: *n");
    for (i = 0; i<ulMembers; i++)
        printf("\n\t%s", ppszMembers[i]);

    EssGFreeMemberInfo(hGrid, ulMembers, ppszMembers);
}

if(!sts)
{
    EssGEndOperation(hGrid, 0);
    EssGDisconnect(hGrid, 0);
}

See Also

- “Using the C Grid API Functions” on page 994
- “C Grid API Structures” on page 1006
EssGBeginDeleteLROs

Begin the operation of deleting all objects linked to a data cell in an Essbase database.

**Syntax**

```
ESSG_FUNC_M EssGBeginDeleteLROs (hGrid, usCells, pCells);
```

**Parameter** | **Data Type** | **Description**
--- | --- | ---
 hGrid; | ESSG_HGRID_T | Grid handle returned by EssGNewGrid().
 usCells; | ESSG_USHORT_T | The number of cell ranges specified in pCells.
 pCells; | “ESSG_RANGE_T” on page 1012 | Array of cell ranges for which to delete linked objects.

**Notes**

To delete a single LRO use EssGDeleteLRO.

**Return Value**

If successful, returns ESSG_STS_NOERR.

**See Also**

- “Using the C Grid API Functions” on page 994
- “C Grid API Structures” on page 1006
- EssGBeginCreateLRO
- EssGDeleteLRO
- EssGFreeCellLinkResults
- EssGGetCellLinkResults
- EssGGetLRODesc
- EssGGetLRO
- EssGUpdateLRO

EssGBeginDrillAcross

Begin a `drill-across` to retrieve cells from a linked partition.

**Syntax**

```
ESSG_FUNC_M EssGBeginDrillAcross (hGrid, hDAGrid, hLRO, usCells, pDrillCells, usOption);
```

**Parameter** | **Data Type** | **Description**
--- | --- | ---
 hGrid; | ESSG_HGRID_T | Handle of the original grid returned by EssGNewGrid().
 hDAGrid; | ESSG_HGRID_T | Handle of new grid to receive drill results.
 hLRO; | ESSG_HLRO_T | Handle of the linked partition.
 usCells; | ESSG_USHORT_T | The number of cell ranges specified in pDrillCells.
### Parameter | Data Type | Description
--- | --- | ---
pDrillCells; “ESSG_RANGE_T” on page 1012 | Array of cell ranges associated with the linked partition.
ulOption; ESSG_ULONG_T | Option specifying whether to return Zoom-In results if sent by the server. Use:
- ESSG_OPT_ZOOM to return Zoom-In results.
- ESSG_OPT_NOZOOM to suppress Zoom-In results.

### Return Value
If successful, returns ESSG_STS_NOERR.

### See Also
- “Using the C Grid API Functions” on page 994
- “C Grid API Structures” on page 1006
- EssGBeginRemoveOnly
- EssGGetCellLinkResults
- EssGBeginDrillOrLink

---

## EssGBeginDrillOrLink

Begins the operation of querying the links associated with one or more data cells in an Essbase database.

### Syntax

```c
ESSG_FUNC_M EssGBeginDrillOrLink (hGrid, usCells, pDrillCells, ulOptions);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hGrid;</td>
<td>ESSG_HGRID_T</td>
<td>Grid handle returned by EssGNewGrid().</td>
</tr>
<tr>
<td>usCells;</td>
<td>ESSG_USHORT_T</td>
<td>The number of cell ranges in the array of ranges specified in pDrillCells.</td>
</tr>
<tr>
<td>pDrillCells; “ESSG_RANGE_T” on page 1012</td>
<td>Array of cell ranges to query for links.</td>
<td></td>
</tr>
<tr>
<td>ulOptions;</td>
<td>ESSG_ULONG_T</td>
<td>Option specifying whether to return Zoom-In results if sent by the server. Use:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESSG_OPT_ZOOM to return Zoom-In results.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESSG_OPT_NOZOOM to suppress Zoom-In results.</td>
</tr>
</tbody>
</table>

### Return Value
If successful, returns ESSG_STS_NOERR.

### See Also
- “Using the C Grid API Functions” on page 994
- “C Grid API Structures” on page 1006
- EssGBeginRemoveOnly
EssGBeginKeepOnly

Begins a keep-only operation to isolate cells to keep, removing all others.

Syntax

```c
ESSG_FUNC_M EssGBeginKeepOnly (hGrid, usCells, pKeepCells, ulOptions);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hGrid</td>
<td>ESSG_HGRID_T</td>
<td>Handle passed back from EssGNewGrid.</td>
</tr>
<tr>
<td>usCells</td>
<td>ESSG_USHORT_T</td>
<td>A count of the number of cell ranges in pKeepCells (the size of array).</td>
</tr>
<tr>
<td>pKeepCells</td>
<td>“ESSG_RANGE_T” on</td>
<td>Describes the cells to be kept. The members to be kept applies only to one</td>
</tr>
<tr>
<td></td>
<td>page 1012</td>
<td>dimension. That is, if the user decides to keep, for example, &quot;Qtr1&quot;, then</td>
</tr>
<tr>
<td></td>
<td></td>
<td>all other members of the Time dimension will be removed and the only</td>
</tr>
<tr>
<td></td>
<td></td>
<td>representative of the Time dimension will be &quot;Qtr1&quot;. All other dimensions</td>
</tr>
<tr>
<td></td>
<td></td>
<td>in the report will be left untouched. This is a one-dimensional array of</td>
</tr>
<tr>
<td></td>
<td></td>
<td>cell ranges. More than one member from a dimension may be specified. Also,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>multiple dimensions may be specified.</td>
</tr>
<tr>
<td>ulOptions</td>
<td>ESSG_ULONG_T</td>
<td>Reserved for future use. Should be set to zero.</td>
</tr>
</tbody>
</table>

Notes

The cells to be kept are described by a one-dimensional array of cell ranges. Items to be kept apply on a per dimension basis.

Return Value

If successful, returns ESSG_STS_NOERR.

Access

None.

Example

```c
ESSG_VOID_T ESSG_BeginKeepOnly (ESSG_HGRID_T hGrid)
{
    ESSG_FUNC_M    sts = ESS_STS_NOERR;
    ESSG_PPDATA_T  ppDataIn;
    ESSG_PPDATA_T  ppDataOut;
    ESSG_RANGE_T   rDataRangeIn, rDataRangeOut;
    ESSG_ULONG_T   ulOptions;
    ESSG_USHORT_T  usCells;
    ESSG_RANGE_T   pKeepCells;
    ESSG_USHORT_T  usState;
```
/* connect the grid to a database on the server */
sts = EssGConnect(hGrid, "Rainbow", "Admin", "Password", "Demo", "Basic", ESSG_CONNECT_DEFAULT);

if(sts == 0)
{
    ppDataIn = BuildTable(&rDataRangeIn);
    pKeepCells.ulRowStart = 1;
    pKeepCells.ulColumnStart = 0;
    pKeepCells.ulNumRows = 1;
    pKeepCells.ulNumColumns = 1;
    ulOptions = 0;
    usCells = 1;

    /* start the keep-only operation */
    sts = EssGBeginKeepOnly(hGrid, usCells, &pKeepCells, ulOptions);
}

if(sts == 0)
{
    /* send the entire grid to define the query */
    sts = EssGSendRows(hGrid, &rDataRangeIn, ppDataIn);
}

if(sts == 0)
{
    /* perform the keep-only operation */
    sts = EssGPerformOperation(hGrid, 0);

    /* free the built data */
    FreeTwoDim(ppDataIn, rDataRangeIn.ulNumRows);
}

if (sts == 0)
{
    /* determine the results of the keep-only operation */
    sts = EssGGetResults(hGrid, 0, &rDataRangeOut, &usState);
}

if(sts ==0)
{
    /* get all the data */
    sts = EssGGetRows(hGrid, 0, &rDataRangeOut, &rDataRangeOut, &ppDataOut);
}

if(sts == 0)
{
    DisplayOutput(ppDataOut, rDataRangeOut);
    /* free the returned data */
    EssGFreeRows(hGrid, &rDataRangeOut, ppDataOut);
}

if(!sts)
See Also

- “Using the C Grid API Functions” on page 994
- “C Grid API Structures” on page 1006

## EssGBeginLock

Locks blocks at the database.

### Syntax

```c
ESSG_FUNC_M EssGBeginLock (hGrid, ulOptions);
```

### Parameter | Data Type     | Description
---|---------------|--------------------------------------------------
| hGrid     | ESSG_HGRID_T | Handle passed back from EssGNewGrid.            
| ulOptions | ESSG_ULONG_T | Reserved for future use. Should be set to zero. |

### Notes

- This function is functionally identical to calling the `EssGRetrieve` function using `ESSG_RET_LOCKONLY` for the `ulOptions` parameter.
- Returns no data to the caller.
- You do not need to retrieve any rows for this operation. It is sufficient to call `EssGSendRows` and `EssGPerformOperation`.

### Return Value

If successful, returns `ESSG_STS_NOERR`.

### Access

None.

### Example

```c
ESSG_VOID_T ESSG_BeginLock (ESSG_HGRID_T hGrid)
{
    ESSG_FUNC_M   sts = ESS_STS_NOERR;
    ESSG_PPDATA_T ppDataIn;
    ESSG_RANGE_T  rDataRangeIn;
    ESSG_ULONG_T  ulOptions;

    /* connect the grid to a database on the server */
    sts = EssGConnect(hGrid, "Rainbow", "Admin",
                      "Password", "Demo", "Basic",
                      ESSG_CONNECT_DEFAULT);
```
if(sts == 0)
{
    ppDataIn = BuildTable(&rDataRangeIn);

    /* start the lock operation */
    ulOptions = 0;
    sts = EssGBeginLock(hGrid, ulOptions);
}

if(sts == 0)
{
    /* send the entire grid to define the query */
    sts = EssGSendRows(hGrid, &rDataRangeIn,
                       ppDataIn);
}

if(sts == 0)
{
    /* perform the lock operation */
    sts = EssGPerformOperation(hGrid, 0);

    /* Free the built data */
    FreeTwoDim(ppDataIn, rDataRangeIn.ulNumRows);
}

if(!sts)
{
    EssGEndOperation(hGrid, 0);
    EssG Disconnect(hGrid, 0);
}

See Also

- “Using the C Grid API Functions” on page 994
- “C Grid API Structures” on page 1006

**EssGBeginPivot**

Begins a pivot.

**Syntax**

```c
ESSG_FUNC_M EssGBeginPivot (hGrid, pStartCell, pEndCell, ulOptions);
```

**Parameter** | **Data Type** | **Description**
---|---|---
`hGrid` | ESSG_HGRID_T | Handle passed back from EssGNewGrid.
`pStartCell` | "ESSG_RANGE_T" on page 1012 | Describes the cell where the pivot is to originate. The member in this cell describes the dimension to be pivoted. This parameter cannot be NULL.
`pEndCell` | "ESSG_RANGE_T" on page 1012 | Describes the cell where the dimension is to be placed. A NULL value for this parameter indicates a pivot from Row to Column, or Column to Row for the dimension members.
### Parameter | Data Type | Description
--- | --- | ---
ulOptions | ESSG_ULONG_T | Reserved for future use. Should be set to zero.

### Notes
The caller supplies the starting cell and the destination cell for the pivot.

### Return Value
If successful, returns **ESSG_STS_NOERR**.

### Access
None.

### Example

```c
ESSG_VOID_T ESSG_BeginPivot (ESSG_HGRID_T hGrid)
{
    ESSG_FUNC_M    sts = ESS_STS_NOERR;
    ESSG_PPDATA_T        ppDataIn;
    ESSG_PPDATA_T        ppDataOut;
    ESSG_RANGE_T         rDataRangeIn, rDataRangeOut;
    ESSG_ULONG_T         ulOptions;
    ESSG_RANGE_T         pStartCell;
    ESSG_RANGE_T         pEndCell;
    ESSG_USHORT_T        usState;

    /* connect the grid to a database on the server */
    sts = EssGConnect(hGrid, "Rainbow", "Admin",
        "Password", "Demo", "Basic",
        ESSG_CONNECT_DEFAULT);

    if(sts == 0)
    {
        ppDataIn = BuildTable(&rDataRangeIn);

        pStartCell.ulRowStart = 0;
        pStartCell.ulColumnStart = 3;
        pStartCell.ulNumRows = 1;
        pStartCell.ulNumColumns = 1;

        pEndCell.ulRowStart = 1;
        pEndCell.ulColumnStart = 1;
        pEndCell.ulNumRows = 1;
        pEndCell.ulNumColumns = 1;
        ulOptions = 0;

        /* start the pivot operation */
        sts = EssGBeginPivot(hGrid,&pStartCell, &pEndCell, ulOptions);
    }

    if(sts == 0)
    {
        /* send the entire grid to define the query */
        sts = EssGSendRows(hGrid, &rDataRangeIn ppDataIn);
    }
```
if(sts == 0) {
    /* perform the pivot operation */
    sts = EssGPerformOperation(hGrid, 0);

    /* free the built data */
    FreeTwoDim(ppDataIn, rDataRangeIn.ulNumRows);
}
if(sts == 0) {
    /* determine the results of the pivot operation */
    sts = EssGGetResults(hGrid, 0, &rDataRangeOut, &usState);
}
if(sts ==0) {
    /* get all the data */
    sts = EssGGetRows(hGrid, 0, &rDataRangeOut,
        &rDataRangeOut, &ppDataOut);
}

if(sts == 0) {
    DisplayOutput(ppDataOut, rDataRangeOut);
    /* free the returned data */
    EssGFreeRows(hGrid, &rDataRangeOut, ppDataOut);
}

if(!sts) {
    EssGEndOperation(hGrid, 0);
    EssGDisconnect(hGrid, 0);
}

See Also

- “Using the C Grid API Functions” on page 994
- “C Grid API Structures” on page 1006

**EssGBeginRemoveOnly**

Begin a remove-only operation, isolating the cells to be removed.

**Syntax**

`ESSG_FUNC_M EssGBeginRemoveOnly (hGrid, usCells, pRemoveCells, ulOptions);`

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hGrid</td>
<td>ESSG_HGRID_T</td>
<td>Handle passed back from EssGNewGrid.</td>
</tr>
<tr>
<td>usCells</td>
<td>ESSG_USHORT_T</td>
<td>A count of the number of cell ranges in pRemoveCells (the size of array).</td>
</tr>
<tr>
<td>Parameter</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>pRemoveCells</td>
<td>“ESSG_RANGE_T” on page 1012</td>
<td>Describes the cells to be removed. The members removed applies only to one dimension. That is, if the user decides to remove, for example, “Qtr1”, then all other members of the Time dimension will be kept. All other dimensions in the report will be left untouched. This is a one-dimensional array of cell ranges. More than one member from a dimension may be specified. Also, multiple dimensions may be specified.</td>
</tr>
<tr>
<td>ulOptions</td>
<td>ESSG_ULONG_T</td>
<td>Reserved for future use. Should be set to zero.</td>
</tr>
</tbody>
</table>

**Notes**

The cells to be removed are described by a one-dimensional array of cell ranges. Items to be removed apply on a per dimension basis.

**Return Value**

If successful, returns ESSG_STS_NOERR.

**Example**

```c
ESSG_VOID_T ESSG_BeginRemoveOnly (ESSG_HGRID_T hGrid)
{
    ESSG_STS_T     sts = ESS_STS_NOERR;
    ESSG_PPDATA_T        ppDataIn;
    ESSG_PPDATA_T        ppDataOut;
    ESSG_RANGE_T          rDataRangeIn, rDataRangeOut;
    ESSG_ULONG_T          ulOptions;
    ESSG_USHORT_T        usCells;
    ESSG_RANGE_T          pRemoveCells;
    ESSG_USHORT_T        usState;
    /* connect the grid to a database on the server */
    sts = EssGConnect(hGrid, "Rainbow", "Admin", "Password", "Demo", "Basic", ESSG_CONNECT_DEFAULT);
    if(sts == 0)
    {
        ppDataIn = BuildTable(&rDataRangeIn);
        pRemoveCells.ulRowStart = 1;
        pRemoveCells.ulColumnStart = 0;
        pRemoveCells.ulNumRows = 1;
        pRemoveCells.ulNumColumns = 1;
        ulOptions = 0;
        usCells = 1;
        /* start the remove-only operation */
        sts = EssGBeginRemoveOnly(hGrid, usCells, &pRemoveCells, ulOptions);
    }
    if(sts == 0)
```
/* send the entire grid to define the query */
sts = EssGSendRows(hGrid, &rDataRangeIn,
    ppDataIn);
}

if(sts == 0)
{
    /* perform the remove-only operation */
    sts = EssGPerformOperation(hGrid, 0);

    /* free the built data */
    FreeTwoDim(ppDataIn, rDataRangeIn.ulNumRows);
}
if (sts == 0)
{
    /* determine the results of the remove-only operation */
    sts = EssGGetResults(hGrid, 0, &rDataRangeOut,
        &usState);
}
if(sts ==0)
{
    /* get all the data */
    sts = EssGGetRows(hGrid, 0, &rDataRangeOut,
        &rDataRangeOut, &ppDataOut);
}

if(sts == 0)
{
    DisplayOutput(ppDataOut, rDataRangeOut);
    /* Free the returned data */
    EssGFreeRows(hGrid, &rDataRangeOut, ppDataOut);
}

if(!sts)
{
    EssGEndOperation(hGrid, 0);
    EssGDisconnect(hGrid, 0);
}

See Also

- “Using the C Grid API Functions” on page 994
- “C Grid API Structures” on page 1006
- EssGGetCellLinkResults
- EssGBeginDrillAcross
- EssGBeginDeleteLROs
- EssGBeginDrillOrLink
- EssGFreeCellLinkResults
- EssGGetCellLinkResults
- EssGGetCellLinkResults
**EssGBeginReport**

Runs a report script at the server.

**Syntax**

```
ESSG_FUNC_M EssGBeginReport (hGrid, pszReportIn, ulOptions)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hGrid</td>
<td>ESSG_HGRID_T</td>
<td>Handle passed back from EssGNewGrid.</td>
</tr>
<tr>
<td>pszReportIn</td>
<td>ESSG_STR_T</td>
<td>String (no greater than 64K) containing an Essbase report specification.</td>
</tr>
<tr>
<td>ulOptions</td>
<td>ESSG_ULONG_T</td>
<td>A bitmask which describes returned grid options. Valid values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ESSG_NOATTRIBUTES returns grid without pAttributes values.</td>
</tr>
</tbody>
</table>

**Notes**

- Returns the results as a two-dimensional array of cells.
- You do not need to send any rows for this operation. It is sufficient to call EssGPerformOperation, EssGGetResults, and EssGGetRows.
- Attributes for returned cell values are obtained using a second server request. Passing ESSG_NOATTRIBUTES for the ulOptions parameter will issue one less request of the server, and could, in large resulting grids, be faster.
- Reports passed to the server via the Grid API should be sure to request a tab delimited report format be returned [TABDELIM]. If a non-tab delimited report is returned, the Grid API may be unable to convert the resulting report into a grid.
- If the report specification modifies the string used for #Missing aliases, then Missing cells will be returned as string types (ESSG_DT_STRING) with the new #Missing alias as the text and not as ESSG_DT_MISSING cells.
- Client programs that call EssGBeginReport() and other report functions need to take into account new “C Grid API Structures” on page 1006 and “C Grid API Data Types” on page 1004 (specifically StringEx and MemberEx). Older programs should be revised in order to work with the newer servers.

**Return Value**

If successful, returns ESSG_STS_NOERR.

**Access**

None.

**Example**

```
ESSG_VOID_T ESSG_BeginReport (ESSG_HGRID_T hGrid)
{
    ESSG_FUNC_M    sts = ESS_STS_NOERR;
    ESSG_PPDATA_T        ppDataOut;
    ESSG_RANGE_T         rDataRangeOut;
    ESSG_ULONG_T         ulOptions;
```

EssGBeginReport 1033
ESSG_STR_T pszReportIn;
ESSG_USHORT_T usState;

/* connect the grid to a database on the server */
sts = EssGConnect(hGrid, "Rainbow", "Admin", "Password", "Demo", "Basic",
    ESSG_CONNECT_DEFAULT);
if(sts == 0)
{
    pszReportIn = "{TabDelim}<idesc Year !";
    ulOptions = ESSG_NOATTRIBUTES;
    sts = EssGBeginReport(hGrid, pszReportIn,
        ulOptions);
}
if(sts == 0)
{
    /* perform the report */
    sts = EssGPerformOperation(hGrid, 0);
}
if(sts == 0)
{
    /* determine the results of the report */
    sts = EssGGetResults(hGrid, 0, &rDataRangeOut,
        &usState);
}
if(sts ==0)
{
    /* get all the data */
    sts = EssGGetRows(hGrid, 0, &rDataRangeOut,
        &rDataRangeOut, &ppDataOut);
}
if(sts == 0)
{
    DisplayOutput (ppDataOut, rDataRangeOut);
    /* Free the returned data */
    EssGFreeRows(hGrid, &rDataRangeOut, ppDataOut);
}
if(!sts)
{
    EssGEndOperation(hGrid, 0);
    EssGDisconnect(hGrid, 0);
}

See Also
- “Using the C Grid API Functions” on page 994
- “C Grid API Structures” on page 1006

EssGBeginReportFile

Runs a report file at the server.

Syntax

ESSG_FUNC_M EssGBeginReportFile (hGrid, pszReportName, bLocal, ulOptions);
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hGrid</td>
<td>ESG_HGRID_T</td>
<td>Handle passed back from EssGNewGrid.</td>
</tr>
<tr>
<td>pszReportName</td>
<td>ESG_STR_T</td>
<td>Name of report to run. If this report resides on the server, then it should exist in the APPLICATION\DATABASE directory. If this report resides locally, then this string contains the absolute path name of the report.</td>
</tr>
<tr>
<td>bLocal</td>
<td>ESG_BOOL_T</td>
<td>Boolean indicating whether the report exists locally or not. A TRUE value indicates the report exists locally while a FALSE value indicates the report exists on the server.</td>
</tr>
<tr>
<td>ulOptions</td>
<td>ESG_ULONG_T</td>
<td>A bitmask which describes returned grid options. Valid values are: ESG_NOATTRIBUTES returns grid without pAttributes values</td>
</tr>
</tbody>
</table>

**Notes**

- Returns the results as a two-dimensional array of cells.
- You do not need to send any rows for this operation. It is sufficient to call EssGPerformOperation, EssGGetResults, and EssGGetRows.
- Attributes for returned cell values are obtained using a second server request. Passing ESG_NOATTRIBUTES for the ulOptions parameter will issue one less request of the server, and could, in large resulting grids, be faster.
- Reports passed to the server via the Grid API should be sure to request a tab delimited report format be returned [TABDELIM]. If a non-tab delimited report is returned, the Grid API may be unable to convert the resulting report into a grid.
- If the report spec modifies the string used for #Missing aliases, then Missing cells will be returned as string types (ESSG_DT_STRING) with the new #Missing alias as the text and not as ESG_DT_MISSING cells.
- For non-local (Server-based) report file objects, no file extension should be used in the pszReportName parameter.

**Return Value**

If successful, returns ESG_STS_NOERR.

**Access**

None.

**Example**

```
ESSG_VOID_T ESSG_BeginReportFile (ESSG_HGRID_T hGrid)
{
    ESSG_FUNC_M     sts = ESS_STS_NOERR;
    ESSG_PPDATA_T         pDataOut;
    ESSG_RANGE_T          rDataRangeOut;
    ESSG ULONG_T          ulOptions;
    ESSG_STR_T            pszReportName;
    ESSG_BOOL_T           bLocal;
    ESSG_USHORT_T         usState;
```

/* connect the grid to a database on the server */
sts = EssGConnect(hGrid, "Rainbow", "Admin",
    "Password", "Demo", "Basic",
    ESSG_CONNECT_DEFAULT);

if (sts == 0)
{
    pszReportName = "DescYear";
bLocal = ESSG_FALSE;
ulOptions = ESSG_NOATTRIBUTES;

    /* start the report file operation */
    sts = EssGBeginReportFile(hGrid,
        pszReportName, bLocal, ulOptions);
}

if (sts == 0)
{
    /* perform the report operation */
    sts = EssGPerformOperation(hGrid, 0);
}

if (sts == 0)
{
    /* determine the results of the report operation */
    sts = EssGGetResults(hGrid, 0, &rDataRangeOut, &usState);
}

if (sts == 0)
{
    /* get all the data */
    sts = EssGGetRows(hGrid, 0, &rDataRangeOut,
        &rDataRangeOut, &ppDataOut);
}

if (sts == 0)
{
    DisplayOutput(ppDataOut, rDataRangeOut);
    /* Free the returned data */
    EssGFreeRows(hGrid, &rDataRangeOut, ppDataOut);
}

if (!sts)
{
    EssGEndOperation(hGrid, 0);
    EssGDisconnect(hGrid, 0);
}

See Also

- “Using the C Grid API Functions” on page 994
- “C Grid API Structures” on page 1006
EssGBeginRetrieve

Begins the basic retrieval operation.

Syntax

```c
ESSG_FUNC_M EssGBeginRetrieve (hGrid, ulOptions);
```

Parameter  | Data Type  | Description
---------- | ---------- | ------------
hGrid     | ESSG_HGRID_T  | Handle passed back from EssGNewGrid.
ulOptions | ESSG_ULONG_T  | A constant which describes the type of retrieval. One of the following values must be used:

- ESSG_RET_RETRIEVE Retrieve Only
- ESSG_RET_RETRIEVELOCK Retrieve and Lock
- ESSG_RET_LOCKONLY Lock Only (No data is to be retrieved)

Notes

- Optionally locks blocks at the server for later update as the rows are passed in via EssGSendRows.
- You can do a retrieval without sending any rows in order to get a default grid with the only the dimension names used as members.

Return Value

If successful, returns ESSG_STS_NOERR.

Access

None.

Example

```c
#include <essapin.h>
#include <essgapin.h>

ESSG_VOID_T ESSG_BeginRetrieve(ESSG_HGRID_T hGrid)
{
    ESSG_FUNC_M     sts = ESS_STS_NOERR;
    ESSG_PPDATA_T   pDataIn;
    ESSG_PPDATA_T   ppDataOut;
    ESSG_RANGE_T    rDataRangeIn, rDataRangeOut;
    ESSG_ULONG_T    ulOptions;
    ESSG_USHORT_T   usState;

    /* connect the grid to a database on the server */
    sts = EssGConnect(hGrid, "Rainbow", "Admin",
                      "Password", "Demo", "Basic", ESSG_CONNECT_NODIALOG);

    if(sts == 0)
    {
        ppDataIn = BuildTable(&rDataRangeIn);
        ulOptions = ESSG_RET_RETRIEVE;
        /* start the retrieve operation */
```
sts = EssGBeginRetrieve(hGrid, ulOptions);
}

if(sts == 0)
{
    /* send the entire grid to define the query */
    sts = EssGSendRows(hGrid, &rDataRangeIn, ppDataIn);
}

if(sts == 0)
{
    /* perform the retrieval */
    sts = EssGPerformOperation(hGrid, 0);

    /* free the built data */
    FreeTwoDim(ppDataIn, rDataRangeIn.ulNumRows);
}

if(sts == 0)
{
    /* determine the results of the retrieve */
    sts = EssGGetResults(hGrid, 0, &rDataRangeOut, &usState);
}

if(!sts && usState == ESSG_STATE_DONE)
{
    /* get all the data */
    sts = EssGGetRows(hGrid, 0, &rDataRangeOut, &rDataRangeOut,
                       &ppDataOut);
}

if(sts == 0)
{
    DisplayOutput (ppDataOut, rDataRangeOut);
    /* free the returned data */
    EssGFreeRows(hGrid, &rDataRangeOut, ppDataOut);
}

if(!sts)
{
    EssGEndOperation(hGrid, 0);
    EssGDisconnect(hGrid, 0);
}

See Also

- “Using the C Grid API Functions” on page 994
- “C Grid API Structures” on page 1006

EssGBeginSamplingZoomIn

begins a random sampled zoom-in operation.
Syntax

ESSG_FUNC_M EssGBeginSamplingZoomIn (hGrid, usCells, pZoomCells, ulSamplingPercentage, ulOptions);

Parameter | Data Type | Description
--- | --- | ---
hGrid | ESSG_HGRID_T | Handle passed back from EssGNewGrid.
usCells | ESSG_USHORT_T | A count of the number of cell ranges in pZoomCells (the size of array).
pZoomCells | “ESSG_RANGE_T” on page 1012 | Describes the cells to be zoomed in upon. This is a one-dimensional array of cell ranges.
ulSamplingPercentage | ESSG_ULONG_T | The percentage sampling rate. This number is an integer between 1 and 100, inclusive. A depth of 100 percent will retrieve all members of the dimension. This effectively turns sampling off and retrieves all members. A ulSamplingPercentage of 50 will retrieve half the members.
ulOptions | ESSG_ULONG_T | A bitmask which describes the type of zoom-in (across or down) and the level of the zoom. The following two values are mutually exclusive:

- ESSG_ZOOM_DOWN Any page/title dimensions selected will be zoomed down
- ESSG_ZOOM_ACROSS Any page dimensions selected will be zoomed across

The following level values for ulOptions are themselves mutually exclusive:

- ESSG_NEXTLEVEL Children
- ESSG_ALLLEVELS All members
- ESSG_BOTTOMLEVEL Bottom level
- ESSG_SIBLEVEL Sibling level
- ESSG_SAMELEVEL Same level
- ESSG_SAMEGENERATION Same generation
- ESSG_CALCLEVEL Calculation
- ESSG_OPTIONS Use setting for grid options

Use bitwise OR (|) to specify the ulOptions; for example, ESSG_ZOOM_DOWN | ESSG_NEXTLEVEL

Notes

- The cells to be zoomed in upon are described by a one-dimensional array of cell ranges.
- This function differs from the standard grid Zoom-In function, EssGBeginZoomIn(). This function has an argument that sets the sampling depth in terms of a percentage. A depth of 100 percent will retrieve all members of the dimension and a depth of 50 percent will retrieve half the members. This function is especially useful for zooming in on large or very dense dimensions.

Return Value

If successful, returns ESSG_STS_NOERR.
EssG_BeginUpdate

Begins an update of data at the server. This function returns no data to the caller.

Syntax

\[
\text{ESSG} \_ \text{FUNC} \_ \text{M} \quad \text{EssGBeginUpdate} \quad (h\text{Grid}, \ ul\text{Options});
\]

Parameter | Data Type | Description
--- | --- | ---
\(h\text{Grid}\) | ESSG\_HGRID\_T | Handle passed back from EssGNewGrid.
\(\text{ulOptions}\) | ESSG\_ULONG\_T | A constant which indicates whether the blocks must be previously locked or not prior to update. One of the following mutually exclusive values must be used:

- ESSG\_REQUIRELOCK If the blocks haven’t been previously locked, disallow the update.
- ESSG\_LOCKIFNEEDED If the blocks haven’t been previously locked, lock them and allow the update.

Notes

The blocks are unlocked after the operation is complete, when you have called EssGPerformOperation. If you want the blocks to remain locked, set the Update Mode option to TRUE in EssGSetGridOptions. You do not need to retrieve any rows for this operation; it is sufficient to call EssGSendRows and EssGPerformOperation.

Return Value

If successful, returns ESSG\_STS\_NOERR.

Access

None.

Example

\[
\text{ESSG} \_ \text{VOID} \_ \text{T} \quad \text{EssGBeginUpdate} \quad (\text{ESSG} \_ \text{HGRID} \_ \text{T} \ h\text{Grid})
\]

\{
    \text{ESSG} \_ \text{FUNC} \_ \text{M} \quad \text{sts} = \text{ESS} \_ \text{STS}\_\text{NOERR};
    \text{ESSG} \_ \text{PPDATA} \_ \text{T} \quad \text{ppDataIn};
    \text{ESSG} \_ \text{RANGE} \_ \text{T} \quad \text{rDataRangeIn};
    \text{ESSG} \_ \text{ULONG} \_ \text{T} \quad \text{ulOptions};
\}
/ * connect the grid to a database on the server */
sts = EssGConnect(hGrid, "Rainbow", "Admin",
                "Password", "Demo", "Basic", ESSG_CONNECT_NODIALOG);

if(sts == 0)
{
    ppDataIn = BuildTable (&rDataRangeIn);

    ulOptions = ESSG_LOCKIFNEEDED;
    /* start the update operation */
    sts = EssGBeginUpdate(hGrid, ulOptions);
}

if(sts == 0)
{
    /* send the entire grid to define the query */
    sts = EssGSendRows(hGrid, &rDataRangeIn, ppDataIn);
}

if(sts == 0)
{
    /* perform the update */
    sts = EssGPerformOperation(hGrid, 0);

    /* free the built data */
    FreeTwoDim(ppDataIn, rDataRangeIn.ulNumRows);
}

if(!sts)
{
    EssGEndOperation(hGrid, 0);
    EssGDisconnect(hGrid, 0);
}

See Also

- "Using the C Grid API Functions" on page 994
- "C Grid API Structures" on page 1006

### EssGBeginZoomIn

Begins a zoom-in.

**Syntax**

ESSL_FUNC_M EssGBeginZoomIn (hGrid, usCells, pZoomCells, ulOptions);

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hGrid</td>
<td>ESSG_HGRID_T</td>
<td>Handle passed back from EssGNewGrid.</td>
</tr>
<tr>
<td>usCells</td>
<td>ESSG_USHORT_T</td>
<td>A count of the number of cell ranges in pZoomCells (the size of array).</td>
</tr>
</tbody>
</table>
Parameter | Data Type | Description
--- | --- | ---
pZoomCells | “ESSG_RANGE_T” on page 1012 | Describes the cells to be zoomed in upon. This is a one-dimensional array of cell ranges.
ulOptions | ESSG_ULONG_T | A bitmask which describes the type of zoom-in (across or down) and the level of the zoom. The following two values are mutually exclusive:
- ESSG_ZOOM_DOWN—Any page/title dimensions selected will be zoomed down
- ESSG_ZOOM_ACROSS—Any page dimensions selected will be zoomed across

The following level values for ulOptions are themselves mutually exclusive:
- ESSG_NEXTLEVEL—Children
- ESSG_ALLLEVELS—All members
- ESSG_BOTTOMLEVEL—Bottom level
- ESSG_SIBLEVEL—Sibling level
- ESSG_SAMELEVEL—Same level
- ESSG_SAMEGENERATION—Same generation
- ESSG_CALCLEVEL—Calculation
- ESSG_OPTIONS—Use setting for grid options

Use bitwise OR (|) to specify the ulOptions; for example, ESSG_ZOOM_DOWN | ESSG_NEXTLEVEL

Notes
The cells to be zoomed in upon are described by a one-dimensional array of cell ranges.

Return Value
If successful, returns ESSG_STS_NOERR.

Access
None.

Example
ESSG_VOID_T ESSG_BeginZoomIn (ESSG_HGRID_T hGrid)
{
    ESSG_FUNC_M    sts = ESS_STS_NOERR;
    ESSG_PPDATA_T        ppDataIn;
    ESSG_PPDATA_T        ppDataOut;
    ESSG_RANGE_T         rDataRangeIn, rDataRangeOut;
    ESSG_ULONG_T         ulOptions;
    ESSG_USHORT_T        usCells;
    ESSG_RANGE_T         pZoomCells;
    ESSG_USHORT_T        usState;

    /* connect the grid to a database on the server */
    sts = EssGConnect(hGrid, "Rainbow", "Admin",
                      "Password", "Demo", "Basic", ESSG_CONNECT_DEFAULT);
if (sts == 0) {
  ppDataIn = BuildTable(&rDataRangeIn);
  ulOptions = ESSG_ZOOM_DOWN | ESSG_ALLLEVELS;
  pZoomCells.ulRowStart = 0;
  pZoomCells.ulColumnStart = 2;
  pZoomCells.ulNumRows = 1;
  pZoomCells.ulNumColumns = 1;
  usCells = 1;
  /* start the zoom in operation */
  sts = EssGBeginZoomIn(hGrid, usCells, &pZoomCells, ulOptions);
}
if (sts == 0) {
  /* send the entire grid to define the query */
  sts = EssGSendRows(hGrid, &rDataRangeIn, ppDataIn);
}
if (sts == 0) {
  /* perform the zoom-in */
  sts = EssGPerformOperation(hGrid, 0);
  /* Free the built data */
  FreeTwoDim(ppDataIn, rDataRangeIn.ulNumRows);
}
if (sts == 0) {
  /* determine the results of the zoom-in */
  sts = EssGGetResults(hGrid, 0, &rDataRangeOut, &usState);
}
if (sts == 0) {
  /* get all the data */
  sts = EssGGetRows(hGrid, 0, &rDataRangeOut, &rDataRangeOut, &ppDataOut);
}
if (sts == 0) {
  DisplayOutput(ppDataOut, rDataRangeOut);
  /* Free the returned data */
  EssGFreeRows(hGrid, &rDataRangeOut, ppDataOut);
}
if (!sts) {
  EssGEndOperation(hGrid, 0);
  EssGDisconnect(hGrid, 0);
EssGBeginZoomOut

Begins a zoom-out.

**Syntax**

```c
ESSG_FUNC_M EssGBeginZoomOut (hGrid, usCells, pZoomCells, ulOptions);
```

**Parameter** | **Data Type** | **Description**
--- | --- | ---
``hGrid`` | ESSG_HGRID_T | Handle passed back from EssGNewGrid.
``usCells`` | ESSG_USHORT_T | A count of the number of cell ranges in ``pZoomCells`` (the size of array).
``pZoomCells`` | “ESSG_RANGE_T” on page 1012 | Describes the cells to be zoomed out upon. This is a one-dimensional array of cell ranges.
``ulOptions`` | ESSG_ULONG_T | Reserved for future use. Should be set to zero.

**Return Value**

If successful, returns ESSG_STS_NOERR.

**Access**

None.

**Example**

```c
ESSG_VOID_T ESSG_BeginZoomOut (ESSG_HGRID_T hGrid)
{
    ESSG_FUNC_M     sts = ESS_STS_NOERR;
    ESSG_PPDATA_T   ppDataIn;
    ESSG_PPDATA_T   ppDataOut;
    ESSG_RANGE_T    rDataRangeIn, rDataRangeOut;
    ESSGULONG_T     ulOptions;
    ESSGUSHORT_T    usCells;
    ESSG_RANGE_T    pZoomCells;
    ESSGUSHORT_T    usState;

    /* connect the grid to a database on the server */
    sts = EssGConnect(hGrid, "Rainbow", "Admin",
                      "Password", "Demo", "Basic",
                      ESSG_CONNECT_DEFAULT);

    if(sts == 0)
    {
        ppDataIn = BuildTable(&rDataRangeIn);
```
pZoomCells.ulRowStart = 1;
pZoomCells.ulColumnStart = 1;
pZoomCells.ulNumRows = 1;
pZoomCells.ulNumColumns = 1;
ulOptions = 0;
usCells = 1;

/* start the zoom out operation */
sts = EssGBeginZoomOut(hGrid, usCells, 
                        &pZoomCells, ulOptions);
}

if(sts == 0)
{
    /* send the entire grid to define the query */
    sts = EssGSendRows(hGrid, &rDataRangeIn,
                        ppDataIn);
}

if(sts == 0)
{
    /* perform the zoom-out */
    sts = EssGPerformOperation(hGrid, 0);

    /* Free the built data */
    FreeTwoDim(ppDataIn, rDataRangeIn.ulNumRows);
}
if (sts == 0)
{
    /* determine the results of the zoom-out */
    sts = EssGGetResults(hGrid, 0, &rDataRangeOut,
                         &usState);
}
if(sts ==0)
{
    /* get all the data */
    sts = EssGGetRows(hGrid, 0, &rDataRangeOut,
                      &rDataRangeOut, &ppDataOut);
}

if(sts == 0)
{
    DisplayOutput(ppDataOut, rDataRangeOut);
    /* free the returned data */
    EssGFreeRows(hGrid, &rDataRangeOut, ppDataOut);
}

if(!sts)
{
    EssGEndOperation(hGrid, 0);
    EssGDisconnect(hGrid, 0);
}
}

See Also

- “Using the C Grid API Functions” on page 994
EssGCancelOperation

Cancels an operation at any stage during an operation.

**Syntax**

```c
ESSG_FUNC_M EssGCancelOperation (hGrid, ulOptions);
```

**Parameter**  |  **Data Type**  |  **Description**  
---|---|---
`hGrid`  |  `ESSG_HGRID_T`  |  Handle passed back from `EssGNewGrid`.  
`ulOptions`  |  `ESSG_ULONG_T`  |  Reserved for future use. Should be set to zero.

**Notes**

- You can make this call at any time after `EssGBeginXxx` has been called.
- The current operation is cancelled, and all resources are freed.

**Return Value**

If successful, returns `ESSG_STS_NOERR`.

**Access**

None.

**Example**

```c
ESSG_VOID_T EssGCancelOperation (ESSG_HGRID_T hGrid)
{
    ESSG_FUNC_M     sts = ESS_STS_NOERR;
    ESSG_ULONG_T          ulOptions;
    ESSG_STR_T            pszReportIn;

    /* connect the grid to a database on the server */
    sts = EssGConnect(hGrid, "Rainbow", "Admin",
                        "Password", "Demo", "Basic",
                        ESSG_CONNECT_DEFAULT);
    if(sts == 0)
    {
        pszReportIn = "{TabDelim}<idesc Year !";
        ulOptions = ESSG_NOATTRIBUTES;
        sts = EssGBeginReport(hGrid, pszReportIn,
                              ulOptions);
    }
    if(sts == 0)
    {
        ulOptions = 0;
        sts = EssGCancelOperation(hGrid, ulOptions);
    }
}
```
if(!sts)
{
    EssGDisconnect(hGrid, 0);
}

See Also

- “Using the C Grid API Functions” on page 994
- “C Grid API Structures” on page 1006

EssGCell

Retrieves from the server a singular value representing a solitary datapoint.

Syntax

\[
\text{ESSG\_FUNC\_M EssGCell (hGrid, usCount, pszMbrs, pDataCell);}
\]

Parameter | Data Type | Description
--- | --- | ---
\(hGrid\) | ESSG\_HGRID\_T | Handle passed back from EssGNewGrid.
\(usCount\) | ESSG\_USHORT\_T | Number of members being sent in. The maximum number of dimensions that EssGCell can report is 20.
\(pszMbrs\) | ESSG\_PSTR\_T | Array of member names to query. No more than one representative per dimension is allowed.
\(pDataCell\) | “ESSG\_DATA\_T” on page 1006 | Value returned by server.

Notes

- You can specify a maximum of:
  - 20 members.
  - One member per dimension.
- If you do not specify a member for a dimension, the top level (dimension) member is used as the default.

Return Value

If successful, returns ESSG\_STS\_NOERR.

Access

None.

Example

\[
\text{ESSG\_VOID\_T EssGCell (ESSG\_HGRID\_T hGrid)}
\]

\{
    ESSG\_FUNC\_M \hspace{1em} \text{sts} = \text{ESS\_STS\_NOERR};
\}
ESSG_USHORT_T        usCount;
ESSG_DATA_T          DataCell;
    "West", "Audio",
    "Sales"};

/* connect the grid to a database on the server */
sts = EssGConnect(hGrid, "Rainbow", "Admin",
    "Password", "Demo", "Basic",
    ESSG_CONNECT_NODIALOG);

/* retrieve cell value */
usCount = 5;
if(sts == 0)
    sts = EssGCell(hGrid, usCount, pszMbrs,&DataCell);

if(!sts)
{
    switch(DataCell.usType)
    {
    case(ESSG_DT_STRING):
        printf("%s", DataCell.Value.pszStr+1);
        break;
    case(ESSG_DT_LONG):
        printf("%ld", DataCell.Value.lData);
        break;
    case(ESSG_DT_DOUBLE):
        printf("%g", DataCell.Value.dblData);
        break;
    case(ESSG_DT_BLANK):
        break;
    case(ESSG_DT_RESERVED):
        printf("#Reserved");
        break;
    case(ESSG_DT_ERROR):
        printf("#Error");
        break;
    case(ESSG_DT_MISSING):
        printf("#Missing");
        break;
    case(ESSG_DT_ZERO):
        printf("%ld", DataCell.Value.lData);
        break;
    case(ESSG_DT_NOACCESS):
        printf("#NoAccess");
        break;
    case(ESSG_DT_MEMBER):
        printf("%s", DataCell.Value.pszStr+1);
        break;
    default:
        break;
    }
}
if(!sts)
    EssGDisconnect(hGrid, 0);
EssGCreateMemberwKeyStr

Creates a combined string using the member name and member key as input. A key is a value generated by Essbase that uniquely identifies a member name in the outline.

Syntax

```
ESSG_FUNC_M EssGCreateMemberwKeyStr (pszMember, pszKey, *pszOutStr);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pszMember</td>
<td>ESSG_STR_T</td>
<td>Member name (input).</td>
</tr>
<tr>
<td>pszKey</td>
<td>ESSG_STR_T</td>
<td>Member key (input).</td>
</tr>
<tr>
<td>*pszOutStr</td>
<td>ESSG_STR_T</td>
<td>Output string of the format:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;member-name length&gt;&lt;member-name&gt;&lt;key length &gt;&lt;key&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>where the length elements are 2 bytes in size. Note that &lt;member-name&gt; is null-terminated.</td>
</tr>
</tbody>
</table>

Notes

You must free the string *pszOutStr using EssGFreeMemberwKeyStr.

Example

```
ESSG_VOID_T ESSG_BeginZoomIn(ESSG_HGRID_T hGrid)
{
    ESSG_STS_T     sts = ESS_STS_NOERR;
    ESSG_DATA_T   **ppDataIn;
    ESSG_DATA_T   **ppDataOut;
    ESSG_RANGE_T  rDataRangeIn, rDataRangeOut;
    ESSG_ULONG_T   ulOptions;
    ESSG_USHORT_T  usCells;
    ESSG_RANGE_T   pZoomCells;
    ESSG_USHORT_T  usState;
    ESSG_USHORT_T  usMember2Len, usKey2Len;
    ESSG_SHORT_T   sOption, sOptionGet;
    ESSG_SHORT_T   tmpShort, tmpShortGet, i;
    ESSG_PVOID_T   pOption, pOptionGet;
    ESSG_STR_T     pMember, pKey, pOutStr;
    ESSG_STR_T     pMember2, pKey2;

    /* connect the grid to a database on the server */
    sts = EssGConnect(hGrid, server, "essexer", pwd, app, db, ESSG_CONNECT_NODIALOG);

    /* set grid option*/
    tmpShort = ESSG_TRUE;
    sOption = ESSG_OP_MEMBERANDUNIQUENAME ;
```
pOption = (ESSG_PVOID_T)tmpShort;     // pOption holds the actual value not a pointer

sts = EssGSetGridOption(hGrid, sOption, pOption);
printf("EssGSetGridOption  sts  %ld\n",sts);

sOptionGet = ESSG_OP_MEMBERANDUNIQUENAME ;  
pOptionGet = &tmpShortGet;
if(!sts)
{
    sts = EssGGetGridOption(hGrid, sOptionGet, pOptionGet);
    printf("EssGGetGridOption  sts  %ld\n",sts);
    printf("EssGSetGridOption set ESSG_OP_MEMBERANDUNIQUENAME TO %d\n", (int)tmpShortGet);
}

if(sts == 0)
{
    ppDataIn = BuildTable(&rDataRangeIn);

    ulOptions = ESSG_ZOOM_DOWN | ESSG_NEXTLEVEL;

    pZoomCells.ulRowStart = 0;
    pZoomCells.ulColumnStart = 2;
    pZoomCells.ulNumRows = 1;
    pZoomCells.ulNumColumns = 1;
    usCells = 1;

    /* start the zoom in operation */
    sts = EssGBeginZoomIn(hGrid, usCells, &pZoomCells, ulOptions);
    printf("EssGBeginZoomIn  sts:  %ld\n",sts);
}

//Display Input
DisplayOutput(ppDataIn, rDataRangeIn);
printf("\n\n");
if(sts == 0)
/* send the entire grid to define the query */
sts = EssGSendRows(hGrid, &rDataRangeIn, ppDataIn);

if(sts == 0)
{
    /* perform the zoom-in */
    sts = EssGPerformOperation(hGrid, 0);

    /* Free the built data */
    FreeTwoDim(ppDataIn, rDataRangeIn.ulNumRows);
}
if (sts == 0)
{
    /* determine the results of the zoom-in */
    sts = EssGGetResults(hGrid, 0, &rDataRangeOut, &usState);
}
if(sts ==0)
{
    /* get all the data */
    sts = EssGGetRows(hGrid, 0, &rDataRangeOut, &rDataRangeOut, &ppDataOut);
if(sts == 0)
{
    DisplayOutput(ppDataOut, rDataRangeOut);

    /* Retreive member and key from cell */
    sts = EssGGetFromMemberwKey (((ppDataOut[1][0]).Value).pszStr, &pMember, &pKey);
    printf("After EssGGetFromMemberwKey\n Member: %s, Key: %s \n\n",
           pMember+2,
           pKey+2);

    //Member is "Qtr1", Key is "[2004].[Qtr1]", pOutStr is in the format
    //nn<member-name>nn<'key> - where nn is string length
    usMember2Len = strlen("Qtr1");
    pMember2 = malloc(usMember2Len+3);
        memset(pMember2, 0, usMember2Len+3);
    usKey2Len = strlen("[2004].[Qtr1]");
    pKey2 = malloc(usKey2Len+3);
        memset(pKey2, 0, usKey2Len+3);
    memcpy(pMember2, &usMember2Len, 2);
    memcpy(pMember2+2, "Qtr1", usMember2Len);
    memcpy(pKey2, &usKey2Len, 2);
    memcpy(pKey2+2, "[2004].[Qtr1]", usKey2Len);
    sts = EssGCreateMemberwKeyStr(pMember2, pKey2, &pOutStr);

    /*Note: because not all elements in pOutStr are actual characters,
    e.g. the 2 bytes for the size of Member and size of Key, plus the
    \0 ending characters, the printf below does not display the actual
    contents of the array */
    for (i=0; i < usMember2Len + usKey2Len + 4 + 2; ++i)
        printf("%c", pOutStr[i]);

    /* Free the returned data */
    EssGFreeRows(hGrid, &rDataRangeOut, ppDataOut);
    sts = EssGFreeMemberwKeyStr(pOutStr);
}

if( sts == 0)
{
    EssGEndOperation(hGrid, 0);
    EssGDisconnect(hGrid, 0);
}

See Also

- EssGFreeMemberwKeyStr
- EssGGetFromMemberwKey
EssGConnect

Connects a grid to an Essbase database.

**Syntax**

ESSG_FUNC_M EssGConnect (hGrid, Server, Username, Password, Application, Database, ulOptions);

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hGrid</td>
<td>ESSG_HGRID_T</td>
<td>Handle passed back from EssGNewGrid.</td>
</tr>
<tr>
<td>Server</td>
<td>ESSG_SERVER_T</td>
<td>Network server name string. The server name can be expressed as hostname or hostname:port.</td>
</tr>
<tr>
<td>Username</td>
<td>ESSG_USERNAME_T</td>
<td>Name of valid user at server.</td>
</tr>
<tr>
<td>Password</td>
<td>ESSG_PASSWORD_T</td>
<td>Password of user.</td>
</tr>
<tr>
<td>Application</td>
<td>ESSG_APPLICATION_T</td>
<td>Name of a valid application on server.</td>
</tr>
<tr>
<td>Database</td>
<td>ESSG_DATABASE_T</td>
<td>Name of a valid database for application on server.</td>
</tr>
<tr>
<td>ulOptions</td>
<td>ESSG_ULONG_T</td>
<td>Options flag. Values are ESSG_CONNECT_NODIALOG, which attempts to login and connect without displaying dialog, using the default/passed setting; or ESSG_CONNECT_DEFAULT which will display the login and selection dialog.</td>
</tr>
</tbody>
</table>

**Notes**

- Calls EssAutoLogin, therefore all rules that apply to EssAutoLogin apply to this function. For example, none of the parameters are case-sensitive.
- If ulOptions is set to ESSG_CONNECT_NODIALOG, none of the connection related parameters can be NULL or empty. When ulOptions is set to ESSG_CONNECT_DEFAULT, and a buffer is passed for the connect parameters, the user's selections from the dialog will be returned in these buffers.
- All security information is utilized. Therefore, if the user connects to a database to which he or she does not have read access, all read operations will fail.

**Return Value**

If successful, returns ESSG_STS_NOERR.

**Access**

None.

**Example**

```c
#include <essapin.h>
#include <essgapin.h>

ESSG_FUNC_M         sts = ESS_STS_NOERR;
ESSG_INIT_T         InitStruct;
ESSG_HANDLE_T       Handle;
```
ESSG_SERVER_T Server;
ESSG_USERNAME_T UserName;
ESSG_PASSWORD_T Password;
ESSG_APPLICATION_T Application;
ESSG_DATABASE_T Database;
ESSG_ULONG_T ulOptions;
ESSG_HGRID_T hGrid;

InitStruct.ulVersion = ESSG_VERSION;
InitStruct.ulMaxRows = 1000;
InitStruct.ulMaxColumns = 200;
InitStruct.pfnMessageFunc = ESS_NULL;
InitStruct.pUserdata = ESS_NULL;

/* initializes EGAPI */
sts = EssGInit(&InitStruct, &Handle);

/* initializes a specific grid */
if(!sts)
    sts = EssGNewGrid(Handle, &hGrid);

strcpy(Server, "Rainbow");
strcpy(UserName, "Admin");
strcpy(Password, "Password");
strcpy(Application, "Demo");
strcpy(Database, "Basic");
ulOptions = ESSG_CONNECT_NODIALOG;

/* connects the grid to a database on the server */
if(!sts)
    sts = EssGConnect(hGrid, Server, UserName, Password, Application,
                     Database, ulOptions);
}

See Also

- “Using the C Grid API Functions” on page 994
- “C Grid API Structures” on page 1006

**EssGConnectEx**

Connects a grid to an Essbase database using a user authentication token rather than a username and password.

**Syntax**

```c
ESSG_FUNC_M EssGConnectEx (hGrid, Server, Token, Application, Database, ulOptions);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hGrid</td>
<td>ESSG_HGRID_T</td>
<td>Handle passed back from EssGNewGrid.</td>
</tr>
<tr>
<td>Server</td>
<td>ESSG_SERVER_T</td>
<td>Network server name string. The server name can be expressed as hostname or hostname:port.</td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Token</td>
<td>ESSG_TOKEN_T</td>
<td>The token representing the username and password of an authenticated user.</td>
</tr>
<tr>
<td>Username</td>
<td>ESSG_USERNAME_T</td>
<td>Name of valid user at server.</td>
</tr>
<tr>
<td>Password</td>
<td>ESSG_PASSWORD_T</td>
<td>Password of user.</td>
</tr>
<tr>
<td>Application</td>
<td>ESSG_APPLICATION_T</td>
<td>Name of a valid application on server.</td>
</tr>
<tr>
<td>Database</td>
<td>ESSG_DATABASE_T</td>
<td>Name of a valid database for application on server.</td>
</tr>
<tr>
<td>ulOptions</td>
<td>ESSG_ULONG_T</td>
<td>Options flag. Values are ESSG_CONNECT_NODIALOG, which attempts to login and connect without displaying dialog, using the default/passed setting; or ESSG_CONNECT_DEFAULT which will display the login and selection dialog.</td>
</tr>
</tbody>
</table>

Notes

- If this function fails, the corresponding EssGConnect() function is automatically called in order to try to verify a username and password for the user.
- Calls EssAutoLogin, therefore all rules that apply to EssAutoLogin apply to this function. For example, none of the parameters are case-sensitive.
- If ulOptions is set to ESSG_CONNECT_NODIALOG, none of the connection related parameters can be NULL or empty. When ulOptions is set to ESSG_CONNECT_DEFAULT, and a buffer is passed for the connect parameters, the user's selections from the dialog will be returned in these buffers.
- All security information is utilized. Therefore, if the user connects to a database to which he or she does not have read access, all read operations will fail.

Return Value

If successful, returns ESSG_STS_NOERR.

Access

None.

See Also

- EssGConnect
- “Using the C Grid API Functions” on page 994
- “C Grid API Structures” on page 1006

**EssGDeleteLRO**

Deletes a specified LRO from an Essbase database.

Syntax

```c
ESSG_FUNC_M EssGDeleteLRO (hGrid, hLRO);
```
EssGDestroyGrid

Destroys a grid instance.

Syntax

```c
ESSG_FUNC_M EssGDestroyGrid (hGrid)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hGrid</td>
<td>ESSG_HGRID_T</td>
<td>Handle passed back from EssGNewGrid.</td>
</tr>
</tbody>
</table>

Notes

Frees any memory associated with the passed grid handle, and makes the handle invalid.

Return Value

If successful, returns ESSG_STS_NOERR.

Access

None.

Example

```c
#include <essapin.h>
#include <essgapin.h>
EssGDestroyGrid
```
ESSG_FUNC_M stts = ESS_STS_NOERR;
ESSG_INIT_T InitStruct;
ESSG_HANDLE_T Handle;
ESSG_HGRID_T hGrid;

InitStruct.ulVersion = ESSG_VERSION;
InitStruct.ulMaxRows = 1000;
InitStruct.ulMaxColumns = 200;
InitStruct.pfnMessageFunc = ESS_NULL;
InitStruct.pUserData = ESS_NULL;

/* initializes EGAPI */
stts = EssGInit(&InitStruct, &Handle);

/* initializes a specific grid */
if(!stts)
    stts = EssGNewGrid(Handle, &hGrid);

/* destroys a grid instance */
if(!stts)
    stts = EssGDestroyGrid(hGrid);

See Also
- “Using the C Grid API Functions” on page 994
- “C Grid API Structures” on page 1006
- EssGNewGrid

### EssGDisconnect

Disconnects a grid from a database at the server.

**Syntax**

```c
ESSG_FUNC_M EssGDisconnect (hGrid, ulOptions);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hGrid</td>
<td>ESSG_HGRID_T</td>
<td>Handle passed back from EssGNewGrid.</td>
</tr>
<tr>
<td>ulOptions</td>
<td>ESSG_ULONG_T</td>
<td>Reserved for future use. Should be set to zero.</td>
</tr>
</tbody>
</table>

**Return Value**

If successful, returns ESSG_STS_NOERR.

**Example**

```c
#include <essapin.h>
#include <essgapin.h>

ESSG_FUNC_M stts = ESS_STS_NOERR;
ESSG_ULONG_T ulOptions = 0;
ESSG_INIT_T InitStruct;
ESSG_HANDLE_T Handle;
```
ESSG_HGRID_T hGrid;

InitStruct.ulVersion = ESSG_VERSION;
InitStruct.ulMaxRows = 1000;
InitStruct.ulMaxColumns = 200;
InitStruct.pfnMessageFunc = ESS_NULL;
InitStruct.pUserData = ESS_NULL;

/* initializes EGAPI */
sts = EssGInit(&InitStruct, &Handle);

/* initializes a specific grid */
if(!sts)
    sts = EssGNewGrid(&Handle, &hGrid);

/* connects the grid to a database on the server */
if(!sts)
    sts = EssGConnect(hGrid, "Rainbow", "Admin",
                     "Password", "Demo", "Basic",
                     ESSG_CONNECT_DEFAULT);

/* disconnects a grid from database at server */
if(!sts)
    sts = EssGDisconnect(hGrid, ulOptions);

/* terminate the EGAPI */
if(!sts)
    sts = EssGTerm(Handle);
}

See Also

- “Using the C Grid API Functions” on page 994
- “C Grid API Structures” on page 1006

**EssGDTBeginDrillThrough**

Returns the drill-through instance handle for the given data cell range(s).

**Syntax**

```
ESSG_FUNC_M EssGDTBeginDrillThrough (hGrid, usCells, pCells, ppDTInst);
```

**Parameter** | **Data Type** | **Description**
--- | --- | ---

| hGrid; | ESSG_HGRID_T | Handle of the original grid returned by EssGNewGrid(). |
| usCells; | ESSG_USHORT_T | Number of cell ranges in the pCells array. |
| pCells; | “ESSG_RANGE_T” on page 1012 | Array of cell ranges selected to receive drill-through report data. |
| ppDTInst; | ESSG_PPDTHINST_T | Drill-through instance handle returned for the given data cell range(s). |

**See Also**

- “C Grid API Constants” on page 997
EssGDTConnect

Takes drill-through connection information for a given drill-through handle, and connects to Oracle Essbase Studio.

**Syntax**

```
ESSG_FUNC_M EssGDTConnect (pDTInst);
```

**Parameter** | **Data Type** | **Description**
--- | --- | ---
pDTInst | ESSG_PDTINST_T | Initialized drill-through instance handle

**Example**

For a code example, see “C Grid API Drill-Through Example” on page 1105.

**See Also**

- “C Grid API Structures” on page 1006
- EssGDTEndDrillThrough
- EssGDTExecuteReport
- EssGDTGetData
- EssGDTGetInfo
- EssGDTListReports
- EssGDTRequestDrillThrough
- EssGDTSetInfo

EssGDTEndDrillThrough

Ends the drill-through session and frees up memory for the given drill-through instance handle.

**Syntax**

```
ESSG_FUNC_M EssGDTEndDrillThrough (pDTInst);
```
**Parameter** | **Data Type** | **Description**
--- | --- | ---
pDTInst; | ESSG_PDTHINST_T | Initialized drill-through instance handle for the given data cell range.

**Example**
For a code example, see “C Grid API Drill-Through Example” on page 1105.

**See Also**
- “C Grid API Constants” on page 997
- “C Grid API Structures” on page 1006
- EssGDTConnect
- EssGDTExecuteReport
- EssGDTGetData
- EssGDTGetHeader
- EssGDTGetInfo
- EssGDTListReports
- EssGDTRequestDrillThrough
- EssGDTSetInfo

**EssGDTExecuteReport**
Executes the report identified by its index to an array of report structures.

**Syntax**

```c
ESSG_FUNC_M EssGDTExecuteReport (pDTInst, Index);
```

**Parameter** | **Data Type** | **Description**
--- | --- | ---
pDTInst; | ESSG_PDTHINST_T | Initialized drill-through instance handle
Index; | ESSG_ULONG_T | Index of the report to be executed

**Example**
For a code example, see “C Grid API Drill-Through Example” on page 1105.

**See Also**
- “C Grid API Constants” on page 997
- “C Grid API Structures” on page 1006
- EssGDTConnect
- EssGDTEndDrillThrough
- EssGDTGetData
- EssGDTGetHeader
- EssGDTGetInfo
- EssGDTListReports
- EssGDTRequestDrillThrough
- EssGDTSetInfo
EssGDTGetData

Retrieves an array of report data for the given drill-through instance handle.

Syntax

```c
ESSG_FUNC_M EssGDTGetData (pDTInst, ppData, pulCount);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pDTInst;</td>
<td>ESSG_PDTHINST_T</td>
<td>Initialized drill-through instance handle.</td>
</tr>
<tr>
<td>ppData;</td>
<td>“ESSG_DTDATA_T” on page 1009</td>
<td>Array of report data structures for given data cells.</td>
</tr>
<tr>
<td>pulCount;</td>
<td>ESSG_PULONG_T</td>
<td>Count of data blocks in the <code>ppData</code> array.</td>
</tr>
</tbody>
</table>

Notes

- Call `EssGDTGetData()` until `pulCount` is 0 (zero).
- Free memory for `ppData` (ESSG_DTDATA_T) with `EssFree()` after you call `EssGDTGetData()`.

Example

For a code example, see “C Grid API Drill-Through Example” on page 1105.

See Also

- “C Grid API Constants” on page 997
- “C Grid API Structures” on page 1006
- `EssGDTConnect`
- `EssGDTEndDrillThrough`
- `EssGTDExecuteReport`
- `EssGDTGetHeader`
- `EssGDTGetInfo`
- `EssGDTListReports`
- `EssGDTRequestDrillThrough`
- `EssGDTSetInfo`
- `EssFree`

EssGDTGetHeader

Retrieves the report data header information for the given drill-through instance handle.

Syntax

```c
ESSG_FUNC_M EssGDTGetHeader (pDTInst, ppHeader, pulCount);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pDTInst;</td>
<td>ESSG_PDTHINST_T</td>
<td>Initialized drill-through instance handle.</td>
</tr>
<tr>
<td>ppHeader;</td>
<td>“ESSG_DTHEADER_T” on page 1009</td>
<td>Array of header information structures for given columns.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>---------------</td>
<td>-----------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>pulCount</td>
<td>ESSG_PULONG_T</td>
<td>Count of data blocks in the <code>ppHeader</code> header information array.</td>
</tr>
</tbody>
</table>

**Notes**

Free memory for `ppHeader` (ESSG_DTHEADER_T) with `EssFree()` after you call `EssGDTGetHeader()`.

**Example**

For a code example, see “C Grid API Drill-Through Example” on page 1105.

**See Also**

- “C Grid API Constants” on page 997
- “C Grid API Structures” on page 1006
- `EssGDTConnect`
- `EssGDTEndDrillThrough`
- `EssGDTExecuteReport`
- `EssGDTGetData`
- `EssGDTGetInfo`
- `EssGDTListReports`
- `EssGDTRequestDrillThrough`
- `EssGDTSetInfo`
- `EssFree`

---

**EssGDTGetInfo**

Retrieves drill-through connection information for a given drill-through handle.

**Syntax**

```c
ESSG_FUNC_M EssGDTGetInfo (pDTInst, pDTInfo);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pDTInst</td>
<td>ESSG_PDTINST_T</td>
<td>Initialized drill-through instance handle</td>
</tr>
<tr>
<td>pDTInfo</td>
<td>“ESSG_DTINFO_T” on page 1010</td>
<td>Pointer to a structure of connection information for a given range of data cells</td>
</tr>
</tbody>
</table>

**Notes**

- Allocate memory for `ESSG_DTINFO_T` before you call `EssGDTGetInfo()`.
- `password` is not returned in `pDTInfo`; that is, the `password` field in `ESSG_DTINFO_T` is not returned.

**Example**

For a code example, see “C Grid API Drill-Through Example” on page 1105.
See Also

- “C Grid API Constants” on page 997
- “C Grid API Structures” on page 1006
- EssGDTConnect
- EssGDTEndDrillThrough
- EssGDTExecuteReport
- EssGDTGetData
- EssGDTGetHeader
- EssGDTListReports
- EssGDTRequestDrillThrough
- EssGDTSetInfo

**EssGDTGetReportData**

Executes the predefined default drill-through report for the given data cell range, and returns report data via the given grid handle `hDAGrid`.

**Syntax**

```c
ESSG_FUNC_M EssGDTGetReportData (hGrid, hDAGrid, usCells, pCells);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hGrid;</td>
<td>ESSG_HGRID_T</td>
<td>Handle of the original grid returned by <code>EssGNewGrid()</code>.</td>
</tr>
<tr>
<td>hDAGrid;</td>
<td>ESSG_HGRID_T</td>
<td>Handle of the new grid to receive drill-through report data.</td>
</tr>
<tr>
<td>usCells;</td>
<td>ESSG_USHORT_T</td>
<td>Number of cell ranges in the <code>pCells</code> array.</td>
</tr>
<tr>
<td>pCells;</td>
<td>“ESSG_RANGE_T” on page 1012</td>
<td>Array of cell ranges selected to receive drill-through report data.</td>
</tr>
</tbody>
</table>

See Also

- “C Grid API Constants” on page 997
- “C Grid API Structures” on page 1006
- EssGDTEndDrillThrough
- EssGDTGetData
- EssGDTGetHeader
- EssGDTGetInfo
- EssGInit
- EssGDTListReports
- EssGDTReportCount
- EssGDTRequestDrillThrough

**EssGDTListReports**

Returns an array of report structures for the given drill-through instance handle.
Syntax

```c
ESSG_FUNC_M EssGDTListReports (pDTInst, ppReports, pulCount);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pDTInst;</td>
<td>ESSG_PDTINST_T</td>
<td>Initialized drill-through instance handle</td>
</tr>
<tr>
<td>ppReports;</td>
<td>&quot;ESSG_DTREPORT_T&quot; on page 1010</td>
<td>An array of report structures for the given drill-through instance handle</td>
</tr>
<tr>
<td>pulCount;</td>
<td>ESSG_PULONG_T</td>
<td>Number of blocks in the ppReports header information array</td>
</tr>
</tbody>
</table>

Example

For a code example, see “C Grid API Drill-Through Example” on page 1105.

See Also

- “C Grid API Constants” on page 997
- “C Grid API Structures” on page 1006
- EssGDTConnect
- EssGDTEndDrillThrough
- EssGDTExecuteReport
- EssGDTGetData
- EssGDTGetHeader
- EssGDTGetInfo
- EssGDTRequestDrillThrough
- EssGDTSetInfo

**EssGDTReportCount**

Returns the number of reports defined for the given data cell range(s).

Syntax

```c
ESSG_FUNC_M EssGDTReportCount (hGrid, usCells, pCells, uspReportNum);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hGrid;</td>
<td>ESSG_HGRID_T</td>
<td>Handle of the original grid returned by EssGNewGrid().</td>
</tr>
<tr>
<td>usCells;</td>
<td>ESSG_USHORT_T</td>
<td>Number of cell ranges in the pCells array.</td>
</tr>
<tr>
<td>pCells;</td>
<td>&quot;ESSG_RANGE_T&quot; on page 1012</td>
<td>Array of cell ranges selected to receive drill-through report data.</td>
</tr>
<tr>
<td>uspReportNum;</td>
<td>ESSG_PUSHORT_T</td>
<td>Number of reports defined for the given data cell range(s).</td>
</tr>
</tbody>
</table>

See Also

- “C Grid API Constants” on page 997
- “C Grid API Structures” on page 1006
- EssGDTEndDrillThrough
- EssGDTGetData
EssGDTRequestDrillThrough

Returns the drill-through instance handle for the given data cell range.

Syntax

```c
ESSG_FUNC_M EssGDTRequestDrillThrough (hGrid, usCells, pCells, ppDTInst);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hGrid;</td>
<td>ESSG_HGRID_T</td>
<td>Handle of the original grid returned by EssGNewGrid().</td>
</tr>
<tr>
<td>usCells;</td>
<td>ESSG_USHORT_T</td>
<td>Number of cell ranges in the pCells array.</td>
</tr>
<tr>
<td>pCells;</td>
<td>“ESSG_RANGE_T” on page 1012</td>
<td>Array of cell ranges selected to receive drill-through report data.</td>
</tr>
<tr>
<td>ppDTInst;</td>
<td>ESSG_PPDTHINST_T</td>
<td>Drill-through instance handle returned for the given data cell range(s).</td>
</tr>
</tbody>
</table>

Notes

- Sends a request to the Essbase Server for an optimized Extended Member Comment
- Initializes a drill-through session with the given Extended Member Comment
- Returns the drill-through instance handle, ppDTInst.

Example

For a code example, see “C Grid API Drill-Through Example” on page 1105.

See Also

- “C Grid API Constants” on page 997
- “C Grid API Structures” on page 1006
- EssGDTConnect
- EssGDTEndDrillThrough
- EssGDTEncodeReport
- EssGDTGetData
- EssGDTGetHeader
- EssGDTGetInfo
- EssGDTListReports
- EssGDTSetInfo
- EssOtlGetMemberCommentEx
- EssOtlSetMemberCommentEx
**EssGDTSetInfo**

Sets drill-through connection information for a given drill-through handle.

**Syntax**

```c
ESSG_FUNC_M EssGDTSetInfo (pDTInst, pDTInfo);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pDTInst</td>
<td>ESSG_PDTHINST_T</td>
<td>Initialized drill-through instance handle</td>
</tr>
<tr>
<td>pDTInfo</td>
<td>&quot;ESSG_DTINFO_T&quot;</td>
<td>Pointer to a structure of connection information for a given range of data cells</td>
</tr>
</tbody>
</table>

**Notes**

The `inputOption` field in `ESSG_DTINFO_T` is ignored.

**Example**

For a code example, see “C Grid API Drill-Through Example” on page 1105.

**See Also**

- “C Grid API Constants” on page 997
- “C Grid API Structures” on page 1006
- `EssGDTConnect`
- `EssGDTEndDrillThrough`
- `EssGDTExecuteReport`
- `EssGDTGetData`
- `EssGDTGetHeader`
- `EssGDTGetInfo`
- `EssGDTListReports`
- `EssGDTRequestDrillThrough`

**EssGEndOperation**

Frees any internal resources used after the operation is complete and all rows have been returned.

**Syntax**

```c
ESSG_FUNC_M EssGEndOperation (hGrid, ulOptions);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hGrid</td>
<td>ESSG_HGRID_T</td>
<td>Handle passed back from EssGNewGrid.</td>
</tr>
<tr>
<td>ulOptions</td>
<td>ESSG_ULONG_T</td>
<td>Reserved for future use. Should be set to zero.</td>
</tr>
</tbody>
</table>
Notes

This call is optional and can be made to free internal resources after an operation is complete. If you do not make this call, internal resources are freed when the next operation starts, or when the caller disconnects the grid, whichever comes first.

Return Value

If successful, returns ESSG_STS_NOERR.

Access

None.

Example

EssGEndOperation(hGrid, 0);

See an example that uses this code in the EssGBeginRetrieve Example section.

See Also

- “Using the C Grid API Functions” on page 994
- “C Grid API Structures” on page 1006

EssGFreeCellLinkResults

Releases all resources reserved to store the links resulting from a previous call to EssGGetCellLinkResults().

Syntax

ESSG_FUNC_M EssGFreeCellLinkResults(hGrid, pDrillData);

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hGrid</td>
<td>ESSG_HGRID_T</td>
<td>Grid handle returned by EssGNewGrid().</td>
</tr>
<tr>
<td>pDrillData</td>
<td>“ESSG_DRILLDATA_T” on page 1008</td>
<td>Reference to an array of ESSG_DRILLDATA_T structures containing information about the linked objects.</td>
</tr>
</tbody>
</table>

Notes

EssGGetCellLinkResults() takes a reference to a pointer to ESSG_DRILLDATA_T, but that this function only requires the pointer to ESSG_DRILLDATA_T.

Return Value

If successful, returns ESSG_STS_NOERR.

See Also

- “Using the C Grid API Functions” on page 994
- “C Grid API Structures” on page 1006
- EssGBeginCreateLRO
- EssGBeginDeleteLROs
EssGFreeMemberInfo

Frees any data returned by any call that returns member information, including EssGGetMemberInfo, and EssGGetDataPointResults.

Syntax

```c
ESSG_FUNC_M EssGFreeMemberInfo (hGrid, ulMembers, pszMembers);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hGrid</td>
<td>ESSG_HGRID_T</td>
<td>Handle passed back from EssGNewGrid.</td>
</tr>
<tr>
<td>ulMembers</td>
<td>ESSG_ULONG_T</td>
<td>Describes the number of elements in the <code>ppszMembers</code> array to be freed.</td>
</tr>
<tr>
<td>pszMembers</td>
<td>ESSG_PSTR_T</td>
<td>Pointer to a one-dimensional array of member names to be freed.</td>
</tr>
</tbody>
</table>

Notes

The parameters to this function include the number of elements in the one-dimensional array, and the one-dimensional array of member names itself.

Return Value

If successful, returns ESSG_STS_NOERR.

Access

None.

Example

```c
EssGFreeMemberInfo(hGrid, ulMembers, pszMembers);
```

See an example that uses this code in `EssGGetMemberInfo`.

See Also

- “Using the C Grid API Functions” on page 994
- “C Grid API Structures” on page 1006

EssGFreeMemberwKeyStr

Frees the combined string of member name and member key that is created by EssGCreateMemberwKeyStr.
Syntax

`EssGFreeMemberwKeyStr (pszStr);`

Parameter | Data Type | Description
---|---|---
pszStr | ESSG_STR_T | Input. Combined member/key string of the format: `<member-name><length><member-name><key length><key>`

Example

```c
ESSG_VOID_T ESSG_BeginZoomIn(ESSG_HGRID_T hGrid)
{
    ESSG_STS_T      sts = ESS_STS_NOERR;
    ESSG_DATA_T **ppDataIn;
    ESSG_DATA_T **ppDataOut;
    ESSG_RANGE_T rDataRangeIn, rDataRangeOut;
    ESSG_ULONG_T ulOptions;
    ESSG_USHORT_T usCells;
    ESSG_RANGE_T pZoomCells;
    ESSG_USHORT_T usState;
    ESSG_USHORT_T usMember2Len, usKey2Len;
    ESSG_SHORT_T    sOption, sOptionGet;
    ESSG_PVOID_T   pOption, pOptionGet;
    ESSG_STR_T     pMember, pKey, pOutStr;
    ESSG_STR_T     pMember2, pKey2;

    /* connect the grid to a database on the server */
    sts = EssGConnect(hGrid, server, "essexer", pwd, app, db, ESSG_CONNECT_NODIALOG);

    /* set grid option*/
    tmpShort = ESSG_TRUE;
    sOption = ESSG_OP_MEMBERANDUNIQUENAME ;
    pOption = (ESSG_PVOID_T)tmpShort;     // pOption holds the actual value not a pointer

    sts = EssGSetGridOption(hGrid, sOption, pOption);
    printf("EssGSetGridOption  sts  %ld\n",sts);

    sOptionGet = ESSG_OP_MEMBERANDUNIQUENAME ;
    pOptionGet = &tmpShortGet;
    if(!sts)
    {
        sts = EssGGetGridOption(hGrid, sOptionGet, pOptionGet);
        printf("EssGGetGridOption  sts  %ld\n",sts);
        printf("EssGSetGridOption set ESSG_OP_MEMBERANDUNIQUENAME TO %d\n",
                (int)tmpShortGet);
    }

    if(sts == 0)
    {
        ppDataIn = BuildTable(&rDataRangeIn);
        ulOptions = ESSG_ZOOM_DOWN | ESSG_NEXTLEVEL;

        pZoomCells_ulRowStart = 0;
    }
```
pZoomCells.ulColumnStart = 2;
pZoomCells.ulNumRows = 1;
pZoomCells.ulNumColumns = 1;
usCells = 1;

/* start the zoom in operation */
sts = EssGBeginZoomIn(hGrid, usCells, &pZoomCells, ulOptions);
printf("EssGBeginZoomIn  sts:  %ld\n",sts);
}

//Display Input
DisplayOutput(ppDataIn, rDataRangeIn);
printf("\n\n");
if(sts == 0)
/* send the entire grid to define the query */
sts = EssGSendRows(hGrid, &rDataRangeIn, ppDataIn);

if(sts == 0)
{
/* perform the zoom-in */
sts = EssGPerformOperation(hGrid, 0);

/* Free the built data */
FreeTwoDim(ppDataIn, rDataRangeIn.ulNumRows);
}
if (sts == 0)
{
/* determine the results of the zoom-in */
sts = EssGGetResults(hGrid, 0, &rDataRangeOut, &usState);
}
if(sts ==0)
{
/* get all the data */
sts = EssGGetRows(hGrid, 0, &rDataRangeOut, &rDataRangeOut, &ppDataOut);
}
if(sts == 0)
{
DisplayOutput(ppDataOut, rDataRangeOut);

/* Retreive member and key from cell */
sts = EssGGetFromMemberwKey (((ppDataOut[1][0]).Value).pszStr, &pMember, &pKey);
printf("After EssGGetFromMemberwKey\n Member: %s, Key: %s \n\n", 
pMember+2,
pKey+2);

//Member is  "Qtr1", Key is ":[2004].[Qtr1]", pOutStr is in the format
//nn<member-name>nn<'key> - where nn is string length

usMember2Len = strlen("Qtr1");
pMember2 = malloc(usMember2Len+3);
memset(pMember2, 0, usMember2Len+3);
usKey2Len = strlen("[2004].[Qtr1]"泓); pKey2 = malloc(usKey2Len+3);
memset(pKey2, 0, usKey2Len+3);
memcpy(pMember2, &usMember2Len, 2);
memcpy(pMember2+2, "Qtr1", usMember2Len);
memcpy(pKey2, &usKey2Len, 2);
memcpy(pKey2+2, "[2004].[Qtr1]", usKey2Len);

sts = EssGCreateMemberwKeyStr(pMember2, pKey2, &pOutStr);

/* Note: because not all elements in pOutStr are actual characters, 
e.g. the 2 bytes for the size of Member and size of Key, plus the \
\0 ending characters, the printf below does not display the actual 
contents of the array */
for (i=0; i < usMember2Len + usKey2Len + 4 + 2; ++i)
printf("%c", pOutStr[i]);

/* Free the returned data */
EssGFreeRows(hGrid, &DataRangeOut, ppDataOut);
sts = EssGFreeMemberwKeyStr (pOutStr);

}

if( sts == 0)
{
   EssGEndOperation(hGrid, 0);
   EssGDisconnect(hGrid, 0);
}

See Also

- EssGCreateMemberwKeyStr
- EssGGetFromMemberwKey

EssGFreeRows

Frees data that has been returned via EssGGetRows.

Syntax

ESSG_FUNC_M EssGFreeRows (hGrid, pRange, ppData);

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hGrid</td>
<td>ESSG_HGRID_T</td>
<td>Handle passed back from EssGNewGrid.</td>
</tr>
<tr>
<td>pRange</td>
<td>&quot;ESSG_RANGE_T&quot; on page 1012</td>
<td>Describes the extent of the data.</td>
</tr>
<tr>
<td>ppData</td>
<td>&quot;ESSG_DATA_T&quot; on page 1006</td>
<td>A two-dimensional array of data to be freed.</td>
</tr>
</tbody>
</table>

Return Value

If successful, returns ESSG_STS_NOERR.
Access
None.

Example

```c
EssGFreeRows(hGrid, &rDataRangeOut, ppDataOut);
```

See an example that uses this code in the EssGBeginRetrieve Example section.

See Also
- “Using the C Grid API Functions” on page 994
- “C Grid API Structures” on page 1006

**EssGGetAPIContext**

Gets the API login context handle for the specified grid.

**Syntax**

```c
ESSG_FUNC_M EssGGetAPIContext (hGrid, pEssHctx);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hGrid</td>
<td>ESSG_HGRID_T</td>
<td>Handle passed back from EssGNewGrid.</td>
</tr>
<tr>
<td>pEssHctx</td>
<td>ESSG_PPVOID_T</td>
<td>Variable for the return of the API context handle of the connected grid.</td>
</tr>
</tbody>
</table>

**Notes**

- This allows the caller to call non-Grid API functions that require a login context handle.
- If there is no valid connection to the server, there is no valid API context handle and the call will fail and set *pEssHctx to ESS_INVALID_HCTX.
- Do not use the returned login context in API functions that would change the context information.

**Return Value**

If successful, returns ESSG_STS_NOERR.

**Example**

```c
#include <essapin.h>
#include <essgapin.h>

ESSG_FUNC_M     sts = ESS_STS_NOERR;
ESSG_INIT_T     InitStruct;
ESSG_HANDLE_T   Handle;
ESSG_PVOID_T    EssHctx;
ESSG_HGRID_T    hGrid;

InitStruct.ulVersion = ESSG_VERSION;
InitStruct.ulMaxRows = 1000;
InitStruct.ulMaxColumns = 200;
InitStruct.pfnMessageFunc = ESS_NULL;
```
InitStruct.pUserData = ESS_NULL;

/* initializes EGAPI */
sts = EssGInit(&InitStruct, Handle);

if(!sts)
    sts = EssGNewGrid(Handle, &hGrid);

/* connect the grid to a database on the server */
if(!sts)
    sts = EssGConnect(hGrid, "Rainbow", "Admin",
                      "Password", "Demo",
                      "Basic", ESSG_CONNECT_DEFAULT);

/* Get API context handle for the specified grid */
if(!sts)
    sts = EssGGetAPIContext(hGrid, &EssHctx);
}

See Also

- “Using the C Grid API Functions” on page 994
- “C Grid API Structures” on page 1006

EssGGetAPInstance

Gets the API initialization instance handle.

Syntax

`ESSG_FUNC_M EssGGetAPInstance (Handle, pEssHinst);`

Parameter | Data Type | Description
--- | --- | ---
Handle | ESSG_HANDLE_T | Handle passed back from EssGInit.
pEssHinst | ESSG_PPVOID_T | Variable for the return of the API instance handle used by the Grid API.

Notes

This handle the caller to call non-Grid API functions that require an instance handle.

Return Value

If successful, returns ESSG_STS_NOERR.

Access

None.

Example

```
#include <essapin.h>
#include <essgapin.h>

ESSG_FUNC_M     sts = ESS_STS_NOERR;
ESSG_PVOID_T    EssHinst;
```
ESSG_INIT_T InitStruct;
ESSG_HANDLE_T Handle;

InitStruct.ulVersion = ESSG_VERSION;
InitStruct.ulMaxRows = 1000;
InitStruct.ulMaxColumns = 200;
InitStruct.pfnMessageFunc = ESS_NULL;
InitStruct.pUserdata = ESS_NULL;

/* initializes EGAPI */
sts = EssGInit(&InitStruct, Handle);

/* get API initialization instance handle */
if(!sts)
  sts = EssGGetAPIInstance(Handle, &EssHinst);

See Also

● “Using the C Grid API Functions” on page 994
● “C Grid API Structures” on page 1006

EssGGetCellLinkResults

Retrieves a list of links resulting from a previous call to EssGBeginDrillOrLink().

Syntax

ESSG_FUNC_M EssGGetCellLinkResults (hGrid, pfCanDrill, pNumLROs,
ppDrillData, pRangeOut, pState);

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hGrid;</td>
<td>ESSG_HGRID_T</td>
<td>Grid handle returned by EssGNewGrid().</td>
</tr>
<tr>
<td>pfCanDrill;</td>
<td>ESSG_PBOOL_T</td>
<td>Returns True if the cell has linked objects. If you request Zoom-In results by specifying the ESSG_OPT_ZOOM option with EssGBeginDrillOrLink(), pfCanDrill returns False.</td>
</tr>
<tr>
<td>pNumLROs;</td>
<td>ESSG_PULONG_T</td>
<td>Returns the number of links retrieved.</td>
</tr>
<tr>
<td>ppDrillData;</td>
<td>“ESSG_DRILLDATA_T” on page 1008</td>
<td>Returns references to an array of ESSG_DRILLDATA_T structures containing information about the linked objects.</td>
</tr>
<tr>
<td>pRangeOut;</td>
<td>“ESSG_RANGE_T” on page 1012</td>
<td>Returns the cell ranges for the Zoom-In if no LROs were found and you specified the ESSG_OPT_ZOOM option in your call to EssGBeginDrillOrLink().</td>
</tr>
<tr>
<td>pState;</td>
<td>ESSG_PUSHORT_T</td>
<td>Returns one of the following states of operation:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● In progress—Not all cells have been retrieved</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● Done—All cells have been retrieved</td>
</tr>
</tbody>
</table>

Notes

● This function allocates memory for ESSG_DRILLDATA_T. Release that memory with a call to EssGFreeCellLinkResults.
The handles retrieved from this function are valid until the next call to any of the following Grid API functions:

- EssGBeginDrillOrLink()
- EssGBeginCreateLRO()
- EssGUpdateLRO()
- EssGBeginDeleteLRO()
- EssGDeleteLRO()

**Return Value**

If successful, returns ESSG_STS_NOERR.

**See Also**

- “Using the C Grid API Functions” on page 994
- “C Grid API Structures” on page 1006
- EssGBeginRemoveOnly
- EssGGetGridOption
- EssGPerformOperation
- EssGSetGridOption
- EssGBeginCreateLRO
- EssGBeginDrillAcross
- EssGBeginDeleteLROs
- EssGBeginDrillOrLink
- EssGDeleteLRO
- EssGFreeCellLinkResults
- EssGGetLRODesc
- EssGGetLinkedPartitionDesc
- EssGGetLRO
- EssGUpdateLRO

---

**EssGGetDataPointResults**

Retrieves information from the EssGBeginDataPoint call (EssGBeginDataPoint, EssGSendRows, EssGPerformOperation).

**Syntax**

```c
ESSG_FUNC_M EssGGetDataPointResults (hGrid, pulMembers, ppszMembers, pState);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hGrid</td>
<td>ESSG_HGRID_T</td>
<td>Handle passed back from EssGNewGrid.</td>
</tr>
<tr>
<td>pulMembers</td>
<td>ESSG_PULONG_T</td>
<td>Count of members being returned.</td>
</tr>
</tbody>
</table>

---
**Parameter** | **Data Type** | **Description**
--- | --- | ---
*ppszMembers | ESSG_PSTR_T | Pointer to a one dimensional array of size pulMembers of members returned from the server. The API allocates this memory and should be freed by the caller using EssGFreeMemberInfo.

**Note:** The *ppszMembers* parameter should be freed by the caller using EssGFreeMemberInfo.

pState | ESSG_PUSHORT_T | Variable for the return of the state of the operation. This can be one of the following values:
- ESSG_STATE_DONE Operation complete
- ESSG_STATE_INPROGRESS The operation is in progress

**Notes**

Make this call multiple times until the *pState* variable returns ESSG_STATE_DONE.

**Return Value**

If successful, returns ESSG_STS_NOERR.

**Access**

None.

**Example**

```c
sts = EssGGetDataPointResults(hGrid, &ulMembers,
```

See an example that uses this code in the EssGBeginDataPoint Example section.

**See Also**

- “Using the C Grid API Functions” on page 994
- “C Grid API Structures” on page 1006

---

**EssGGetFormattedValue**

Returns the formatted value for the given cell.

**Syntax**

```c
ESS_FUNC_M EssGGetFormattedValue(hGrid, pData, *fmtVal)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hGrid</td>
<td>ESSG_HGRID_T</td>
<td>Grid handle</td>
</tr>
<tr>
<td>pData</td>
<td>ESSG_PDATA_T</td>
<td>Pointer to the ESSG_DATA_T structure of the cell.</td>
</tr>
<tr>
<td>*fmtVal</td>
<td>ESSG_STR_T</td>
<td>Pointer to formatted value for this cell</td>
</tr>
</tbody>
</table>

---

EssGGetFormattedValue 1075
Not  

- The grid option ESSG_OP_GET_FORMATTED_VALUE should be turned on to obtain the formatted values.
- You do not need to free the returned pointer, as this is managed by the API.

**Return Value**
- 0—If successful
- Error number—If unsuccessful

**EssGGetFromMemberwKey**

Returns a member name and key. A key is a value generated by Essbase that uniquely identifies a member name in the outline.

**Syntax**

```
ESSG_FUNC_M EssGGetFromMemberwKey (pszOutStr, pszMember, pszKey);
```

**Parameter**  

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pszOutStr</td>
<td>ESSG_STR_T Input string of the format: &lt;member-name length&gt;&lt;member-name&gt;&lt;key length &gt;&lt;key&gt;, where the length elements are 2 bytes in size. Note that &lt;member-name&gt; is null-terminated. The string is returned from the API or can be created using EssGCreateMemberwKeyStr.</td>
</tr>
<tr>
<td>pszMember</td>
<td>ESSG_STR_T Member name (output).</td>
</tr>
<tr>
<td>pszKey</td>
<td>ESSG_STR_T Member key (output).</td>
</tr>
</tbody>
</table>

**Notes**

When the usType field of the ESSG_DATA_T structure is set to ESSG_DT_MEMBERwKEY, then the pszStr field of Value(ESSG_DATA_VALUE) field is interpreted as the format required for pszOutStr.

**Example**

```c
ESSG_VOID_T ESSG_BeginZoomIn(ESSG_HGRID_T hGrid)
{
    ESSG_STS_T      sts = ESS_STS_NOERR;
    ESSG_DATA_T  **ppDataIn;
    ESSG_DATA_T  **ppDataOut;
    ESSG_RANGE_T rDataRangeIn, rDataRangeOut;
    ESSG_ULONG_T ulOptions;
    ESSG_USHORT_T usCells;
    ESSG_RANGE_T pZoomCells;
    ESSG_USHORT_T usState;
    ESSG_USHORT_T usMember2Len, usKey2Len;
    ESSG_SHORT_T   sOption, sOptionGet;
    ESSG_SHORT_T   tmpShort, tmpShortGet, i;
    ESSG_PVOID_T   pOption, pOptionGet;
    ESSG_STR_T   pMember, pKey, pOutStr;
    ESSG_STR_T   pMember2, pKey2;
```
/* connect the grid to a database on the server */
sts = EssGConnect(hGrid, server, "essexer", pwd, app, db, ESSG_CONNECT_NODIALOG);

/* set grid option*/
tmpShort = ESSG_TRUE;
soption = ESSG_OP_MEMBERANDUNIQUENAME ;
poption = (ESSG_PVOID_T)tmpShort;     // pOption holds the actual value not a pointer

    sts = EssGSetGridOption(hGrid, soption, pOption);
printf("EssGSetGridOption  sts  %ld\n",sts);
soptionGet = ESSG_OP_MEMBERANDUNIQUENAME ;
poptionGet = &tmpShortGet;
    if(!sts)
    {
        sts = EssGGetGridOption(hGrid, soptionGet, pOptionGet);
printf("EssGGetGridOption  sts  %ld\n",sts);
printf("EssGSetGridOption set ESSG_OP_MEMBERANDUNIQUENAME TO %d\n",
(int)tmpShortGet);
    }

if(sts == 0)
{    
    ppDataIn = BuildTable(&rDataRangeIn);
    ulOptions = ESSG_ZOOM_DOWN | ESSG_NEXTLEVEL;

    pZoomCells.ulRowStart = 0;
pZoomCells.ulColumnStart = 2;
pZoomCells.ulNumRows = 1;
pZoomCells.ulNumColumns = 1;
usCells = 1;

    /* start the zoom in operation */
    sts = EssGBeginZoomIn(hGrid, usCells, &pZoomCells, ulOptions);
printf("EssGBeginZoomIn  sts:  %ld\n",sts);
}
//Display Input
DisplayOutput(ppDataIn, rDataRangeIn);
printf("\n\n");
if(sts == 0)/* send the entire grid to define the query */
sts = EssGSendRows(hGrid, &rDataRangeIn, ppDataIn);

if(sts == 0)
{ /* perform the zoom-in */
sts = EssGPerformOperation(hGrid, 0);

/* Free the built data */
FreeTwoDim(ppDataIn, rDataRangeIn.ulNumRows);
}
if (sts == 0)
{ /* determine the results of the zoom-in */
  sts = EssGGetResults(hGrid, 0, &rDataRangeOut, &usState);
} if(sts == 0)
{
  /* get all the data */
  sts = EssGGetRows(hGrid, 0, &rDataRangeOut, &rDataRangeOut, &ppDataOut);
}

if(sts == 0)
{
  DisplayOutput(ppDataOut, rDataRangeOut);

  /* Retireve member and key from cell */
  sts = EssGGetFromMemberwKey(((ppDataOut[1][0]).Value).pszStr, &pMember, &pKey);
  printf("After EssGGetFromMemberwKey
Member: %s, Key: %s \n\n",
    pMember+2,
    pKey+2);

  //Member is "Qtr1", Key is "[2004].[Qtr1]", pOutStr is in the format
  //nn<member-name>nn<'key> - where nn is string length

  usMember2Len = strlen("Qtr1");
  pMember2 = malloc(usMember2Len+3);
    memset(pMember2, 0, usMember2Len+3);
  usKey2Len = strlen("[2004].[Qtr1]" );
  pKey2 = malloc(usKey2Len+3);
    memset(pKey2, 0, usKey2Len+3);

  memcpy(pMember2, &usMember2Len, 2);
  memcpy(pMember2+2, "Qtr1", usMember2Len);
  memcpy(pKey2, &usKey2Len, 2);
  memcpy(pKey2+2, "[2004].[Qtr1]", usKey2Len);

  sts = EssGCreateMemberwKeyStr(pMember2, pKey2, &pOutStr);

  /*Note: because not all elements in pOutStr are actual characters,
   * e.g. the 2 bytes for the size of Member and size of Key, plus the
   * \0 ending characters, the printf below does not display the actual
   * contents of the array */
  for (i=0; i < usMember2Len + usKey2Len + 4 + 2; ++i)
    printf("%c", pOutStr[i]);

  /* Free the returned data */
  EssGFreeRows(hGrid, &rDataRangeOut, ppDataOut);
  sts = EssGFreeMemberwKeyStr (pOutStr);
}

if( sts == 0)
{
  EssGEndOperation(hGrid, 0);
  EssGDisconnect(hGrid, 0);
}
See Also

- EssGCreateMemberwKeyStr
- EssGFreeMemberwKeyStr

**EssGGetGridOption**

Gets individual grid options.

**Syntax**

```c
ESSG_FUNC_M EssGGetGridOption (hGrid, sOption, pOption);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hGrid</td>
<td>ESSG_HGRID_T</td>
<td>Handle passed back from EssGNewGrid().</td>
</tr>
<tr>
<td>sOption</td>
<td>ESSG_SHORT_T</td>
<td>Number indicating what option is being retrieved.</td>
</tr>
<tr>
<td>pOption</td>
<td>ESSG_PVOID_T</td>
<td>Pointer to the option retrieved. With the exception of the ESSG_OP_USERGRIDDATA pointer, this data is read-only and should not be freed by the caller.</td>
</tr>
</tbody>
</table>

**Return Value**

If successful, returns ESSG_STS_NOERR.

**Example**

```c
ESSG_VOID_T EssG_GetGridOption(ESSG_HGRID_T hGrid)
{
    ESSG_FUNC_M    sts = ESS_STS_NOERR;
    ESSG_SHORT_T   sOption;    
    ESSG_SHORT_T   tmpShort;   
    ESSG_PVOID_T   pOption;    

    /* connect the grid to a database on the server */
    sts = EssGConnect(hGrid, "Rainbow", "Admin", "Password", "Demo", "Basic",
                      ESSG_CONNECT_DEFAULT);
    /* get grid option */
    sOption = ESSG_OP_DRILLLEVEL;
    pOption = &tmpShort;
    if(!sts)
        sts = EssGGetGridOption(hGrid, sOption, pOption);
    if(!sts)
    {
        printf("\n%s: %d", "DRILLLEVEL", tmpShort);
        EssGD Disconnect(hGrid, 0);
    }
}
```

See Also

- “Using the C Grid API Functions” on page 994
- “C Grid API Structures” on page 1006
EssGGetGridPerspective

Returns the perspective for a grid.

**Syntax**

```c
ESSG_FUNC_M EssGGetGridPerspective(hGrid, sAttrdim, *pPerspectiveType, *pPerspectiveString)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hGrid</td>
<td>ESSG_HGRID_T</td>
<td>Handle passed back from EssNewGrid().</td>
</tr>
<tr>
<td>sAttrdim</td>
<td>ESSG_STR_T</td>
<td>Attribute dimension name for which the perspective is queried</td>
</tr>
<tr>
<td>*pPerspectiveType</td>
<td>ESSG_SHORT_T</td>
<td>Type of perspective. See “Grid Perspective Types” on page 1003.</td>
</tr>
<tr>
<td>*pPerspectiveString</td>
<td>ESSG_STR_T</td>
<td>Pointer to perspective tuple set being returned.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Null for perspective types other than ESSG_PERSP_EXPLICIT.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• For ESSG_PERSP_EXPLICIT, this value should be explicitly freed.</td>
</tr>
</tbody>
</table>

**Return Value**

- 0—If successful
- Error number—If unsuccessful

**See Also**

- EssGSetGridPerspective

EssGGetIsCellDrillable

Checks whether a cell is associated with a drill-through URL.

**Syntax**

```c
ESS_FUNC_M EssGGetIsCellDrillable (hGrid, pData, pIsDrillable);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hGrid</td>
<td>ESSG_HGRID_T</td>
<td>Grid handle returned by EssNewGrid()</td>
</tr>
<tr>
<td>pData</td>
<td>ESS_PDATA_T</td>
<td>Pointer to the ESSG_DATA_T structure of the cell</td>
</tr>
<tr>
<td>pIsDrillable</td>
<td>ESS_PBOOL_T</td>
<td>True, if the cell is associated with a drill-through URL; False otherwise</td>
</tr>
</tbody>
</table>

**Return Value**

- If successful, sets `pIsDrillable` accordingly.
- If unsuccessful, returns an error code.
Example

```c
#define ESSG_OP_GET_DRILLTHRU_URLS 41

ESSG_STS_T sts = EssGInit(&InitStruct, &Handle);
sts = EssGNewGrid(Handle, &hGrid);
sts = EssGConnect(hGrid, Server, UserName, Password, Application, Database, ulOptions);
sts = EssGSetGridOption(hGrid, ESSG_OP_GET_DRILLTHRU_URLS, (ESSG_PVOID_T)(ESSG_TRUE));

ppDataIn = BuildQuery(&rRangeDataIn);
sts = EssGBeginRetrieve(hGrid, ESSG_RET_RETRIEVE);
sts = EssGSendRows(hGrid, &rRangeDataIn, ppDataIn);
sts = EssGPerformOperation(hGrid, 0);

/*To retrieve the cell drillable property of a cell*/

EssGGetIsCellDrillable(hGrid, &(cells[ulRow][ulCol]), &bIsDrillable);
if (bIsDrillable)
    printf("bIsDrillable: true");
else
    printf("bIsDrillable: false");
```

**EssGGetLinkedPartitionDesc**

Retrieves the description for a linked partition. You specify the partition with a unique handle returned by an *EssGGetCellLinkResults()* function call.

**Syntax**

```c
ESSG_FUNC_M EssGGetLinkedPartitionDesc (hGrid, hLRO, pConnectInfo);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hGrid;</td>
<td>ESSG_HGRID_T</td>
<td>Grid handle returned by <em>EssGNewGrid()</em></td>
</tr>
<tr>
<td>hLRO;</td>
<td>ESSG_HLRO_T</td>
<td>Handle to the linked partition (returned in a DRILLDATA structure by the <em>EssGGetCellLinkResults()</em> function). The handle must specify a linked object of type ESSG_PARTITIONTYPE.</td>
</tr>
<tr>
<td>pConnectInfo;</td>
<td>“ESSG_CONNECTINFO_T” on page 1006</td>
<td>Returns connection information for the linked partition.</td>
</tr>
</tbody>
</table>

**Notes**

To retrieve descriptions for linked objects that are not partitions, use *EssGGetLRODesc*.

**Return Value**

If successful, returns ESSG_STS_NOERR.

**See Also**

- “Using the C Grid API Functions” on page 994
- “C Grid API Structures” on page 1006
- *EssGBeginRemoveOnly*
EssGGetLRO

Retrieves a LRO from an Essbase database.

**Syntax**

```c
ESSG_FUNC_M EssGGetLRO (hGrid, hLRO, szTargetFile, ulOption);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hGrid</td>
<td>ESSG_HGRID_T</td>
<td>Grid handle returned by EssGNewGrid().</td>
</tr>
<tr>
<td>hLRO</td>
<td>ESSG_HLRO_T</td>
<td>Handle to the linked object (returned in a DRILLDATA structure by the EssGGetCellLinkResults() function).</td>
</tr>
<tr>
<td>szTargetFile</td>
<td>ESSG_STR_T</td>
<td>The name of the target file, including path, into which the object is retrieved.</td>
</tr>
<tr>
<td>ulOption</td>
<td>ESSG_ULONG_T</td>
<td>Option specifying whether to retrieve the object, its catalog entry, or both.</td>
</tr>
</tbody>
</table>

**Notes**

To retrieve a cell note, use EssGGetLRODesc. EssGGetLRO does not retrieve cell note information.

**Return Value**

If successful, returns ESG_STS_NOERR.

**See Also**

- “Using the C Grid API Functions” on page 994
- “C Grid API Structures” on page 1006
- EssGBeginRemoveOnly
Retrieves the description information for a linked object. You specify the object with a unique handle returned by an `EssGGetCellLinkResults()` function call.

**Syntax**

```
ESSG_FUNC_M EssGGetLRODesc (hGrid, hLRO, pLroDesc);
```

**Parameter** | **Data Type** | **Description**
--- | --- | ---
`hGrid` | ESSG_HGRID_T | Grid handle returned by `EssGNewGrid()`.
`hLRO` | ESSG_HLRO_T | Handle to the linked partition (returned in a DRILLDATA structure by the `EssGGetCellLinkResults()` function). The handle can specify a linked object of any type except `ESSG_PARTITIONTYPE`.
`pLroDesc` | ESSG_LPLRODESC_T | Returns an LRO description structure containing information about the specified object.

**Notes**

To retrieve descriptions for partitions (object type `ESSG_PARTITIONTYPE`), use `EssGGetLinkedPartitionDesc`.

**Return Value**

If successful, returns `ESSG_STS_NOERR`.

**See Also**

- “Using the C Grid API Functions” on page 994
- “C Grid API Structures” on page 1006
- `EssGGetCellLinkResults`
- `EssGBeginCreateLRO`
- `EssGBeginDeleteLROs`
- `EssGDeleteLRO`
EssGGetMemberInfo

Returns member relationship information from within one dimension.

Syntax

```c
ESSG_FUNC_M EssGGetMemberInfo (hGrid, pszMbrName, sAction, bAliases, pulMembers, *ppszMembers);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hGrid;</td>
<td>ESSG_HGRID_T</td>
<td>Handle passed back from EssGNewGrid.</td>
</tr>
<tr>
<td>pszMbrName;</td>
<td>ESSG_STR_T</td>
<td>Name of the member for which relationship information will be obtained.</td>
</tr>
<tr>
<td>sAction;</td>
<td>ESSG_SHORT_T</td>
<td>Number indicating what type of relationship information will be returned.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The following values are valid for this parameter and are mutually exclusive:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESSG_NEXLEVEL Children</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESSG_ALLLEVELS All members</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESSG_BOTTOMLEVEL Bottom level</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESSG_SIBLEVEL Sibling level</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESSG_SAMELEVEL Same level</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESSG_SAMEGENERATION Same generation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESSG_CALCLEVEL Calculation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESSG_PARENTLEVEL Parent of member</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESSG_TOPLEVEL Dimension member belongs to</td>
</tr>
<tr>
<td>bAliases;</td>
<td>ESSG_BOOL_T</td>
<td>Indicates whether alias names will be returned.</td>
</tr>
<tr>
<td>pulMembers;</td>
<td>ESSG_PULONG_T</td>
<td>Count of members being returned.</td>
</tr>
<tr>
<td>*ppszMembers;</td>
<td>ESSG_PSTR_T</td>
<td>Pointer to a one dimensional array of size pulMembers of members returned from</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the server. The API allocates this memory and should be freed by the caller.</td>
</tr>
</tbody>
</table>

Notes

- `pszMbrName` cannot be null.
- Free the `ppszMembers` parameter using EssGFreeMemberInfo.
Return Value

If successful, returns ESSG_STS_NOERR.

Access

None.

Example

```c
ESSG_VOID_T ESSG_GetMemberInfo(ESSG_HGRID_T hGrid)
{
  ESSG_FUNC_M  sts = ESS_STS_NOERR;
  ESSG_STR_T   pszMbrName;
  ESSG_SHORT_T sAction;
  ESSG_BOOL_T  bAliases;
  ESSG ULONG_T ulMembers, ind;
  ESSG PSTR_T  pszMembers;
  char tmp[5] = "Year";

  pszMbrName = tmp;
  sAction = ESSG_NEXLEVEL;
  bAliases = ESSG_FALSE;

  /* connect the grid to a database on the server */
  sts = EssGConnect(hGrid, "Rainbow", "Admin",
                   "Password", "Demo", "Basic",
                   ESSG_CONNECT_NODIALOG);

  /* get member information */
  if(sts == 0)
    sts = EssGGetMemberInfo(hGrid,pszMbrName, sAction, bAliases,
                            &ulMembers, &pszMembers);

  if (sts == 0)
    { 
      printf("\nNext Level of %s:\n", pszMbrName);
      for (ind = 0; ind < ulMembers; ind++)
        printf("\t%s\n", *(pszMembers + ind));
      EssGFreeMemberInfo(hGrid, ulMembers, pszMembers);
    }
  if(!sts)
    sts = EssGDisconnect(hGrid, 0);
}
```

See Also

- “Using the C Grid API Functions” on page 994
- “C Grid API Structures” on page 1006

**EssGGetResults**

Retrieves information about the data returned after an operation has been completed (EssGBeginXxx, EssGSendRows, EssGPerformOperation).
Syntax

`ESSG_FUNC_M EssGGetResults (hGrid, ulOptions, pRangeOut, pState);`

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hGrid</td>
<td>ESSG_HGRID_T</td>
<td>Handle passed back from <code>EssGNewGrid</code>.</td>
</tr>
<tr>
<td>ulOptions</td>
<td>ESSG_ULONG_T</td>
<td>Reserved for future use. Should be set to zero.</td>
</tr>
<tr>
<td>pRangeOut</td>
<td>&quot;ESSG_RANGE_T&quot; on page 1012</td>
<td>Describes the extent of the data returned from the server. This parameter describes the total amount of data that will be returned. The caller can break up the retrieval with multiple calls to <code>EssGGetRows</code>.</td>
</tr>
<tr>
<td>pState</td>
<td>ESSG_PUSHORT_T</td>
<td>Variable for the return of the state of the operation. This can be one of the following values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESSG_STATE_DONE Operation complete</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESSG_STATE_INPROGRESS The operation is in progress</td>
</tr>
</tbody>
</table>

Notes

After this call is made and the `pState` variable contains `ESSG_STATE_DONE`, the caller should call `EssGGetRows` to retrieve the actual data from the server.

Return Value

If successful, returns `ESSG_STS_NOERR`.

Access

None.

Example

```
sts = EssGGetResults(hGrid, 0, &rDataRangeOut, &usState);
```

See an example that uses this code in the `EssGBeginRetrieve` Example section.

See Also

- “Using the C Grid API Functions” on page 994
- “C Grid API Structures” on page 1006

**EssGGetRows**

Retrieves data after an operation has been completed (`EssGBegin`, `EssGSendRows`, `EssGPerformOperation`, `EssGGetResults`).

Syntax

`ESSG_FUNC_M EssGGetRows (hGrid, ulOptions, pRangeRequested, pRangeOut, pppDataOut);`
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hGrid</td>
<td>ESSG_HGRID_T</td>
<td>Handle passed back from EssGNewGrid.</td>
</tr>
<tr>
<td>ulOptions</td>
<td>ESSG_ULONG_T</td>
<td>Reserved for future use. Should be set to zero.</td>
</tr>
<tr>
<td>pRangeRequested</td>
<td>“ESSG_RANGE_T” on page 1012</td>
<td>Describes the extent of the data requested. This can be less than or equal to the number of rows and columns returned from the EssGGetResults call.</td>
</tr>
<tr>
<td>pRangeOut</td>
<td>“ESSG_RANGE_T” on page 1012</td>
<td>Describes the extent of the data returned.</td>
</tr>
<tr>
<td>*pppDataOut</td>
<td>“ESSG_DATA_T” on page 1006</td>
<td>The address of a two-dimensional array of data. The memory for this array is allocated by the API and should be freed by the caller using EssGFreeRows.</td>
</tr>
</tbody>
</table>

**Notes**
- You can make multiple calls to EssGGetRows, but the pRangeRequested->ulStartRow in each subsequent call must be greater than the last row received.
- The pRangeRequested variable should define the rows desired to be returned. If multiple buffers of data are being returned, each subsequent call to EssGGetRows should update the rows in the pRangeRequested parameter.
- If the caller requests rows where some of the requested rows are valid, while others are out of range, the valid rows will be filled in. The invalid rows remain undefined.
- If the caller requests rows that are completely out of range from the information that is available, an error is returned.

**Return Value**
If successful, returns ESSG_STS_NOERR.

**Access**
None.

**Example**
```
sts = EssGGetRows(hGrid, 0, &rDataRangeOut, &rDataRangeOut, &ppDataOut);
```
See an example that uses this code in the **EssGBeginRetrieve** Example section.

**See Also**
- “Using the C Grid API Functions” on page 994
- “C Grid API Structures” on page 1006

**EssGGetSmartlistforCell**

Returns the name of SmartList (Text List) object associated with a cell when the cell type is ESSG_DT_SMARTLIST.
An Essbase database can have multiple TextList objects and members.

This API call lets you identify which TextList object a cell is associated with.

You do not have to free returned pointers, as this is managed by the API.

As the grid is stateless, names returned are valid until you perform an EssGEndOperation.

**Syntax**

```c
ESS_FUNC_M EssGGetSmartlistforCell (hGrid, pData, *pSmartlistname)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hGrid</td>
<td>ESSG_HGRID_T</td>
<td>Grid handle</td>
</tr>
<tr>
<td>pData</td>
<td>ESSG_PDATA_T</td>
<td>Pointer to the cell ESSG_DATA_T structure of the cell.</td>
</tr>
<tr>
<td>*pSmartlistname</td>
<td>ESSG_STR_T</td>
<td>Pointer to name of the TextList object the cell is associated with</td>
</tr>
</tbody>
</table>

**Return Value**

- 0—If successful
- Error number—If unsuccessful

**See Also**

- EssOt1FreeSmartListInfo
- EssOt1GetMemberSmartList
- EssOt1GetSmartListInfo
- EssOt1PutSmartList

**EssGInit**

Initializes the Grid API.

**Syntax**

```c
ESSG_FUNC_M EssGInit (pInitStruct, pHandle);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pInitStruct</td>
<td>&quot;ESSG_INIT_T&quot; on page 1011</td>
<td>Pointer to a structure containing useful information for the EGAPI.</td>
</tr>
<tr>
<td>pHandle</td>
<td>ESSG_PHANDLE_T</td>
<td>Pointer to the handle sent back from the EGAPI.</td>
</tr>
</tbody>
</table>

**Notes**

- Required before calling all other EGAPI functions except for EssGVersion.
- Make this call only once at the beginning of a session.
- This function returns a handle, which you must pass to EssGNewGrid for each grid being used, and to any other EGAPI call requiring a non-grid specific handle.
- A thread may require its own handle (pHandle) to avoid overwriting another thread's networking status information.
**Return Value**

If successful, returns ESSG_STS_NOERR.

**Access**

None.

**Example**

```c
#include <essapin.h>
#include <essgapin.h>

ESSG_FUNC_M     sts = ESS_STS_NOERR;
ESSG_INIT_T     InitStruct;
ESSG_HANDLE_T   Handle;

InitStruct.ulVersion = ESSG_VERSION;
InitStruct.ulMaxRows = 1000;
InitStruct.ulMaxColumns = 200;
InitStruct.pfnMessageFunc = ESS_NULL;
InitStruct.pUserdata = ESS_NULL;

sts = EssGInit(&InitStruct, &Handle);
```

**See Also**

- “Using the C Grid API Functions” on page 994
- “C Grid API Structures” on page 1006

### EssGLoginSetPass

Connects a grid to an Essbase database, and changes the user password.

**Syntax**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hGrid;</td>
<td>ESSG_HGRID_T</td>
<td>Handle from EssGNewGrid()</td>
</tr>
<tr>
<td>Server;</td>
<td>ESSG_SERVER_T</td>
<td>Name of a valid server</td>
</tr>
<tr>
<td>Username;</td>
<td>ESSG_USERNAME_T</td>
<td>Name of a valid user on the server</td>
</tr>
<tr>
<td>Password;</td>
<td>ESSG_PASSWORD_T</td>
<td>User's password</td>
</tr>
<tr>
<td>NewPassword;</td>
<td>ESSG_PASSWORD_T</td>
<td>User's new password</td>
</tr>
</tbody>
</table>

**Return Value**

If successful, returns ESSG_STS_NOERR.

**Example**

```c
#include
#include
```


```c
{  
    ESSG_FUNC_M    sts = ESS_STS_NOERR;  
    ESSG_INIT_T    InitStruct;  
    ESSG_HANDLE_T   Handle;  
    ESSG_SERVER_T   Server;  
    ESSG_USERNAME_T UserName;  
    ESSG_PASSWORD_T Password;  
    ESSG_PASSWORD_T NewPassword;  
    ESSG_HGRID_T    hGrid;  

    InitStruct.ulVersion = ESSG_VERSION;  
    InitStruct.ulMaxRows = 1000;  
    InitStruct.ulMaxColumns = 200;  
    InitStruct.pfnMessageFunc = ESS_NULL;  
    InitStruct.pUserdata = ESS_NULL;  

    /* initializes EGAPI */  
    sts = EssGInit(&InitStruct, Handle);  

    /* initializes a specific grid */  
    if(!sts)  
        sts = EssGNewGrid(Handle, &hGrid);  

        strcpy(Server, "Rainbow");  
        strcpy(UserName, "Admin");  
        strcpy(Password, "Password");  
        strcpy(Password, "NewPassword");  

        /* connects the grid to a database on the server */  
        if(!sts)  
            sts = EssGLoginSetPass(hGrid, Server, UserName, Password, NewPassword);  
}
```

See Also

- “C Grid API Constants” on page 997
- “C Grid API Structures” on page 1006
- EssAutoLogin
- EssGConnect
- EssGInit
- EssInit
- EssLogout

**EssGNewGrid**

Initializes a specific grid.

**Syntax**

```c
ESSG_FUNC_M EssGNewGrid (Handle, phGrid);
```
## Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Handle; /* IN */</td>
<td>ESSG_HANDLE_T</td>
<td>Handle passed back from EssGInit.</td>
</tr>
<tr>
<td>phGrid; /* OUT */</td>
<td>ESSG_PHGRID_T</td>
<td>Pointer to the grid-specific handle sent back from the EGAPI.</td>
</tr>
</tbody>
</table>

### Notes

- This call is required prior to calling any grid-specific API.
- The handle returned should be passed to any subsequent grid-specific API call that manipulates the specific grid.
- The call should be made once for each grid that uses the Grid API.
- Each thread in a multithreaded environment must use its own handle (phGrid) to call a grid-specific API, such as EssGSendRows() or EssGBeginOperation().

### Return Value

If successful, returns ESSG_STS_NOERR.

### Access

None.

### Example

```c
#include <essapin.h>
#include <essgapin.h>

ESSG_FUNC_M     sts = ESS_STS_NOERR;
ESSG_INIT_T     InitStruct;
ESSG_HANDLE_T   Handle;
ESSG_HGRID_T    hGrid;

InitStruct.ulVersion = ESSG_VERSION;
InitStruct.ulMaxRows = 1000;
InitStruct.ulMaxColumns = 200;
InitStruct.pfnMessageFunc = ESS_NULL;
InitStruct.pUserData = ESS_NULL;

/* initializes EGAPI */
sts = EssGInit(&InitStruct, &Handle);

/* initializes a specific grid */
if(!sts)
    sts = EssGNewGrid(Handle, &hGrid);
```

### See Also

- “Using the C Grid API Functions” on page 994
- “C Grid API Structures” on page 1006
EssGPerformOperation

Performs an operation after all rows have been sent to the server using EssGBeginXxx and EssGSendRows.

Syntax

```c
ESSG_FUNC_M EssGPerformOperation (hGrid, ulOptions);
```

Parameter Data Type Description

<table>
<thead>
<tr>
<th>hGrid</th>
<th>ESSG_HGRID_T</th>
<th>Handle passed back from EssGNewGrid().</th>
</tr>
</thead>
<tbody>
<tr>
<td>ulOptions</td>
<td>ESSG_ULONG_T</td>
<td>Reserved for future use. Set to zero.</td>
</tr>
</tbody>
</table>

Notes

After this call is made, call EssGGetResults to get information about the data returned.

Return Value

If successful, returns ESSG_STS_NOERR.

Example

```c
sts = EssGPerformOperation(hGrid, 0);
```

See an example that uses this code in the EssGBeginPivot Example section.

See Also

- “Using the C Grid API Functions” on page 994
- “C Grid API Structures” on page 1006

EssGSendRows

Sends the rows to the server once an operation has been started.

Syntax

```c
ESSG_FUNC_M EssGSendRows (hGrid, pRangeIn, ppDataIn);
```

Parameter Data Type Description

<table>
<thead>
<tr>
<th>hGrid</th>
<th>ESSG_HGRID_T</th>
<th>Handle passed back from EssGNewGrid.</th>
</tr>
</thead>
<tbody>
<tr>
<td>pRangeIn</td>
<td>“ESSG_RANGE_T” on page 1012</td>
<td>Describes the extent of the data in ppDataIn.</td>
</tr>
<tr>
<td>ppDataIn</td>
<td>“ESSG_DATA_T” on page 1006</td>
<td>A two-dimensional array of cells describing the data.</td>
</tr>
</tbody>
</table>

Notes

- You can make multiple calls to EssGSendRows, but the pRangeIn->ulStartRow in each subsequent call must be greater than the last row sent in.
- The pRangeIn variable should define the rows in the grid.
If you are sending in multiple buffers of data, each subsequent call to `EssGSendRows` should update the rows in the `pRangeIn` parameter.

After all rows are sent in, you can call `EssGPerformOperation`.

### Return Value

If successful, returns `ESSG_STS_NOERR`.

### Access

None.

### Example

```c
sts = EssGSendRows(hGrid, &rDataRangeIn, ppDataIn);
```

See an example that uses this code in the `EssGBeginRetrieve` Example section.

### See Also

- “Using the C Grid API Functions” on page 994
- “C Grid API Structures” on page 1006

---

## EssGSetGridOption

Sets individual grid options.

### Syntax

```c
ESSG_FUNC_M EssGSetGridOption (hGrid, sOption, pOption);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| hGrid     | ESSG_HGRID_T     | Handle passed back from `EssGNewGrid()`.
| sOption   | ESSG_SHORT_T     | Value indicating what option is being set. For a table of valid values, see Notes. |
| pOption   | ESSG_PVOID_T     | Value of option being set cast to an ESSG_PVOID_T. |

### Notes

- You can use the ESSG_OP_USERGRIDDATA pointer to store grid-specific information that is private to the application.
- The following table lists valid options for `sOption` and the corresponding description and data type:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
<th>Data Type Expected</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESSG_OP_ALIASNAMES</td>
<td>Alias names</td>
<td>ESSG_BOOL_T</td>
<td>ESSG_FALSE</td>
</tr>
<tr>
<td>ESSG_OP_ALIASTABLE</td>
<td>Alias names table</td>
<td>ESSG_STR_T</td>
<td></td>
</tr>
<tr>
<td>ESSG_OP_DATALESS</td>
<td>Enable dataless navigation.</td>
<td>ESSG_BOOL_T</td>
<td>ESSG_FALSE</td>
</tr>
</tbody>
</table>

---

EssGSetGridOption 1093
<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
<th>Data Type Expected</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESSG_OP_DRILLLEVEL</td>
<td>Drill-level</td>
<td>ESSG_SHORT_T</td>
<td>ESSG_NEXTLEVEL</td>
</tr>
<tr>
<td>ESSG_OP_EMPTYGRIDERROR</td>
<td>If FALSE, don't issue error on queries which result in no data and return only the grid header.</td>
<td>ESSG_BOOL_T</td>
<td>ESSG_TRUE</td>
</tr>
<tr>
<td>ESSG_OP_INCSEL</td>
<td>Include selection</td>
<td>ESSG_BOOL_T</td>
<td>ESSG_FALSE</td>
</tr>
<tr>
<td>ESSG_OP_INDENT</td>
<td>Indent style</td>
<td>ESSG_SHORT_T</td>
<td>ESSG_INDENTTOTALS</td>
</tr>
<tr>
<td>ESSG_OP/latest</td>
<td>Turn on the ability to specify the latest member.</td>
<td>ESSG_BOOL_T</td>
<td>ESSG_FALSE</td>
</tr>
<tr>
<td>ESSG_OP/latestMember</td>
<td>Specify the latest member.</td>
<td>ESSG_STR_T</td>
<td>NULL</td>
</tr>
<tr>
<td>ESSG_OP_REPEATMBRNAMES</td>
<td>Repeat member names.</td>
<td>ESSG_BOOL_T</td>
<td>ESSG_FALSE</td>
</tr>
<tr>
<td>ESSG_OP_RETAINTHREAD</td>
<td>If set to TRUE, don’t disconnect from server thread at end of grid operation. May improve performance when submitting several operations in sequence.</td>
<td>ESSG_BOOL_T</td>
<td>ESSG_FALSE</td>
</tr>
<tr>
<td>ESSG_OP_SELGROUP</td>
<td>Remove Unselected Groups. Zooms on all occurrences of selected member but removes any other members from the same dimension, including the selected member itself.</td>
<td>ESSG_BOOL_T</td>
<td>ESSG_FALSE</td>
</tr>
<tr>
<td>ESSG_OP_SELONLY</td>
<td>Within Selected Group. Zooms on only the exact instance of the member that is selected.</td>
<td>ESSG_BOOL_T</td>
<td>ESSG_FALSE</td>
</tr>
<tr>
<td>ESSG_OP_SUPMISSING</td>
<td>Suppress missing rows.</td>
<td>ESSG_BOOL_T</td>
<td>ESSG_FALSE</td>
</tr>
<tr>
<td>ESSG_OP_SUPUNDER</td>
<td>Replace underscores with spaces.</td>
<td>ESSG_BOOL_T</td>
<td>ESSG_FALSE</td>
</tr>
<tr>
<td>ESSG_OP_SUPZEROS</td>
<td>Suppress zero rows.</td>
<td>ESSG_BOOL_T</td>
<td>ESSG_FALSE</td>
</tr>
<tr>
<td>ESSG_OP_UPDATEMODE</td>
<td>Update mode</td>
<td>ESSG_BOOL_T</td>
<td>ESSG_FALSE</td>
</tr>
<tr>
<td>ESSG_OP_USEBOTHFORROWDIMS</td>
<td>Use both member names and aliases for the row dimensions.</td>
<td>ESSG_BOOL_T</td>
<td>ESSG_FALSE</td>
</tr>
<tr>
<td>ESSG_OP_USERGRIDDATA</td>
<td>Pointer to user data</td>
<td>ESSG_PVOID_T</td>
<td>NULL</td>
</tr>
<tr>
<td>ESSG_OP_RETAINTHREAD</td>
<td>Retain threads</td>
<td>ESSG_BOOL_T</td>
<td>ESSG_FALSE</td>
</tr>
<tr>
<td>ESSG_OP_EMPTYGRIDERROR</td>
<td>Issue an empty grid error</td>
<td>ESSG_BOOL_T</td>
<td>ESSG_FALSE</td>
</tr>
<tr>
<td>ESSG_OP_DATALESS</td>
<td>Navigate without data</td>
<td>ESSG_BOOL_T</td>
<td>ESSG_FALSE</td>
</tr>
<tr>
<td>ESSG_OP_SPANHYBRIDANALYSIS</td>
<td>Span drill to relational source</td>
<td>ESSG_BOOL_T</td>
<td>ESSG_FALSE</td>
</tr>
</tbody>
</table>

**Return Value**

If successful, returns ESSG_STS_NOERR.
Example

```
ESSG_VOID_T ESSG_SetGridOption (ESSG_HGRID_T hGrid)
{
    ESSG_STS_T     sts = ESS_STS_NOERR;
    ESSG_SHORT_T         sOption;
    ESSG_SHORT_T         tmpShort;
    ESSG_PVOID_T         pOption;

    /* connect the grid to a database on the server */
    sts = EssGConnect(hGrid, "Rainbow", "Admin", "Password", "Demo", "Basic",
                      ESSG_CONNECT_DEFAULT);

    tmpShort = 2;
    sOption = ESSG_OP_DRILLLEVEL;
    pOption = (ESSG_PVOID_T)tmpShort;

    /* set grid option */
    if(!sts)
        sts = EssGSetGridOption(hGrid, sOption, pOption);

    if(!sts)
        EssGDisconnect(hGrid, 0);
}
```

See Also

- “Using the C Grid API Functions” on page 994
- “C Grid API Structures” on page 1006

**EssGSetGridPerspective**

This function sets perspective for a grid. Perspective is similar to grid option. If the set perspective is valid, the grid context is the same.

**Syntax**

```
ESSG_FUNC_M EssGSetGridPerspective(hGrid, sAttrdim, sPerspectiveType,
                                    pPerspectiveString)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hGrid</td>
<td>ESSG_HGRID_T</td>
<td>Handle passed back from EssGNewGrid().</td>
</tr>
<tr>
<td>sAttrdim</td>
<td>ESSG_STR_T</td>
<td>Attribute dimension name for which perspective has to set</td>
</tr>
<tr>
<td>sPerspectiveType</td>
<td>ESSG_SHORT_T</td>
<td>Type of perspective. See “Grid Perspective Types” on page 1003.</td>
</tr>
</tbody>
</table>
EssGSetPath

Sets the ESSBASEPATH environment variable for the current process.

Syntax

```
ESSG_FUNC_M EssGSetPath (pszPath);
```

Parameter | Data Type     | Description
----------|---------------|-------------
pszPath;   | ESSG_STR_T    | Pointer to the string describing the ESSBASEPATH environment variable

Notes

- Call `EssGSetPath()` before calling `EssGInit()`.
- `pszPath` cannot exceed 120 characters, as defined in ESSG_PATHLEN.
- `pszPath` applies only to the current process.
- Essbase DLLs must be accessible from the system path. `EssGSetPath()` does not resolve the path for the Essbase DLLs.

Return Value

- If successful, returns ESSG_STS_NOERR.
- If `pszPath` is too long, returns API_NAME_TOO_LONG (1030009).
Example

```c
ESS_STS_T
ESSG_SetPath(ESS_STR_T pszPath)
{
    ESS_STS_T  sts
    ESSG_STR_T pszPath = "C:\Hyperion\products\Essbase";
    sts = EssGSetPath (pszPath);
    return sts;
}
```

See Also

- “Using the C Grid API Functions” on page 994
- “C Grid API Structures” on page 1006

**EssGTerm**

Terminates the Grid API.

**Syntax**

```c
ESSG_FUNC_M EssGTerm (Handle);
```

**Parameter** | **Data Type** | **Description**
---|---|---
Handle | ESSG_HANDLE_T | Handle to instance of the EGAPI.

**Notes**

- This call is required.
- Signifies termination of use of the Grid API.
- This call should be made only once per session and then only at the end of that session.

**Return Value**

If successful, returns ESSG_STS_NOERR.

**Access**

None.

**Example**

```c
#include <essapin.h>
#include <essgapin.h>

ESSG_FUNC_M     sts = ESS_STS_NOERR;
ESSG_INIT_T     InitStruct;
ESSG_HANDLE_T   Handle;

InitStruct.ulVersion = ESSG_VERSION;
InitStruct.ulMaxRows = 1000;
InitStruct.ulMaxColumns = 200;
InitStruct.pfnMessageFunc = ESS_NULL;
InitStruct.pUserdata = ESS_NULL;
```
/ * initialize EGAPI */
sts = EssGInit(&InitStruct, &Handle);

/* terminate the EGAPI */
if(!sts)
  sts = EssGTerm(Handle);

See Also
- “Using the C Grid API Functions” on page 994
- “C Grid API Structures” on page 1006

**EssGUnlock**

Unlocks any blocks that were locked at the server.

**Syntax**

ESSG_FUNC_M EssGUnlock(hGrid, ulOptions);

**Parameter | Data Type | Description**

| hGrid | ESSG_HGRID_T | Handle passed back from EssGNewGrid.
| ulOptions | ESSG_ULONG_T | Reserved for future use. Should be set to zero.

**Return Value**

If successful, returns ESSG_STS_NOERR.

**Access**

None.

**Example**

ESSG_VOID_T ESSG_Unlock(ESSG_HGRID_T hGrid)

{  
  ESSG_FUNC_M   sts = ESS_STS_NOERR;
  ESSG_ULONG_T   ulOptions = 0;

  /* connect the grid to a database on the server */
  sts = EssGConnect(hGrid, "Rainbow", "Admin", "Password", "Demo", "Basic",
                   ESSG_CONNECT_NODIALOG);

  /* unlock the locked blocks at server */
  if(!sts)
    sts = EssGUnlock(hGrid, ulOptions);

  if(!sts)
    EssGDisconnect(hGrid, 0);
}

1098 C Grid API Function Reference
EssGUpdateLRO

Updates the description and contents of a linked object.

Syntax

```
ESSG_FUNC_M EssGUpdateLRO (hGrid, hLRO, pLroDesc, ulOption);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hGrid;</td>
<td>ESSG_HGRID_T</td>
<td>Grid handle returned by EssGNewGrid().</td>
</tr>
<tr>
<td>hLRO;</td>
<td>ESSG_HLRO_T</td>
<td>Handle to the linked object (returned in a DRILLDATA structure by the EssGGetCellLinkResults() function).</td>
</tr>
<tr>
<td>pLroDesc;</td>
<td>“ESSG_LRODESC_T” on page 1011</td>
<td>A structure containing information about the LRO to be updated.</td>
</tr>
<tr>
<td>ulOption;</td>
<td>ESSG_ULONG_T</td>
<td>Option specifying whether to store the object on the server. Use ESS_STORE_OBJECT_API to store winapp and URL objects on the server. Use ESS_NOSTORE_OBJECT_API to store cell notes off the server (and in the index file).</td>
</tr>
</tbody>
</table>

Return Value

If successful, returns ESSG_STS_NOERR.

See Also

- “Using the C Grid API Functions” on page 994
- “C Grid API Structures” on page 1006
- EssGBeginCreateLRO
- EssGBeginDeleteLROs
- EssGDeleteLRO
- EssGGetLRODesc

EssGVersion

Returns the version number for the API.

Syntax

```
ESSG_FUNC_M EssGVersion (pulVersion);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pulVersion</td>
<td>ESSG_PULONG_T</td>
<td>Pointer to the current version number of the Grid API.</td>
</tr>
</tbody>
</table>
Notes

- The number is incremented whenever changes requiring either a recompile or relink by a client occur.
- You do not need to initialize the Grid API before you use this function.

Return Value

If successful, returns ESSG_STS_NOERR.

Access

None.

Example

#include <essapin.h>
#include <essgapin.h>

ESSG_FUNC_M     sts = ESS_STS_NOERR;
ESSG_ULONG_T    ulVersion;

/* get version number for the API */
sts = EssGVersion(&ulVersion);

See Also

- “Using the C Grid API Functions” on page 994
- “C Grid API Structures” on page 1006
C Grid API Example

This example illustrates the steps needed to perform a basic retrieval. The following grid shows a five dimensional template with one datapoint illustrated.

<table>
<thead>
<tr>
<th>Actual</th>
<th>Year</th>
<th>Sales</th>
<th>Product</th>
<th>Market</th>
</tr>
</thead>
</table>

The following code fragment shows how the data structures are setup and the function calls that are needed to perform the retrieval.

```c
/* This function allocates the necessary data to send to the server */

ESSG_PPDATA_T AllocTwoDims(ESSG_ULONG_T ulRows, ESSG_ULONG_T ulCols)
{
    ESSG_PPDATA_T ppTemp;
    ESSG_ULONG_T ulIndex;

    if(ulRows)
    ppTemp = (ESSG_PPDATA_T) malloc(sizeof(ESSG_DATA_T*) * ulRows);
    if(ppTemp == NULL)
        return ppTemp;

    memset(ppTemp, 0, (sizeof(ESSG_PDATA_T) * ulRows));

    for (ulIndex = 0; ulIndex < ulRows; ulIndex++)
    {
        ppTemp[ulIndex] = (ESSG_PDATA_T)malloc(sizeof(ESSG_DATA_T) * ulCols);
        if(ppTemp[ulIndex])
            memset(ppTemp[ulIndex], 0, (sizeof(ESSG_DATA_T) * ulCols));
    }
```
return ppTemp;
}

/* This function frees the memory allocated by AllocTwoDims */
void FreeTwoDim(ESSG_PPDATA_T ppDataToFree, ESS_ULONG_T ulRows)
{
    ESS_ULONG_T ulIndex;
    for (ulIndex = 0; ulIndex < ulRows; ulIndex++)
    {
        if(ppDataToFree[ulIndex]->usType == ESSG_DT_STRING)
        {
            free(ppDataToFree[ulIndex]->Value.pszStr);
        }
        free(ppDataToFree[ulIndex]);
    }
    free(ppDataToFree);
}

/* This function builds a table based on the above grid. */
/* Note: The items in the grid are hard coded. */
ESSG_PPDATA_T BuildTable(ESSG_PRANGE_T pRange)
{
    ESSG_PPDATA_T   ppTable;
    ESS_ULONG_T     ulRow, ulCol;

    /* Your code would probably not be hard-coded here... */
    pRange->ulRowStart      = 0;
    pRange->ulColumnStart   = 0;
    pRange->ulNumRows       = 2;
    pRange->ulNumColumns    = 5;
    ppTable = AllocTwoDims(2, 5);

    /* ROW 1 */
    ppTable[0][0].usType = ESSG_DT_BLANK;
    ppTable[0][1].usType = ESSG_DT_BLANK;
    ppTable[0][2].usType = ESSG_DT_STRING;
    /* Some compilers allow you to specify \p to indicate */
    /* the length of the string */
    ppTable[0][2].Value.pszStr = "\pYear";
    ppTable[0][3].usType = ESSG_DT_STRING;
    ppTable[0][3].Value.pszStr = "\pProduct";
    ppTable[0][4].usType = ESSG_DT_STRING;
    ppTable[0][4].Value.pszStr = "\pMarket";

    /* ROW 2 */
    ppTable[1][0].usType = ESSG_DT_STRING;
    ppTable[1][0].Value.pszStr = "\pActual";
    ppTable[1][1].usType = ESSG_DT_STRING;
    ppTable[1][1].Value.pszStr = "\pSales";
    ppTable[1][2].usType = ESSG_DT_DOUBLE;
    ppTable[1][2].dblData = 123.45;
    ppTable[1][3].usType = ESSG_DT_BLANK;
    ppTable[1][4].usType = ESSG_DT_BLANK;
return (ppTable);
}

/* This function makes the necessary calls to the */
/* EGAPI to perform a basic retrieval. */
/* NOTE: This example does not show the */
/* initialization of the EGAPI or the grid. */
/* Also, the hGrid is assumed to be external. */

void CallEGAPI(void)
{
    ESSG_PPDATA_T   ppDataIn,
    ESSG_PPDATA_T   ppDataOut;
    ESSG_RANGE_T    rRangeDataIn, rRangeDataOut;
    ESSG_STS_T      sts;
    ESSG_ULONG_T    ulRow, ulCol;
    ESSG_USHORT_T   usState;

    /* Connect the grid to a database on the server */
    sts = EssGConnect(hGrid, "Server", "User", "Password",
                      "App", "Db", ESSG_CONNECT_DEFAULT);
    if (sts == 0)
    {
        ppDataIn = BuildTable(rRangeDataIn);
        /* Start the retrieve operation */
        sts = EssGBeginRetrieve(hGrid, ESSG_RET_RETRIEVE);
    }
    if (sts == 0)
    {
        /* Send the entire grid to define the query */
        sts = EssGSendRows(hGrid, rRangeDataIn, ppDataIn);
    }
    if (sts == 0)
    {
        /* We're done sending rows, perform the retrieval */
        sts = EssGPerformOperation(hGrid, 0);
        /* Free the data we built */
        FreeTwoDim(ppDataIn, rRangeDataIn.ulNumRows);
    }
    if (sts == 0)
    {
        /* Determine the results of the retrieve and how much data */
        /* is being returned. */
        sts = EssGGetResults(hGrid, 0, rRangeDataOut, usState);
    }
    if (sts == 0)
    {
        /* Get all of the data */
        sts = EssGGetRows(hGrid, 0, rRangeDataOut, rRangeDataOut, ppDataOut);
    }
    if (sts == 0)
    {
        /* Iterate though the data ... */
        /* First the rows */
        for (ulRow = rRangeDataOut.ulRowStart;
ulRow < rRangeDataOut.ulNumRows;
ulRow++
{
    /* Then the columns */
    for (ulCol = rRangeDataOut.ulColumnStart;
        ulCol < rRangeDataOut.ulNumColumns;
        ulCol++)
    {
        /* Here's a cell ... just render it. */
        switch (ppDataOut[ulRow][ulCol].usType)
        {
            case (ESSG_DT_STRING):
                DisplayString(ppDataOut[ulRow]
                [ulCol].Value.pszStr);
                break;
            case (ESSG_DT_LONG):
                DisplayValue(ppDataOut[ulRow]
                [ulCol].Value.lData);
                break;
            case (ESSG_DT_DOUBLE):
                DisplayValue(ppDataOut[ulRow]
                [ulCol].Value.dblData);
                break;
            case (ESSG_DT_BLANK):
                DisplayBlank();
                break;
            case (ESSG_DT_MISSING):
                DisplayMissing();
                break;
            case (ESSG_DT_ZERO):
                DisplayValue(0);
                break;
            case (ESSG_DT_NOACCESS):
                DisplayNoAccess();
                break;
            case (ESSG_DT_MEMBEREX):
                DisplayString(ppDataOut[ulRow]
                [ulCol].Value.pszStr+1);
                break;
            default:
                DisplayOops();
                break;
        }
    }
    /* Tell the API we don't care about this request any more */
    EssGEndOperation(hGrid, 0);
    /* Free the data returned */
    EssGFreeRows(hGrid, rRangeDataOut, ppDataOut);
}
/* Disconnect if you wish */
EssGDisconnect(hGrid, 0);}
C Grid API Drill-Through Example

```c
void main(int argc, char *argv[]) {
    ESSG_STS_T      sts = ESS_STS_NOERR;
    ESSG_HGRID_T    hGrid;
    ESSG_HANDLE_T   Handle;
    ESSG_INIT_T     InitStruct;

    /* BEGIN: initialize grid handle and create a new grid */
    InitStruct.ulVersion      = ESSG_VERSION;
    InitStruct.ulMaxRows      = 1000;
    InitStruct.ulMaxColumns   = 200;
    InitStruct.pfnMessageFunc = ESS_NULL;
    InitStruct.pUserData      = ESS_NULL;
    sts = EssGInit(&InitStruct, Handle);
    if (sts != ESS_STS_NOERR)
        return;
    sts = EssGNewGrid(Handle, hGrid);
    if (sts != ESS_STS_NOERR)
        return;
    /* END: initialize grid handle and create a new grid */

    ESSG_DTTest(Handle, hGrid);
    sts = EssGTerm(Handle);
}

void ESSG_DTTest(ESSG_HANDLE_T  Handle, ESSG_HGRID_T hGrid)
{
    ESSG_STS_T        errsts,
    sts            = ESS_STS_NOERR;
    ESSG_HLRO_T       hLRO           = 0;
    ESSG_PPDATA_T     ppDataIn;
    /* ESSG_PPDATA_T  ppDataOut; */
    ESSG_RANGE_T      rDataRangeIn,
    rDataRangeOut;
    ESSG_USHORT_T     usCells;
    ESSG_USHORT_T     usState        = 0;
    ESSG_RANGE_T      Range;
    ESSG_PDTHINST_T   pDTInst;
    ESSG_STR_T        ErrMesg;
    ESSG_ULONG_T      ErrSize        = 255;
    memset(&rDataRangeOut, 0, sizeof(ESSG_RANGE_T));
    ErrMesg = malloc(255);

    /* connect the grid to a database on the server */
    sts = EssGConnect(hGrid, server, user, pwd, app, db, ESSG_CONNECT_DEFAULT);
    if (sts == ESS_STS_NOERR)
    {
        ppDataIn = BuildTableForDrillThru (&rDataRangeIn);
    }

    /* END: connect the grid to a database on the server */
    // MORE CODE...
}
```
DisplayOutput(ppDataIn, rDataRangeIn);

usCells = 1;
Range.ulRowStart = 1;
Range.ulColumnStart = 6;
Range.ulNumRows = 1;
Range.ulNumColumns = 1;
sts = EssGBeginDrillOrLink(hGrid, usCells, &Range, ESSG_OPT_ZOOM);
}

if(sts == ESS_STS_NOERR)
/* send the entire grid to define the query */
sts = EssGSendRows(hGrid, &rDataRangeIn, ppDataIn);

if(sts == ESS_STS_NOERR)
{
/* perform the drillorlink operation */
sts = EssGPerformOperation(hGrid, 0);

/* free the built data */
FreeTwoDim(ppDataIn, rDataRangeIn.ulNumRows);
}

if (sts == ESS_STS_NOERR)
sts = EssGDTRequestDrillThrough(hGrid, usCells, &Range, &pDTInst);

if (sts == ESS_STS_NOERR)
{
/* Get the DT Info corresponding to the DT handle */
sts = ESSGDTGetInfo(pDTInst);

/* Set the password info for executing the drill through report */
sts = ESSGDTSetInfo(pDTInst);

/* determine the list of reports associated with the data cell range. */
sts = ESSGDTListReports(pDTInst);

/* Execute the report. Using index 0 for now as we have only one report */
sts = ESSGDTExecuteReport(pDTInst, 0);
if (sts )   /* Error Condition print error mesg */
    errsts = EssDTAPIGetError(pDTInst, &sts, ErrMesg, ErrSize);

/* Get the headers for the report associated with the data cell range. */
sts = ESSGDTGetHeader(pDTInst);
if (sts )   /* Error Condition print error mesg */
    EssDTAPIGetError(pDTInst, &sts, ErrMesg, ErrSize);

/* Get the data for the report associated with the data cell range. */
sts = ESSGDTGetData(pDTInst);
if (sts )   /* Error Condition print error mesg */
    EssDTAPIGetError(pDTInst, &sts, ErrMesg, ErrSize);
}
sts = EssGDTEndDrillThrough(pDTInst);
}

See the following functions for more information on Drill-Through:

- EssGDTConnect
- EssGDTEndDrillThrough
- EssGDTExecuteReport
- EssGDTGetData
- EssGDTGetHeader
- EssGDTGetInfo
- EssGDTListReports
- EssGDTRequestDrillThrough
- EssGDTSetInfo

**ESSG_OP_MEMBERANDUNIQUENAME Example**

The following example illustrates the use of the Grid API constant `ESSG_OP_MEMBERANDUNIQUENAME`.

```c
ESSG_VOID_T ESSG_BeginZoomIn(ESSG_HGRID_T hGrid)
{
    ESSG_STS_T      sts = ESS_STS_NOERR;
    ESSG_DATA_T    **ppDataIn;
    ESSG_DATA_T    **ppDataOut;
    ESSG_RANGE_T    rDataRangeIn, rDataRangeOut;
    ESSG_ULONG_T    ulOptions;
    ESSG_USHORT_T   usCells;
    ESSG_RANGE_T    pZoomCells;
    ESSG_USHORT_T   usState;
    ESSG_USHORT_T   usMember2Len, usKey2Len;
    ESSG_SHORT_T    sOption, sOptionGet;
    ESSG_PVOID_T    pOption, pOptionGet;
    ESSG_STR_T     pMember, pKey, pOutStr;
    ESSG_STR_T     pMember2, pKey2;

    /* connect the grid to a database on the server */
    sts = EssGConnect(hGrid, server, "essexer", pwd, app, db, ESSG_CONNECT_NODIALOG);

    /* set grid option*/
    tmpShort = ESSG_TRUE;
    sOption = ESSG_OP_MEMBERANDUNIQUENAME ;
    pOption = (ESSG_PVOID_T)tmpShort; // pOption holds the actual value not a pointer

    sts = EssGSetGridOption(hGrid, sOption, pOption);
    printf("EssGSetGridOption  sts  %ld\n",sts);
    sOptionGet = ESSG_OP_MEMBERANDUNIQUENAME ;
```

**ESSG_OP_MEMBERANDUNIQUENAME Example**

The following example illustrates the use of the Grid API constant `ESSG_OP_MEMBERANDUNIQUENAME`.

```c
ESSG_VOID_T ESSG_BeginZoomIn(ESSG_HGRID_T hGrid)
{
    ESSG_STS_T      sts = ESS_STS_NOERR;
    ESSG_DATA_T    **ppDataIn;
    ESSG_DATA_T    **ppDataOut;
    ESSG_RANGE_T    rDataRangeIn, rDataRangeOut;
    ESSG_ULONG_T    ulOptions;
    ESSG_USHORT_T   usCells;
    ESSG_RANGE_T    pZoomCells;
    ESSG_USHORT_T   usState;
    ESSG_USHORT_T   usMember2Len, usKey2Len;
    ESSG_SHORT_T    sOption, sOptionGet;
    ESSG_PVOID_T    pOption, pOptionGet;
    ESSG_STR_T     pMember, pKey, pOutStr;
    ESSG_STR_T     pMember2, pKey2;

    /* connect the grid to a database on the server */
    sts = EssGConnect(hGrid, server, "essexer", pwd, app, db, ESSG_CONNECT_NODIALOG);

    /* set grid option*/
    tmpShort = ESSG_TRUE;
    sOption = ESSG_OP_MEMBERANDUNIQUENAME ;
    pOption = (ESSG_PVOID_T)tmpShort; // pOption holds the actual value not a pointer

    sts = EssGSetGridOption(hGrid, sOption, pOption);
    printf("EssGSetGridOption  sts  %ld\n",sts);
    sOptionGet = ESSG_OP_MEMBERANDUNIQUENAME ;
```
pOptionGet = &tmpShortGet;
    if(!sts)
    {
        sts = EssGGetGridOption(hGrid, sOptionGet, pOptionGet);
        printf("EssGGetGridOption  sts  %ld\n",sts);
        printf("EssGSetGridOption set ESSG_OP_MEMBERANDUNIQUE_NAME TO %d\n",
            (int)tmpShortGet);
    }

    if(sts == 0)
    {
        ppDataIn = BuildTable(&rDataRangeIn);

        ulOptions = ESSG_ZOOM_DOWN | ESSG_NEXTLEVEL;

        pZoomCells.ulRowStart = 0;
        pZoomCells.ulColumnStart = 2;
        pZoomCells.ulNumRows = 1;
        pZoomCells.ulNumColumns = 1;
        usCells = 1;

        /* start the zoom in operation */
        sts = EssGBeginZoomIn(hGrid, usCells, &pZoomCells, ulOptions);
        printf("EssGBeginZoomIn  sts:  %ld\n",sts);
    }

    //Display Input
    DisplayOutput(ppDataIn, rDataRangeIn);
    printf("\n\n");
    if(sts == 0)
    /* send the entire grid to define the query */
    sts = EssGSendRows(hGrid, &rDataRangeIn, ppDataIn);

    if(sts == 0)
    {
        /* perform the zoom-in */
        sts = EssGPerformOperation(hGrid, 0);

        /* Free the built data */
        FreeTwoDim(ppDataIn, rDataRangeIn.ulNumRows);
    }
    if (sts == 0)
    {
        /* determine the results of the zoom-in */
        sts = EssGGetResults(hGrid, 0, &rDataRangeOut, &usState);
    }
    if(sts ==0)
    {
        /* get all the data */
        sts = EssGGetRows(hGrid, 0, &rDataRangeOut, &rDataRangeOut, &ppDataOut);
    }

    if(sts == 0)
    {
        DisplayOutput(ppDataOut, rDataRangeOut);
/* Retreive member and key from cell */
sts = EssGGetFromMemberwKey (((ppDataOut[1][0]).Value).pszStr, &pMember, &pKey);
    printf("After EssGGetFromMemberwKey\n Member: %s, Key: %s \n\n", pMember+2, pKey+2);

//Member is "Qtr1", Key is "[2004].[Qtr1]", pOutStr is in the format
//nn<member-name>nn<key> - where nn is string length
usMember2Len = strlen("Qtr1");
pMember2 = malloc(usMember2Len+3);
    memset(pMember2, 0, usMember2Len+3);
usKey2Len = strlen("[2004].[Qtr1]" );
pKey2 = malloc(usKey2Len+3);
    memset(pKey2, 0, usKey2Len+3);
memcpy(pMember2, &usMember2Len, 2);
memcpy(pMember2+2, "Qtr1", usMember2Len );
memcpy(pKey2, &usKey2Len, 2);
memcpy(pKey2+2, "[2004].[Qtr1]", usKey2Len );
sts = EssGCreateMemberwKeyStr(pMember2, pKey2, &pOutStr);

/*Note: because not all elements in pOutStr are actual characters,
   e.g. the 2 bytes for the size of Member and size of Key, plus the
   \0 ending characters, the printf below does not display the actual
   contents of the array */
for (i=0;i < usMember2Len + usKey2Len + 4 + 2; ++i)
    printf("%c", pOutStr[i]);

/* Free the returned data */
EssGFreeRows(hGrid, &rDataRangeOut, ppDataOut);
sts = EssGFreeMemberwKeyStr (pOutStr);
}

if( sts == 0)
{
    EssGEndOperation(hGrid, 0);
    EssGDisconnect(hGrid, 0);
}


**ESSG_DT_MEMBERwKEY Example**

The following example illustrates the use of the Grid API constant ESSG_DT_MEMBERwKEY.

**Note:** DisplayOutput is a function that is called below in ESSG_BeginZoomIn.
ESSG_VOID_T DisplayOutput(ESSG_PPDATA_T ppDataOut, ESSG_RANGE_T pRangeOut)
{
    ESSG_ULONG_T RowIndex, ColumnIndex;
    for (RowIndex = 0; RowIndex < pRangeOut.ulNumRows; RowIndex++)
    {
        for (ColumnIndex = 0; ColumnIndex < pRangeOut.ulNumColumns; ColumnIndex++)
        {
            switch(ppDataOut[RowIndex][ColumnIndex].usType)
            {
                case(ESSG_DT_STRING):
                    printf("%s", ppDataOut[RowIndex][ColumnIndex].Value.pszStr+1);
                    break;
                case(ESSG_DT_LONG):
                    printf("%ld", ppDataOut[RowIndex][ColumnIndex].Value.lData);
                    break;
                case(ESSG_DT_DOUBLE):
                    printf("%g", ppDataOut[RowIndex][ColumnIndex].Value.dblData);
                    break;
                case(ESSG_DT_BLANK):
                    break;
                case(ESSG_DT_RESERVED):
                    printf("#Reserved");
                    break;
                case(ESSG_DT_ERROR):
                    printf("#Error");
                    break;
                case(ESSG_DT_MISSING):
                    printf("#Missing");
                    break;
                case(ESSG_DT_ZERO):
                    printf("%ld", ppDataOut[RowIndex][ColumnIndex].Value.lData);
                    break;
                case(ESSG_DT_NOACCESS):
                    printf("#NoAccess");
                    break;
                case(ESSG_DT_MEMBER):
                    printf("%s", ppDataOut[RowIndex][ColumnIndex].Value.pszStr+1);
                    break;
                case(ESSG_DT_MEMBERwKEY):
                    printf("%s", ppDataOut[RowIndex][ColumnIndex].Value.pszStr+2);
                    printf(" (Key = %s)", ppDataOut[RowIndex][ColumnIndex].Value.pszStr+5+
                        strlen(ppDataOut[RowIndex][ColumnIndex].Value.pszStr+2));
                    break;
                default:
                    break;
            }
            if (ColumnIndex < pRangeOut.ulNumColumns - 1)
            {
                printf(",");
            }
            printf("\n");
        }
    }
    printf("\n");
    printf("\n");
}

1110  C Grid API Examples
BuildTable Example Function

The following function examples call this example function:

...  

ESSG_PPDATA_T BuildTable (ESSG_PRANGE_T pRange)
{
    ESSG_PPDATA_T ppTable;
    ESSG_STR_T current_str;
    ESSG_USHORT_T slen = 0;

    pRange->ulRowStart = 0;
    pRange->ulColumnStart = 0;
    pRange->ulNumRows = 2;
    pRange->ulNumColumns = 5;
    ppTable = AllocTwoDims(2, 5);

    /* ROW 1 */
    ppTable[0][0].usType = ESSG_DT_BLANK;
    ppTable[0][1].usType = ESSG_DT_BLANK;
    slen = strlen("Year");
    current_str = malloc(sizeof(ESSG_CHAR_T)*(slen+2));
    *current_str = slen;
    strcpy( (current_str + 1), "Year");
    ppTable[0][2].usType = ESSG_DT_STRING;
    ppTable[0][2].Value.pszStr = current_str;

    slen = strlen("Product");
    current_str = malloc(sizeof(ESSG_CHAR_T)*(slen+2));
    *current_str = slen;
    strcpy( (current_str + 1), "Product");
    ppTable[0][3].usType = ESSG_DT_STRING;
    ppTable[0][3].Value.pszStr = current_str;

    slen = strlen("Market");
    current_str = malloc(sizeof(ESSG_CHAR_T)*(slen+2));
    *current_str = slen;
    strcpy((current_str + 1), "Market");
    ppTable[0][4].usType = ESSG_DT_STRING;
    ppTable[0][4].Value.pszStr = current_str;

    /*** ROW 2 ****/
    slen = strlen("Actual");
    current_str = malloc(sizeof(ESSG_CHAR_T)*(slen+2));
    *current_str = slen;
    strcpy((current_str + 1), "Actual");
    ppTable[1][0].usType = ESSG_DT_STRING;
    ppTable[1][0].Value.pszStr = current_str;
    ppTable[1][1].usType = ESSG_DT_STRING;

    slen = strlen("Sales");
    current_str = malloc(sizeof(ESSG_CHAR_T)*(slen+2));
    *current_str = slen;
    strcpy((current_str + 1), "Sales");
    ppTable[1][1].Value.pszStr = current_str;
The following function examples call this example function:

```c
ESSG_VOID_T DisplayOutput(
    ESSG_HGRID_T hGrid,
    ESSG_PPDATA_T ppDataOut,
    ESSG_RANGE_T pRangeOut)
{
    if (!ppDataOut)
    {
        printf("Data area is empty !\n");
        return;
    }

    ESSG_ULONG_T   RowIndx, ColIndx;
    printf
    ("---- Row: %d Column: %d startRow: %d, startColumn: %d
",
        pRangeOut.ulNumRows,
        pRangeOut.ulNumColumns,
        pRangeOut.ulRowStart,
        pRangeOut.ulColumnStart);

    for(RowIndx = 0; RowIndx < pRangeOut.ulNumRows; RowIndx++)
    {
        for (ColIndx = 0; ColIndx < pRangeOut.ulNumColumns; ColIndx++)
        {
            switch(ppDataOut[RowIndx][ColIndx].usType)
            {
                case(ESSG_DT_STRING):
                    printf("%s", ppDataOut[RowIndx][ColIndx].Value.pszStr+1);
                    break;

                case(ESSG_DT_LONG):
                    printf("%ld",ppDataOut[RowIndx][ColIndx].Value.lData);
                    break;

                case(ESSG_DT_DOUBLE):
                    printf("%g", ppDataOut[RowIndx][ColIndx].Value.dblData);
                    break;

                case(ESSG_DT_BLANK):
                    break;

                case(ESSG_DT_RESERVED):
                    printf("#Reserved");
                    break;
            }
        }
    }
}
```
case(ESSG_DT_ERROR):
    printf("#Error");
    break;

case(ESSG_DT_MISSING):
    printf("#Missing");
    break;

case(ESSG_DT_ZERO):
    printf("%ld", ppDataOut[RowIndx][ColIndx].Value.lData);
    break;

case(ESSG_DT_NOACCESS):
    printf("#NoAccess");
    break;

case(ESSG_DT_MEMBER):
    printf("%s", ppDataOut[RowIndx][ColIndx].Value.pszStr+1);
    break;

case(ESSG_DT_STRINGEX):
    case(ESSG_DT_MEMBEREX):
        printf("%s", ppDataOut[RowIndx][ColIndx].Value.pszStr+2);
        break;

case ESSG_DT_SMARTLIST:
    {
        ESSG_STR_T val = 0;
        printf("SmartList");
        ESSGGetFormattedValue(hGrid,&ppDataOut[RowIndx][ColIndx],&val);
        if(val)printf("-%s",val);
        ESSGGetSmartlistforCell (hGrid,&ppDataOut[RowIndx][ColIndx],&val);
        if(val)printf("Name -%s",val);
    }
    break;

case ESSG_DT_DATE:
    {
        ESSG_STR_T val = 0;
        printf("Date");
        ESSGGetFormattedValue(hGrid,&ppDataOut[RowIndx][ColIndx],&val);
        if(val)printf("-%s",val);
    }
    break;

case ESSG_DT_MNGLESS:
    printf("MeaningLess");
    break;

default:
    break;
}

    printf("(%d, %x)", ppDataOut[RowIndx][ColIndx].usType, ppDataOut[RowIndx][ColIndx].pAttributes);

    if (ColIndx < pRangeOut.ulNumColumns - 1)
FreeTwoDim Example Function

ESG_VOID_T FreeTwoDim(ESSG_PPDATA_T pDataToFree,
                     ESSG ULONG_T ulRows)
{
    ESSG ULONG_T ulIndex;

    for (ulIndex = 0; ulIndex < ulRows; ulIndex++)
    {
        if (pDataToFree[ulIndex]->usType == ESSG_DT_STRING)
        {
            free(pDataToFree[ulIndex]->Value.pszStr);
        }
        free(pDataToFree[ulIndex]);
    }
    free(pDataToFree);
}

delete
C Grid API Error Codes

The C Grid API returns three types of error codes:

- **Success**—API returns a zero value
- **Server error**—API returns a very large number. These numbers are described in the file `esserror.h`
- **API error**—API returns numbers beginning with 1100001. These values are defined in the Grid API C language header file `essgapi.h`.

If you provide a valid Error Function callback when you initialize the Grid API, this callback is called for all EGAPI errors if a valid `hGrid` is passed. The Error Function can in turn stop the default error handling user interface provided within EGAPI by returning zero (0). If a non-zero value is returned from the Error Function, or no Error Function is provided, EGAPI uses the system specific user interface to display the error message.

The following table describes the error status constants returned when a Grid API call fails.

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESSG_ERR_INITREQUIRED</td>
<td>1100001</td>
</tr>
<tr>
<td>ESSG_ERR_CONNECTREQUIRED</td>
<td>1100002</td>
</tr>
<tr>
<td>ESSG_ERR_INVALIDHANDLE</td>
<td>1100003</td>
</tr>
<tr>
<td>ESSG_ERR_INVALIDGRID</td>
<td>1100004</td>
</tr>
<tr>
<td>ESSG_ERR_CANNOTINIT</td>
<td>1100005</td>
</tr>
<tr>
<td>ESSG_ERR_CANNOTCONNECT</td>
<td>1100006</td>
</tr>
<tr>
<td>ESSG_ERR_CANTCREATEGRID</td>
<td>1100007</td>
</tr>
<tr>
<td>ESSG_ERR_INVALIDVERSION</td>
<td>1100008</td>
</tr>
<tr>
<td>ESSG_ERR_CANTGETAPIINST</td>
<td>1100009</td>
</tr>
<tr>
<td>ESSG_ERR_CANTGETAPICTX</td>
<td>1100010</td>
</tr>
<tr>
<td>ESSG_ERR_INVALIDOPTION</td>
<td>1100011</td>
</tr>
<tr>
<td>ESSG_ERR_INVALIDRANGE</td>
<td>1100012</td>
</tr>
<tr>
<td>ESSG_ERR_INVALIDDATA</td>
<td>1100013</td>
</tr>
<tr>
<td>Error Code</td>
<td>Value</td>
</tr>
<tr>
<td>--------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>ESSG_ERR_INVALIDROWORCOLMAX</td>
<td>1100014</td>
</tr>
<tr>
<td>ESSG_ERR_NULLARGUMENT</td>
<td>1100015</td>
</tr>
<tr>
<td>ESSG_ERR_CELLSREQUIRED</td>
<td>1100016</td>
</tr>
<tr>
<td>ESSG_ERR_RANGEREQUIRED</td>
<td>1100017</td>
</tr>
<tr>
<td>ESSG_ERR_INVALIDACTION</td>
<td>1100018</td>
</tr>
<tr>
<td>ESSG_ERR_INVALIDGRIDOPTION</td>
<td>1100019</td>
</tr>
<tr>
<td>ESSG_ERR_INVALIDFUNCTION</td>
<td>1100020</td>
</tr>
<tr>
<td>ESSG_ERR_MEMORY</td>
<td>1100021</td>
</tr>
<tr>
<td>ESSG_ERR_INVALIDROW</td>
<td>1100022</td>
</tr>
<tr>
<td>ESSG_ERR_INVALIDCOLUMN</td>
<td>1100023</td>
</tr>
<tr>
<td>ESSG_ERR_INVALIDPARAM</td>
<td>1100024</td>
</tr>
<tr>
<td>ESSG_ERR_INVALIDCSLVERSION</td>
<td>1100025</td>
</tr>
<tr>
<td>ESSG_ERR_RANGEOVERLAP</td>
<td>1100026</td>
</tr>
<tr>
<td>ESSG_ERR_OPERATIONFAILED</td>
<td>1100027</td>
</tr>
<tr>
<td>ESSG_ERR_CANTSETOPTION</td>
<td>1100028</td>
</tr>
<tr>
<td>ESSG_ERR_INVALIDOPTIONVALUE</td>
<td>1100029</td>
</tr>
<tr>
<td>ESSG_ERR_EMPTYARGUMENT</td>
<td>1100030</td>
</tr>
<tr>
<td>ESSG_ERR_INVALIDLROHANDLE</td>
<td>1100031</td>
</tr>
<tr>
<td>ESSG_ERR_NOLROSAVAILABLE</td>
<td>1100032</td>
</tr>
<tr>
<td>ESSG_ERR_INVALIDLROTYPE</td>
<td>1100033</td>
</tr>
<tr>
<td>ESSG_ERR_GCINITFAIL</td>
<td>1100034</td>
</tr>
<tr>
<td>ESSG_ERR_GCSSETLOCALFAIL</td>
<td>1100035</td>
</tr>
</tbody>
</table>
Part V

Visual Basic Main API

In Visual Basic Main API:

- Using the Visual Basic Main API
- Visual Basic Main API Declarations
- Visual Basic Main API Functions
In This Chapter

Visual Basic Main API Conventions ................................................................. 1119
Understanding Visual Basic API Declarations .................................................. 1119
Visual Basic Functions for Excel ...................................................................... 1120
Visual Basic API Handles .............................................................................. 1120
Visual Basic API File Objects .......................................................................... 1122
Visual Basic API Message Handling ................................................................. 1123
Visual Basic API Function Calls ..................................................................... 1125
Visual Basic API Message Call Sequence ......................................................... 1126
Visual Basic Main API Common Problems and Solutions ................................. 1132
Drill-through Visual Basic API Example ........................................................... 1132

Visual Basic Main API Conventions

Empty Strings in Visual Basic API

"Empty string" is either a fixed size string (ByVal As String * size) filled with spaces or a variable/fixed length string that has chr$(0) in the first position. As a rule, if an input parameter can be referred to as an "empty string," it could be either one of the above described. If the same applies to the return value, it can only be the second one (to allow easy testing for an empty string).

Understanding Visual Basic API Declarations

In your programs, you use the Visual Basic equivalents of C language declarations. The following table shows C language declarations, the way they are declared in the Visual Basic ESB32.BAS file, and how to call them in a Visual Basic procedure.

<table>
<thead>
<tr>
<th>C Declaration</th>
<th>Visual Basic Declaration</th>
<th>Variable to Use When Calling</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pointer to string (LPSTR)</td>
<td>ByVal S As String</td>
<td>Any String or Variant variable</td>
</tr>
<tr>
<td>Pointer to integer (LPINT)</td>
<td>I As Integer</td>
<td>Any Integer or Variant variable</td>
</tr>
<tr>
<td>Pointer to long (LPDWORD)</td>
<td>L As Long</td>
<td>Any Long or Variant variable</td>
</tr>
<tr>
<td>C Declaration</td>
<td>Visual Basic Declaration</td>
<td>Variable to Use When Calling</td>
</tr>
<tr>
<td>---------------------------------------------------</td>
<td>--------------------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>Pointer to struct (such as, LPRECT)</td>
<td>S As Rect</td>
<td>Any variable of that user-defined type</td>
</tr>
<tr>
<td>Integer (INT, UINT, WORD, BOOL)</td>
<td>ByVal I As Integer</td>
<td>Any Integer or Variant variable</td>
</tr>
<tr>
<td>Handle (hWnd, hdc, hMenu, etc.)</td>
<td>ByVal h As Integer</td>
<td>Any Integer or Variant variable</td>
</tr>
<tr>
<td>Long (DWORD, LONG)</td>
<td>ByVal L As Long</td>
<td>Any Long or Variant variable</td>
</tr>
<tr>
<td>Pointer to integers</td>
<td>I As Integer</td>
<td>The first element of the array, for example I(0)</td>
</tr>
<tr>
<td>Pointer to void (void *)</td>
<td>As Any</td>
<td>Any variable (use ByVal with strings)</td>
</tr>
<tr>
<td>Void (function return value)</td>
<td>Sub procedure</td>
<td>not applicable</td>
</tr>
</tbody>
</table>

In the main form of your program, include the following lines to set the Boolean variables ESB_TRUE and ESB_FALSE, which occur in user-defined types such as the initialization structure ESB_INIT_T:

ESB_TRUE = 1
ESB_FALSE = 0

Note: See the Microsoft Visual Basic Documentation for information about using C-language DLLs in Visual Basic programs.

**Visual Basic Functions for Excel**

The Essbase Spreadsheet Toolkit package contains a library of Visual Basic (VB) functions for use in Excel for Windows. The Excel version of VB is commonly referred to as Visual Basic for Applications, or VBA.

The Essbase Spreadsheet Toolkit is a separately sold option in the Essbase Multidimensional Analysis System.

From a VBA program in Excel, you can also call VB functions in the API libraries. To learn how to use the API within VBA, see the Oracle Essbase Spreadsheet Add-in User’s Guide.

**Visual Basic API Handles**

- “Instance Handles” on page 1120
- “Context Handles” on page 1121

**Instance Handles**

An instance handle (similar in concept to a file handle) represents a program’s access to the API, and distinguishes the program-specific resources and settings used within the API. This identification is necessary for DLLs, which may be accessed by several different programs.
simultaneously. When a program initializes the API by calling `EsbInit()`, an instance handle is returned.

**Using Instance Handle in Applications**

An instance handle is declared as type `ESB_HINST_T` in Visual Basic programs.

The instance handle must be passed to the `EsbLogin()` call, which returns a context handle, and also to the API terminate function `EsbTerm()` to free any program-specific resources used within the API.

Instance handles may be passed to other programs, child processes, or threads, which can then log in independently of the original using the same API resources and settings. Make sure that all programs, processes or threads using the same instance handle log out before they terminate the API.

**Note:** A thread may require its own instance handle (`phInstance`) to avoid overwriting another thread's networking status information.

**Context Handles**

A *context handle* represents a single, valid login by a user onto the system. A successful call to `EsbLogin()` returns a context handle, which can be passed to other API calls which require a context handle as an argument.

**Using Context Handles in Applications**

Context handles are defined as type `ESB_HCTX_T` in Visual Basic programs.

In general, a context handle is valid for as long as the user remains logged in to that server (that is, until after a successful `EsbLogout()` call). However, in case such as a server shutdown, a context handle can become invalid. Your program should therefore provide some way for the user to log back in during a session (for example, via a menu option or function key).

**Note:** A context handle is specific to an instance of the API, and contains an implied reference to the resources and settings for the appropriate instance.

**Multiple Context Handles**

A single instance of an API program may make multiple calls to `EsbLogin()`, using the same user name or different user names on one or more Essbase Servers. Each call to `EsbLogin()` returns a unique context handle, and your program must keep track of each context handle returned.

You may have up to 255 context handles per client application in use simultaneously, but if a program performs all its processing on a single server, in general it is easier to use only one context handle and to switch between different applications and/or databases as required, using either the `EsbSetActive()` function or the `EsbAutoLogin()` function.
Local Context Handles

Operations on local objects and files (objects on the client) can use a local context handle. See Using Local Context Handles and Local Contexts.

Sharing Context Handles

In general, it is not advisable to share context handles between multiple programs, processes, or threads, unless such use is guaranteed to be exclusive. A better approach is to use the same instance handle and log in each process separately. Essbase ensures that multiple logins using the same user name on the same server will only occupy one port on that server.

Visual Basic API File Objects

An Essbase object is simply a file Essbase uses, such as a database outline, a calc script, or other data. Essbase has an object system which allows you to refer to such files through the API simply by the name, the file type, and the application and database with which they are associated. This allows objects to be manipulated independently of the underlying file system (which may vary between different platforms and implementations of Essbase).

Objects can reside on any Essbase Server or client. A locking mechanism on the server controls access to objects, so that users with sufficient privileges can lock server objects and copy them to the client (using the EsbGetObject() function), edit them, and save them back to the server (using the EsbPutObject() function). Server objects can also be opened without locking for read-only access, but then cannot be saved back to the server. A user can also create or edit objects on client workstations for their personal use, or save them to the server for other users to share. Clients cannot make server to server copies.

Accessing Objects

When accessing objects through the API, the object name refers to the file name of the object (without any extension). The object types are declared in the API header file in the form ESB_OBJTYPE_XXX (where XXX represents the specific type, as in ESB_OBJTYPE_REPORT). Most objects are associated with an application and database, but some objects such as calc scripts, and rules files can be stored at the application level and used with any database within the application.
Server object files are physically located in the corresponding application or database sub-directory. However, it is not generally advisable to manipulate server object files directly. Always use the appropriate API functions to copy the files locally. Client object files are also stored by default in application and database sub-directories of the directory specified by the LocalPath setting of ESB_INIT_T. These files may be freely manipulated and edited, but you should ensure your program is well-behaved when locking and unlocking server objects which are being edited on the client (always lock an object before editing and unlock it afterwards, whether or not changes are saved).

Local Contexts

If you intend to access file objects on a client machine through the API, you need to create a local context for the API object functions to use. To create a local context, use the EsbCreateLocalContext function, which returns a context handle. This handle can be passed to any of the object API functions instead of a login context handle, and causes the API to perform the requested operation on the local client object system instead of the server. You only need to create a local context once, immediately after your program first initializes the API.

If you create a local context, your program should clean up by calling the EsbDeleteLocalContext() function before terminating the API.

Visual Basic API Message Handling

When your program calls the API, system messages and error messages are generated. Some of those messages are returned by the Essbase Server, and others are internal to the API. Your program must process these messages in some way, and if there is an error which causes the operation in progress to abort, the user may need to be informed.

This section explains the API’s message handling scheme, and then shows what C and Visual Basic developers can do to implement custom message processing in their programs:

- “How the Essbase API Handles Messages” on page 1123
- “Using Message Handling in Visual Basic Programs” on page 1124

How the Essbase API Handles Messages

The following message levels are supported in Essbase:

- Information messages (for information only)
- Warning messages (operation will continue)
- Error messages (operation aborted)
- Serious (operation aborted - system is unstable)
- fatal (operation aborted - system is halting)

When your program uses Essbase API default message handling, all messages of level Error or higher (Serious or Fatal) are displayed on the current application screen.
Using Message Handling in Visual Basic Programs

To implement message handling in Visual Basic Programs, first follow these steps as you code your program:

1. Set the ClientError field of the ESB_INIT_T structure to ESB_TRUE, and define a value for the ErrorStack field of ESB_INIT_T when it calls the initialization function EsbInit().

2. To define a custom message handler,
   - Define it using the following function signature (the function name can be changed):
     ```
     Public Function EsbErrorHandler(ByVal MsgNum As Long, ByVal Level As Long, ByVal uLog As String, ByVal uMsg As String) As Long
     ```
   - Initialize the vbCallbackFuncAddress field of the ESB_INIT_T structure to the address of the custom message handler function. Example:
     ```vba
     Sub ESB_Init()
         Dim Init As ESB_INIT_T
         Dim lx As Long
         ESB_FALSE = 0
         ESB_TRUE = 1
         Init.Version = ESB_API_VERSION
         Init.MaxHandles = 10
         Init.LocalPath = "C:\Hyperion\products\Essbase"
         Init.MessageFile = ""
         'This must be set to True
         Init.ClientError = ESB_TRUE
         Init.ErrorStack = 100
         'This is where the address of the custom function is set for
         Init.vbCallbackFuncAddress = GetProcAddress(AddressOf EsbErrorHandler)
         sts = EsbInit(Init, hInst)
         Debug.Print "EsbInit: sts = " & sts
     End Sub
     ```

3. To use the Essbase provided message handling, call the function EsbGetMessage() to retrieve any information returned after a call to the Essbase API. After the information is retrieved, your program can display or handle the information as necessary.
Setting the ClientError and ErrorStack Fields

The following code fragment shows the ClientError and ErrorStack fields set.

```
Dim Init As ESB_INIT_T

Init.ClientError = ESB_TRUE
Init.ErrorStack = 100
```

Calling EsbGetMessage()

All information, warning, and error messages accumulate in a message stack when an API function executes. When ClientError is set to ESB_TRUE, EsbGetMessage() can be used to retrieve the top message from the stack. When successful, EsbGetMessage() returns a pointer to a message level, a pointer to a message number, and a message string. It also decrements the internal message stack pointer. To ensure that no data is lost, you should call EsbGetMessage() until the function returns an empty string in the message parameter, and zeros in the number parameter. See EsbGetMessage for more details.

Visual Basic programs should call EsbGetMessage() to retrieve any information that may be generated by a call to the API. It is important to do this when the return code generated by an API call is not zero, to obtain any error or status information returned. Additionally, you should call EsbGetMessage() when the return code is zero when there is likely to be additional information returned. For example, successful calls to EsbLogin() or EsbAutoLogin() return useful information about the last login.

**Note:** If you initialize a custom error-handling function, EsbGetMessage cannot be used to retrieve.

### Visual Basic API Function Calls

This section describes calling API functions, using instance and context handles, and handling return codes.

**Function Declarations**

If you are using the API with Visual Basic, you need to include the correct function and constant declaration in your program. The file ESB32.BAS in \ESSBASE\API\INCLUDE contains the correct function and constant declarations.

The ESB32.BAS file is required for 32-bit Visual Basic programs because 32-bit programs use two bytes per character instead of one. Since some Essbase Visual Basic data structures use one-byte data types, ESB32.BAS changes these to use two bytes.
Add ESB32.BAS to your project, or, if you are using your own file for global declarations, copy the declarations included in ESB32.BAS into it.

**Passing the Instance Handle or Context Handle**

You must pass the instance handle returned by the initial call to EsbInit() in calls to EsbLogin() or EsbTerm(). You must pass the context handle returned by EsbLogin() in any function calls associated with a specific login.

**Handling the Return Code**

All Essbase API functions return a status code of type ESB_STS_T. A return code of zero indicates that the function was executed successfully, and a non-zero value indicates an error condition. A full list of error return constants is contained in the header files ESSERROR.H and ESBERROR.BAS.

**Note:** You should always check the return code from any Essbase API function. If the return code is non-zero, any pointers or values returned by the function are undefined.

**Visual Basic API Function Call Sequence**

The API requires that your program call certain functions before others. These are the basic ordering rules:

- A program must call EsbInit() before calling any other API functions.
- A program must call EsbLogin() or EsbAutoLogin() before calling any API functions which require a context handle argument (most API functions). Additionally, if you need to create a local context for API object functions to use, you must call EsbCreateLocalContext() before calling any API functions requiring a context handle argument.
- Some API functions require an active application and database to be set. This is done by having the program call EsbSetActive() or EsbAutoLogin() before they are called.
- A program must not pass a context handle to any API functions after calling EsbLogout() for that handle.
- A program must not call any API functions except EsbInit() after calling EsbTerm().

Topics that discuss Visual Basic API task sequence:

- “Typical VB API Task Sequence” on page 1127
- “Initializing the Visual Basic API” on page 1127
- “Logging In to an Essbase Server” on page 1128
- “Selecting an Active Application and Database” on page 1128
- “Retrieving and Updating Data” on page 1129
- “Recalculating the Database” on page 1130
- “Logging Out from the Essbase Server and Terminating the API” on page 1131
Typical VB API Task Sequence

This is the typical order of operations for a simple API application:

1. Create and initialize an ESB_INIT_T structure.
2. Initialize the API by calling EsbInit().
3. Log in to the required server by calling EsbLogin() or EsbAutoLogin().
4. Select an active application and database by calling EsbSetActive() or EsbAutoLogin().
5. Retrieve (or lock) data by calling EsbReport() or related functions.
6. Update data by calling EsbUpdate() or related functions.
7. Recalculate the database by calling EsbCalc() or related functions.
8. Produce reports against the data by calling EsbReport() or related functions.
9. Log out from the server by calling EsbLogout().
10. Terminate the API by calling EsbTerm().

Initializing the Visual Basic API

A program must initialize the API by calling the EsbInit() function before calling any other Essbase API functions. EsbInit() initializes all internal API state variables, and also allows you to tailor the API to your program’s requirements.

The only function you can call before EsbInit() is EsbGetAPIVersion().

The calling program must pass the EsbInit() function an initialization structure. This structure is defined in ESB32.BAS as ESB_INIT_T. It contains a series of fields which are used to customize the API and set up certain API defaults. You must declare an instance of this structure and initialize the relevant fields before calling EsbInit().

The EsbInit() function returns an instance handle, which should then be passed as an argument to the API login function.

Declaring the Visual Basic Initialization Structure

In Visual Basic, you can declare the initialization structure in the procedure that uses it. The structure shown below is taken from ESB32.BAS.

```vbnet
Type ESB_INIT_T
    Version     As Long                  ' version of API
    MaxHandles  As Integer               ' maximum number of context handles required
    LocalPath   As String * ESB_PATHLEN  ' local path to use for file operations
    MessageFile As String * ESB_PATHLEN  ' full path name of message database file
    HelpFile    As String * ESB_PATHLEN  ' full path name of help file
    ClientError As Integer               ' allows use of a pseudo client error handler
    ErrorStack  As Integer               ' size of the error message stack
End Type
```

In this code, the fields are defined as follows:

- The Version field indicates the current version of the API.
- The MaxHandles field contains the maximum number of simultaneous context handles required by the program. The default is 255.

- The LocalPath field contains the default local path name to use for file and object operations on the client. The default is $ESSBASEPATH\CLIENT, where $ESSBASEPATH is defined by the ESSBASEPATH environment variable.

- The MessageFile field contains the fully qualified path name of the message database file ESSBASE.MDB. If this is not set explicitly, Essbase first tries to use the fully qualified path in the ARBORMSGPATH environment variable. Otherwise, it uses $ESSBASEPATH\BIN \ESSBASE.MDB, where $ESSBASEPATH is defined by the ESSBASEPATH environment variable. If the ESSBASEPATH variable is not set, an error message is returned at run time.

- The HelpFile field contains the fully-qualified name of the application help file. By default, clicking the Help button displays the Essbase System Login help topic shipped with the Oracle Essbase Spreadsheet Add-in User’s Guide online help.

  - If ESSBASEPATH is not defined, the help file name is set to null.

- The ClientError field contains the value ESB_FALSE or ESB_TRUE to indicate whether EsbGetMessage() can be used to retrieve messages.

- The ErrorStack field contains the size of the message stack used by EsbGetMessage(). The default value is 100.

## Logging In to an Essbase Server

In general, the first thing your program should do after calling EsbInit() is to prompt the user for a server name, user name, and password (or use predefined defaults), then attempt to log in to that server by calling EsbLogin(). Alternatively, use the encapsulated login function, EsbAutoLogin(). If this call is successful, then the returned context handle should be stored and used for all subsequent API calls.

## Selecting an Active Application and Database

In addition to the context handle, the login functions also return a list of the applications and databases to which the logged in user has access (a program can obtain this list at any time by calling the EsbListDatabases() function). The program allows the user to select a specific application and database by calling the EsbSetActive() function.

If EsbAutoLogin() is used to log in, it can optionally set the active application and database.

To get information about an Essbase application (for example, whether or not it is already loaded), call the EsbGetApplicationState() or EsbGetApplicationInfo() functions. To get information about a specific database, call the EsbGetDatabaseState() or EsbGetDatabaseInfo() functions. You can call these functions before setting the active application and database.
Retrieving and Updating Data

- "Retrieving Data" on page 1129
- "Updating Data" on page 1129

Retrieving Data

To retrieve data from an Essbase database, either for reporting or for subsequent updating, your program needs to use a report specification. Report specifications can be in the form of a single text string (if it is less than 64 KB in length), a series of text strings, or a file. Report files can reside either on the client machine, or on the Essbase Server.

Sending a Report Specification as a Single String

To send a report specification as a single string, have the program call `EsbReport()` and pass the entire report string, not greater than 32 KB long, as an argument. If the `Output` flag is set to `TRUE` in the call to `EsbReport()`, the program must also read the returned report data by calling `EsbGetString()` repeatedly until a NULL string is returned. The returned data can then be displayed, written to a file, or printed, as required.

Sending a Report Specification as a Series of Strings

To send a report specification as a series of strings, first call `EsbBeginReport()`, then call `EsbSendString()` repeatedly to send each string in the report specification (note that in Windows, each individual string must not be greater than 32 KB long). Finally, terminate the report specification by calling `EsbEndReport()`. If the `Output` flag is set to `TRUE` in the call to `EsbBeginReport()`, the program must also read the returned report data by calling `EsbGetString()` repeatedly until a NULL string is returned. The returned data can then be displayed, written to a file, or printed, as required.

Sending a File as a Report Specification

To send a file as a report specification, use the `EsbReportFile()` function, passing the report file name. If the `Output` flag is set to `TRUE` in the call to `EsbReportFile()`, the program must also read the returned report data by calling `EsbGetString()` repeatedly until a NULL string is returned. The returned data can then be displayed, written to a file, or printed, as required.

Updating Data

To update data in the database, you should first lock the blocks in the database which you are going to update. To do this, do either of the following:

- Send a report specification as described above, with the `Output` flag set to `TRUE` and `Lock` flag also set to `TRUE`. The data output by this report can be modified, then sent back to the database as an update.
Alternatively, if there is new or modified data ready to be loaded, a program can first use it as a report specification to lock the data blocks by setting the Output flag to FALSE and setting the Lock flag to TRUE when calling the appropriate report function.

The database can be updated either from a single string, a series of strings, or a file. Update data files can reside either on the client machine, or on the Essbase Server.

### Sending Update Data as a Single String

To send an update as a single string, call `EsbUpdate()` passing the entire string as an argument. (Note that in Windows, the string must not be greater than 32 KB long). Set the Store flag to TRUE in the call to `EsbUpdate()` so that the database will be updated. If the Unlock flag is also set to TRUE, any locked data blocks in the database will be unlocked once the data is updated, to allow other users to update those blocks.

### Sending Update Data as a Series of Strings

To send an update as a series of strings, first call `EsbBeginUpdate()`, then call `EsbSendString()` repeatedly to send all the data (note that in Windows, each individual data string must not be greater than 32 KB long). Finally, terminate the update by calling `EsbEndUpdate()`. Set the Store flag to TRUE in the call to `EsbUpdate()` so that the database will be updated. If the Unlock flag is also set to TRUE, any locked data blocks in the database will be unlocked once the data is updated.

### Sending Update Data as a File

To send an update as a file, use the `EsbUpdateFile()` function, passing the data file name. Set the Store flag to TRUE in the call to `EsbUpdate()` so that the database will be updated. If the Unlock flag is also set to TRUE, any locked data blocks in the database will be unlocked once the data is updated.

### Recalculating the Database

After updating any data in the database it is essential to perform a recalculation to ensure that the consolidated totals are correct. To recalculate a database, you can either perform the default calculation, or send a specific calculation script. You can also set a calculation script to be the default calc script. Calc scripts can be sent either as a single string, a series of strings, or a file. Calc script files can reside either on the client machine, or on the Essbase Server.

### Sending a Calc Script as a Single String

To send a calc script as a single string, call `EsbCalc()` passing the entire string as an argument (note that the string must not be greater than 32 KB long). Set the Calculate flag to TRUE in the call to `EsbCalc()` so that the calc script will be executed. You then need to check on the progress of the calculation at regular intervals.
Sending a Calc Script as a Series of Strings

To send a calc script as a series of strings, first call `EsbBeginCalc()`, then call `EsbSendString()` repeatedly to send all the strings in the calc script (note that each individual string must not be greater than 32 KB long). Finally, terminate the script by calling `EsbEndCalc()`. Set the `Calculate` flag to `TRUE` in the call to `EsbBeginCalc()` so that the database will be recalculated. You then need to check on the progress of the calculation at regular intervals.

Sending a Calc Script as a File

To send a calc script as a file, use the `EsbCalcFile()` function, passing the calc script file name. Set the `Calculate` flag to `TRUE` in the call to `EsbCalcFile()` so that the database will be recalculated. You then need to check on the progress of the calculation at regular intervals.

Using the Default Calc Script

To recalculate a database using the current default calc script, use the `EsbDefaultCalc()` function. To set the default calc script for a database, use `EsbSetDefaultCalc()`, passing the calc script as a single string. To set the default calc script from a file, use the `EsbSetDefaultCalcFile()` function, passing the calc script file name. Use `EsbGetProcessState()` to determine when the calculation is finished.

Checking the Progress of Calculations

After a database calculation is started, you should check the progress of the calculation at regular intervals (five seconds is recommended) by calling the `EsbGetProcessState()` function. This function returns a structure indicating the calculation state. You should call `EsbGetProcessState()` until it indicates that the calculation is complete or that an error has occurred. You may also cancel a calculation in progress at any time by using the `EsbCancelProcess()` function.

Caution! While a calculation is in progress, do not attempt to call any API functions other than `EsbGetProcessState()` or `EsbCancelProcess()` using the same context handle, until the calculation operation has completed successfully or has been canceled. After `EsbGetProcessState()` indicates the calculation has finished, your program may continue performing other API operations with that context handle.

Logging Out from the Essbase Server and Terminating the API

When all database operations are complete, the application should first log out by calling `EsbLogout()`. This frees up any internal resources reserved within the database, and may also free the login port on the server for use by another user.

When an application program is about to terminate, it should call the `EsbTerm()` function, passing the instance handle which was returned from the original call to `EsbInit()`. This releases all resources used by the Essbase API. After calling this function, no other API calls can be made, unless `EsbInit()` is called again to reinitialize the API.
Visual Basic Main API Common Problems and Solutions

If the API function you are calling is likely to generate a lot of data in a return parameter, such as calls to EsbLogin(), EsbAutoLogin(), EsbGetString(), and EsbListDatabases(), you should ensure that you have reserved enough buffer space to receive the data. This might assist you in identifying and solving problems.

Table 8  Visual Basic Main API Common Problems and Solutions

<table>
<thead>
<tr>
<th>Problem</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Your program is generating protection faults.</td>
<td>If you are having this problem with a Visual Basic program, check the declared indirection level of any pointers being passed to the API.</td>
</tr>
<tr>
<td>Your program generates an Essbase error when calling an API function.</td>
<td>Most of the Essbase error messages are self-explanatory, and it should be fairly obvious where the problem lies. However a couple of common errors to watch out for are (%n indicates a message argument which is replaced by a context-specific string):</td>
</tr>
<tr>
<td></td>
<td>• &quot;NULL argument (%1) passed to ESSAPI function %2&quot;. This message indicates that one or more arguments passed to the API function %2 were NULL. The %1 indicates the number of the first null argument (1-based).</td>
</tr>
<tr>
<td></td>
<td>• &quot;Invalid call sequence in ESSAPI function %1&quot;. This message indicates that you have made a call to an API function (%1) when another function call was required. For example, if you have executed a report function, such as EsbReport(), make sure that you call EsbGetString() repeatedly until a NULL string is returned; or if you have executed a calculation function, e.g. EsbCalc(), that you repeatedly check the calculation state by calling EsbGetProcessState() until the returned value indicates that the calc has completed.</td>
</tr>
<tr>
<td></td>
<td>• &quot;Local operation not allowed in ESSAPI function %s&quot;. You have passed a local context handle to a function which does not allow it; use a login context handle instead.</td>
</tr>
<tr>
<td></td>
<td>• &quot;Cannot open message database %s&quot;. The message database is not accessible on the machine on which your program is running. Ensure that the message database is where Essbase expects to find it. Essbase first examines the MessagePath field of the initialization structure passed to EsbInit(), then the directory and file name specified by the ARBORMSGPATH environment variable, and finally, the $ESSBASEPATH\BIN directory where $ESSBASEPATH is an environment variable. If the message database is not available in any of these directories, Essbase returns an error message at run time. Verify which setting Essbase uses, and then verify that the message database is located where specified. See Chapter 3, &quot;Integrating Essbase With Your Product.”</td>
</tr>
<tr>
<td>Your program is consistently receiving an Essbase error return code from an API function, but no message is displayed, or a message saying &quot;No message for message #%1 in message database&quot; is generated.</td>
<td>Certain internal API errors cannot display a message, typically because the user’s context information is not available when the message occurs. In these cases, make a note of the error code returned from the function, then refer to the list of error messages in the header file ESSERROR.H to find the corresponding message text.</td>
</tr>
</tbody>
</table>

Drill-through Visual Basic API Example

```vbnet
Attribute VB_Name = "Module3"
Dim sts As Long
    Dim hInst As Long
    Dim hDestInst As Long
    Dim hCtx As Long
    Dim hDestCtx As Long
```

1132 Using the Visual Basic Main API
Dim Server As String * ESB_SVRNAMELEN
Dim User As String * ESB_USERNAMELEN
Dim Password As String * ESB_PASSWORDLEN
Dim AppName As String * ESB_APPNAMELEN
Dim DbName As String * ESB_DBNAMELEN

Sub ESB_GetVersion()
    Dim sts As Long
    Dim Release As Integer
    Dim Version As Integer
    Dim Revision As Integer
    
    sts = EsbGetVersion(hCtx, Release, Version, Revision)
    Debug.Print "EsbGetVersion: sts = " & sts
    Debug.Print "Release: " & Release
    Debug.Print "Version: " & Version
    Debug.Print "Revision: " & Revision
End Sub

Sub ESB_Init()
    Dim Init As ESB_INIT_T
    ESB_FALSE = 0
    ESB_TRUE = 1
    
    Init.Version = ESB_API_VERSION
    Init.MaxHandles = 10
    Init.LocalPath = "C:\install\zolahit\products\Essbase\EssbaseClient"
    ' Use default message file
    Init.MessageFile = ""
    ' Use EsbGetMessage to retrieve
    ' messages
    Init.ClientError = ESB_TRUE
    Init.ErrorStack = 100
    'Init.vbCallbackFuncAddress = GetProcAddress(AddressOf EsbErrorHandler)
    
    sts = EsbInit(Init, hInst)
    'MsgBox ("EsbInit = " & sts)
    Debug.Print "EsbInit: sts = " & sts
    
    'For copy objects between servers
    'sts = EsbInit(Init, hDestInst)
    'MsgBox ("EsbInit = " & sts)
    'Debug.Print "EsbInit: sts = " & sts
End Sub

Public Function GetProcAddress(ByVal lngAddressOf As Long) As Long
    GetProcAddress = lngAddressOf
End Function

Public Function EsbErrorHandler(ByVal MsgNum As Long, ByVal Level As Long, ByVal uLog As String, ByVal uMsg As String) As Long
    If Level >= ESB_LEVEL_ERROR Then
        MsgBox "Error: " & MsgNum & " - " & uMsg
    End If
    'MsgBox " Info " & MsgNum & ": Level: " & Level & ": " & uLog & ": " & uMsg
End Function
Sub ESB_GetMessage()
Dim DbName As String
Dim FilterName As String
Const szMessage = 256
Dim Message As String * szMessage
Dim Number As Long
Dim Level As Integer
Dim sts As Long
Dim Object As ESB_OBJDEF_T
Dim hOutline As Long
Dim hMemberProfit As Long

Object.hCtx = hCtx
Object.Type = ESB_OBJTYPE_OUTLINE
Object.AppName = "Temp"
Object.DbName = "Basic"
Object.FileName = "Basic"
sts = EsbOtlOpenOutline(hCtx, Object, ESB_YES, ESB_YES, hOutline)
Debug.Print "EsbOtlOpenOutline: sts = " & sts

sts = EsbOtlFindMember(hOutline, "100-10", hMember)
Debug.Print "EsbOtlFindMember: sts = " & sts

If sts > 0 Then
sts = EsbGetMessage(hInst, Level, Number, Message, szMessage)
Do While Mid$(Message, 1, 1) <> Chr$(0)
    Debug.Print Level
    Debug.Print Number
    Debug.Print Message
    sts = EsbGetMessage(hInst, Level, Number, Message, szMessage)
Loop
End If
End Sub

Sub ESB_Login()
Dim Items As Integer
Dim AppDb As ESB_APPDB_T

Server = "ppamu-pc1"
User = "essexer"
Password = "password"
sts = EsbLogin(hInst, Server, User, Password, Items, hCtx)
Debug.Print "EsbLogin: sts = " & sts

'For n = 1 To Items
'    sts = EsbGetNextItem(hCtx, ESB_LAPPDB_TYPE, AppDb)
'    Debug.Print "EsbGetNextItem: sts = " & sts
'    Debug.Print "App Name: "; AppDb.AppName
'    Debug.Print "Db Name: "; AppDb.DbName
' Next

'For copy objects between servers
'sts = EsbLogin(hDestInst, "qtfsun1:1501", User, Password, Items, hDestCtx)
'Debug.Print "EsbLogin: sts = " & sts
End Sub
Sub ESB_AutoLogin()
    Dim pOption As Integer
    Dim pAccess As Integer

    Server = "localhost"
    'User = "essexer"
    'Password = "Password"
    'AppName = "sample"
    'DbName = "basic"

    'pOption = ESB_AUTO_NODIALOG + ESB_AUTO_NOSELECT
    pOption = ESB_AUTO_DEFAULT
    sts = EsbAutoLogin(hInst, Server, User, Password, AppName, DbName, pOption, pAccess, hCtx)
    'MsgBox ("EsbAutoLogin = " & sts)
    Debug.Print "EsbAutoLogin: sts = " & sts
    ' Call Esb_runreport
End Sub

Sub ESB_LoginSetPassword()
    Dim hInst       As Long
    Dim Server      As String * ESB_SVRNAMELEN
    Dim User        As String * ESB_USERNAMELEN
    Dim Password    As String * ESB_PASSWORDLEN
    Dim NewPassword  As String * ESB_PASSWORDLEN
    Dim Items        As Integer
    Dim AppDb        As ESB_APPDB_T

    Server = "stiahp1:1501"
    User = "essexer"
    Password = "password"
    NewPassword = "password2"
    sts = EsbLoginSetPassword(hInst, Server, User, Password, NewPassword, Items, hCtx)
    Debug.Print "EsbLoginSetPassword: sts = " & sts

    For N = 1 To Items
        sts = EsbGetNextItem(hCtx, ESB_LAPPDB_TYPE, AppDb)
        Debug.Print "EsbGetNextItem: sts = " & sts
        Debug.Print "App Name: "; AppDb.AppName
        Debug.Print "Db Name: "; AppDb.DbName
    Next

    'Reset password back to original
    NewPassword = "password"
    sts = EsbLoginSetPassword(hInst, Server, User, Password, NewPassword, Items, hCtx)
    Debug.Print "EsbLoginSetPassword: sts = " & sts
End Sub

Sub ESB_SetActive()
    Dim AppName As String
    Dim DbName As String
    Dim pAccess As Integer
    Dim sts As Long

    'AppName = "Bugs"
    'DbName = "09129823"
AppName = "vb"
DbName = "Basic"

sts = EsbSetActive(hCtx, AppName, DbName, pAccess)
Debug.Print "EsbSetActive: sts = " & sts
End Sub

Sub ESb_GetStoresInfo() '(Chnl As String)
Dim Object As ESB_OBJDEF_T

Object.hCtx = hCtx
Object.Type = ESB_OBJTYPE_OUTLINE
Object.AppName = AppName
Object.DbName = DbName
Object.FileName = DbName
Dim hMember As Long
Dim ihMember As Long
Dim MbrInfo As ESB_MBRINFO_T
Dim Counts As ESB_MBRCOUNTS_T

sts = EsbSetActive(hCtx, AppName, DbName, Access)

Dim hMemberJan As Long
Dim MbrChldCnt As Long
Dim x As Integer
Dim Parent As String
Dim found As Boolean
Dim img As Integer
Dim Member As String
Dim szAlias As String * ESB_MBRNAMELEN
Dim Alias As String
Dim levelnum As String
Dim ShareStat As Integer
Dim tLevelName As String * ESB_MBRNAMELEN
Const AltGroup As String = "ALT_GROUP"

sts = EsbOtlOpenOutline(hCtx, Object, ESB_YES, ESB_YES, hOutline)

If sts = 0 Then
    sts = EsbOtlFindMember(hOutline, "JOHNSON, ROGER", hMemberJan)
    'sts = EsbOtlFindMember(hOutline, "GMM_A", hMemberJan)
    If hMemberJan = 0 Then
        sts = EsbOtlFindAlias(hOutline, "JOHNSON, ROGER", "default", hMemberJan)
    End If
End If

If sts = 0 And hMemberJan <> 0 Then
    sts = EsbOtlGetMemberInfo(hOutline, hMemberJan, MbrInfo)
    MsgBox ("Member Name = " & MbrInfo.szMember)
    Member = MbrInfo.szMember
    levelnum = MbrInfo.usLevel
    ShareStat = MbrInfo.usShare
    MsgBox ("Shared Member = " & ShareStat)
End If

MbrChldCnt = MbrInfo.ulChildCount
' If ShareStat <> ESB_SHARE_SHARE Then

'Do While x <= MbrChldCnt
For x = 1 To MbrChldCnt
  If x = 1 Then
    sts = EsbOtlGetChild(hOutline, hMemberJan, hMember)
    'sts = EsbOtlGetMemberInfo(hOutline, hMember, MbrInfo)
    'MsgBox ("Child Member Name = " & MbrInfo.szMember)
  Else
    sts = EsbOtlGetNextSibling(hOutline, hMemberJan, hMember)
    ' sts = EsbOtlGetMemberInfo(hOutline, hMember, MbrInfo)
    ' MsgBox ("Sibling Member Name = " & MbrInfo.szMember)
  End If

  sts = EsbOtlGetMemberInfo(hOutline, hMember, MbrInfo)
  MsgBox ("Sibling Member Name = " & MbrInfo.szMember)
  ' szAlias = ""
  'sts = EsbOtlGetMemberAlias(hOutline, hMember, ",", szAlias)
  'sts = EsbOtlGetLevelName(hOutline, sRoot, MbrInfo.usLevel, tLevelName)
  'If sts > 0 Then tLevelName = ""

  'Alias = sTrim(szAlias)
  'Member = sTrim(MbrInfo.szMember)

Next
End Sub

Sub ESB_Logout()

  sts = EsbLogout(hCtx)
  'MsgBox ("EsbLogout = " & sts)
  Debug.Print "EsbLogout: sts = " & sts
End Sub

Sub ESB_Term()

  sts = EsbTerm(hInst)
  'MsgBox ("EsbTerm = " & sts)
  Debug.Print "EsbTerm: sts = " & sts
End Sub

Public Sub ESB_LROListObjects()
  Dim UserName As String * ESB_USERNAMELEN
  Dim listDate As Long
  Dim Items As Integer
  Dim Desc As ESB_LRODESC_API_T
  Dim i As Integer
  Dim j As Integer
  Dim CutOffDate As Date
  Dim MemberName As String * ESB_MBRNAMELEN

  Const ESB_REFERENCE_DATE = #1/1/1970#
  UserName = "essexer"
  CutOffDate = #9/21/2007#
  'CutOffDate = #1/2/1970#
listDate = DateDiff("s", CutOffDate, ESB_REFERENCE_DATE)
'listDate = DateDiff("s", ESB_REFERENCE_DATE, CutOffDate)
'listDate = -1

sts = EsbLROListObjects(hCtx, UserName, listDate, Items)
Debug.Print "EsbLROListObjects: sts = " & sts

Debug.Print "Number of LRO(s): " & Items

If sts = 0 Then
    For i = 1 To Items
        Debug.Print "LRO # " & i; ":"
        sts = EsbGetNextItem(hCtx, ESB_LRO_TYPE, Desc)
        Debug.Print "EsbGetNextItem: sts = " & sts
        Debug.Print "Object Type: " & Desc.ObjType
        Select Case (Desc.ObjType)
            Case 0
                Debug.Print "Cell notes: " & Desc.note
            Case 1
                Debug.Print "Object Name: " & Desc.lroInfo.ObjName
                Debug.Print "Object Description: " & Desc.lroInfo.objDesc
            Case 2
                Debug.Print "Object Name: " & Desc.lroInfo.ObjName
                Debug.Print "Object Description: " & Desc.lroInfo.objDesc
        End Select
        Debug.Print "Member Combination:"
        For j = 1 To Desc.memCount
            sts = EsbLROGetMemberCombo(hCtx, j, MemberName)
            Debug.Print "   " & MemberName
        Next j
    Next i
End If
End Sub

Sub ESB_SetUser()
    Dim sts As Long
    Dim UserInfo As ESB_USERINFO_T

    UserInfo.Name = "Test"
    UserInfo.Type = ESB_TYPE_USER
    UserInfo.Access = ESB_ACCESS_SUPER
    UserInfo.MaxAccess = ESB_ACCESS_SUPER
    UserInfo.PwdChgNow = ESB_TRUE

    sts = EsbSetUser(hCtx,UserInfo)
    Debug.Print "EsbSetUser: sts = " & sts
End Sub

Sub ESB_GetUser()
    Dim sts As Long
    Dim User As String
    Dim.UserInfo As ESB_USERINFO_T

    User = "Test"
' Get User Info structure

sts = EsbGetUser(hCtx, User, UserInfo)
Debug.Print "EsbGetUser: sts = " & sts
End Sub

Public Sub ESB_LROPurgeObjects()
    Dim UserName As String * ESB_USERNAMELEN
    Dim purgeDate As Long
    Dim Items As Integer
    Dim Desc As ESB_LRODESC_API_T
    Dim CutOffDate As Date
    Dim i As Integer
    Const ESB_REFERENCE_DATE = #1/1/1970#

    UserName = "essexer"
    CutOffDate = #9/21/2007#
    purgeDate = DateDiff("s", ESB_REFERENCE_DATE, CutOffDate) 'bug 8-651484045
    'purgeDate = DateDiff("s", CutOffDate, ESB_REFERENCE_DATE)
    'purgeDate = -1

    sts = EsbLROPurgeObjects(hCtx, UserName, purgeDate, Items)
    Debug.Print "EsbLROPurgeObjects: sts = " & sts

    If sts = 0 Then
        For i = 1 To Items
            '*******************************
            '* Get the next LRO description
            '* item from the list
            '*******************************
            sts = EsbGetNextItem(hCtx, ESB_LRO_TYPE, Desc)
            Debug.Print "EsbGetNextItem: sts = " & sts
        Next i
    End If
End Sub

Sub ESB_CreateGroup()
    Dim sts As Long
    Dim GroupName As String
    GroupName = "PowerUsers"
    sts = EsbCreateGroup(hCtx, GroupName)
    Debug.Print "EsbCreateGroup: sts = " & sts
End Sub

Sub ESB_GetDatabaseInfo()
    Dim sts As Long
    Dim AppName As String
    Dim DbName As String
    Dim Items As Integer
    Dim N As Integer
    Dim DbInfo As ESB_DBINFO_T
    Dim DbReqInfo As ESB_DBREQINFO_T

    AppName = "Sample"
    DbName = "Basic"
sts = EsbGetDatabaseInfo(hCtx, AppName, DbName, DbInfo, Items)
Debug.Print "EsbGetDatabaseInfo: sts = " & sts
Debug.Print "DbInfo.status: " & DbInfo.Status

If sts = 0 Then
    For N = 1 To Items
        sts = EsbGetNextItem(hCtx, ESB_DBREQINFO_TYPE, DbReqInfo)
        Debug.Print "EsbGetNextItem: sts = " & sts
    Next
End If
End Sub

Sub ESB_GetDatabaseAccess()
    Dim Items As Integer
    Dim AppName As String
    Dim DbName As String
    Dim User As String
    Dim UserDb As ESB_USERDB_T
    Dim sts As Long

    AppName = "Sample"
    DbName = "Basic"

    User = "user1"
    sts = EsbGetDatabaseAccess(hCtx, User, AppName, DbName, Items)
    Debug.Print "EsbGetDatabaseAccess: sts = " & sts
    For N = 1 To Items
        sts = EsbGetNextItem(hCtx, ESB_USERDB_TYPE, UserDb)
        Debug.Print "EsbGetNextItem: sts = " & sts
        Debug.Print "User: " & User
    Next

    User = "user2"
    sts = EsbGetDatabaseAccess(hCtx, User, AppName, DbName, Items)
    Debug.Print "EsbGetDatabaseAccess: sts = " & sts
    For N = 1 To Items
        sts = EsbGetNextItem(hCtx, ESB_USERDB_TYPE, UserDb)
        Debug.Print "EsbGetNextItem: sts = " & sts
        Debug.Print "User: " & User
    Next

    User = "user3"
    sts = EsbGetDatabaseAccess(hCtx, User, AppName, DbName, Items)
    Debug.Print "EsbGetDatabaseAccess: sts = " & sts
    For N = 1 To Items
        sts = EsbGetNextItem(hCtx, ESB_USERDB_TYPE, UserDb)
        Debug.Print "EsbGetNextItem: sts = " & sts
        Debug.Print "User: " & User
    Next

    User = "user4"
    sts = EsbGetDatabaseAccess(hCtx, User, AppName, DbName, Items)
    Debug.Print "EsbGetDatabaseAccess: sts = " & sts
    For N = 1 To Items
sts = EsbGetNextItem(hCtx, ESB_USERDB_TYPE, UserDb)
Debug.Print "EsbGetNextItem: sts = " & sts
Debug.Print "User: " & User
Next

User = "user5"
sts = EsbGetDatabaseAccess(hCtx, User, AppName, DbName, Items)
Debug.Print "EsbGetDatabaseAccess: sts = " & sts
For N = 1 To Items
    sts = EsbGetNextItem(hCtx, ESB_USERDB_TYPE, UserDb)
    Debug.Print "EsbGetNextItem: sts = " & sts
    Debug.Print "User: " & User
Next

User = "user6"
sts = EsbGetDatabaseAccess(hCtx, User, AppName, DbName, Items)
Debug.Print "EsbGetDatabaseAccess: sts = " & sts
For N = 1 To Items
    sts = EsbGetNextItem(hCtx, ESB_USERDB_TYPE, UserDb)
    Debug.Print "EsbGetNextItem: sts = " & sts
    Debug.Print "User: " & User
Next
End Sub

Sub ESB_GetDatabaseStats()
    Dim Items As Integer
    Dim AppName As String
    Dim DbName As String
    Dim DbStats As ESB_DBSTATS_T
    Dim DimStats As ESB_DIMSTATS_T
    Dim sts As Long
    AppName = "Sample"
    DbName = "Basic"
    sts = EsbGetDatabaseStats(hCtx, AppName, DbName, DbStats, Items)
    Debug.Print "EsbGetDatabaseStats: sts = " & sts
    'MsgBox ("cluster = " & DbStats.ClusterRatio)
    For N = 1 To Items
        sts = EsbGetNextItem(hCtx, ESB_DBSTATS_TYPE, DbStats)
    Next
End Sub

Public Sub ESB_LROAddObject()
    Dim Desc As ESB_LRODESC_API_T
    Dim memCount As Long
    Dim memComb As String
    Dim opt As Integer
    Dim i As Integer

    memCount = 5
    memComb = "Year" & vbCrLf & "Product" & vbCrLf & "Market" & vbCrLf & "Measures" & vbCrLf & "Scenario"
    Desc.UserName = "essexer"
    Desc.ObjType = ESB_LROTYPE_CELLNOTE_API
Desc.note = "Cell note"
opt = ESB_NOSTORE_OBJECT_API
sts = EsbLROAddObject(hCtx, memCount, memComb, opt, Desc)
Debug.Print "EsbLROAddObject: sts = " & sts

Desc.ObjType = ESB_LROTYPE_WINAPP_API
Desc.lroInfo.ObjName = "c:\hyperion\essbase95\bin\essbase.exe"
Desc.lroInfo.objDesc = "Essbase executable."
opt = ESB_STORE_OBJECT_API
sts = EsbLROAddObject(hCtx, memCount, memComb, opt, Desc)
Debug.Print "EsbLROAddObject: sts = " & sts

Desc.ObjType = ESB_LROTYPE_URL_API
Desc.lroInfo.ObjName = "www.oracle.com"
Desc.lroInfo.objDesc = "Oracle homepage"
opt = ESB_NOSTORE_OBJECT_API
sts = EsbLROAddObject(hCtx, memCount, memComb, opt, Desc)
Debug.Print "EsbLROAddObject: sts = " & sts

Desc.ObjType = ESB_LROTYPE_CELLNOTE_API
Desc.note = "Cell note 2"
opt = ESB_NOSTORE_OBJECT_API
sts = EsbLROAddObject(hCtx, memCount, memComb, opt, Desc)
Debug.Print "EsbLROAddObject: sts = " & sts

End Sub

Public Sub ESB_LROGetCatalog()

    Dim Desc As ESB_LRODESC_API_T
    Dim Items As Integer
    Dim memCount As Long
    Dim memComb As String
    Dim i As Integer

    memCount = 5
    memComb = "Qtr1" & vbCrLf & "Profit" & vbCrLf & "100" & vbCrLf & "East" & vbCrLf & "Scenario"
    'memComb = "Jan" & vbCrLf & "Sales" & vbCrLf & "Cola" & vbCrLf & "Utah" & vbCrLf & "Actual"

    sts = EsbLROGetCatalog(hCtx, memCount, memComb, Items)
    Debug.Print "EsbLROGetCatalog: sts = " & sts

    If sts = 0 Then
        For i = 1 To Items
            sts = EsbGetNextItem(hCtx, ESB_LRO_TYPE, Desc)
            Debug.Print "Desc.ObjType = " & Desc.ObjType
            Debug.Print "Desc.note = " & Desc.note
            Debug.Print "Desc.lroInfo.objDesc = " & Desc.lroInfo.objDesc
            Debug.Print "Desc.lroInfo.objName = " & Desc.lroInfo.objName
        Next i
    End If
End Sub

Sub ESB_CopyObject()
    Dim sts As Long
    1142
Dim SrcApp As String
Dim SrcDb As String
Dim SrcObj As String
Dim DestApp As String
Dim DestDb As String
Dim DestObj As String

SrcApp = "Sample"
SrcDb = "Basic"
SrcObj = "Basic"

DestApp = "Sample"
DestDb = "Basic"
DestObj = "Basic1"
ObjType = ESB_OBJTYPE_OUTLINE

sts = EsbCopyObject(hCtx, hDestCtx, ObjType, SrcApp, DestApp, _
  SrcDb, DestDb, SrcObj, DestObj)
Debug.Print "EsbCopyObject: sts = " & sts
End Sub

Sub ESB_GetAssociatedAttributesInfo()
  Dim sts As Long
  Dim MbrName As String
  Dim AttrDimName As String
  Dim Count As Long
  Dim Attribinfo As ESB_ATTRIBUTEINFO_T
  Dim index As Integer
  Dim tempstring As String

  'MbrName = InputBox("Base member name", "Base Member Name")
  'AttrDimName = InputBox("Attribute Dimension Name (Optional)", "Attribute Dimension Name")

  MbrName = "em41666"
  AttrDimName = "Job Start Date"
  sts = EsbGetAssociatedAttributesInfo(hCtx, MbrName, AttrDimName, Count)
  Debug.Print "EsbGetAssociatedAttributesInfo: sts = " & sts

  Debug.Print "Associated Attr info for: " & MbrName
  For index = 1 To Count
    sts = EsbGetNextItem(hCtx, ESB_ATTRIBUTEINFO_TYPE, Attribinfo)
    'Debug.Print "Dim Name: " & Attribinfo.DimName
    Debug.Print "Attribute Dim Name: " & Attribinfo.DimName
    Debug.Print "Attribute Mbr Name: " & Attribinfo.MbrName

    ' NOTE: use of select case statement to discern (and act upon) type of attribute returned
    Select Case VarType(Attribinfo.Attribute)
      Case vbDouble
        Debug.Print "Data Type : Numeric(Double)"
        Debug.Print "Data Value : " & Attribinfo.Attribute
        Debug.Print ""
      Case vbBoolean
        Debug.Print "Data Type : Boolean"
        Debug.Print "Data Value : " & Attribinfo.Attribute
    End Select
  Next index
End Sub
Debug.Print ""

Case vbDate
    Debug.Print "Data Type    : Date"
    Debug.Print "Data Value   : " & Attribinfo.Attribute
    Debug.Print ""
Case vbString
    Debug.Print "Data Type    : String"
    Debug.Print "Data Value   : " & Attribinfo.Attribute
    Debug.Print ""
End Select
Debug.Print ""
Next index
End Sub

Sub ESB_ListConnections()
    Dim Items As Integer
    Dim UserInfo As ESB_USERINFO_T
    Dim sts As Long

    sts = EsbListConnections(hCtx, Items)
    Debug.Print "EsbListConnections: sts = " & sts

    For N = 1 To Items
        sts = EsbGetNextItem(hCtx, ESB_USERINFO_TYPE, UserInfo)
        Debug.Print "EsbGetNextItem: sts = " & sts
    Next
End Sub

Sub ESB_ListRequests()
    Dim Items As Integer
    Dim ReqInfo As ESB_REQUESTINFO_T
    Dim sts As Long

    sts = EsbListRequests(hCtx, UserName, AppName, DbName, Items)
    Debug.Print "EsbListRequests: sts = " & sts

    For N = 1 To Items
        sts = EsbGetNextItem(hCtx, ESB_REQUESTINFO_TYPE, ReqInfo)
        Debug.Print "EsbGetNextItem: sts = " & sts
        Debug.Print "AppName: " & ReqInfo.AppName
        Debug.Print "DbName: " & ReqInfo.DbName
        Debug.Print "DbRequestCode: " & ReqInfo.DbRequestCode
        Debug.Print "LoginID: " & ReqInfo.LoginId
        Debug.Print "LoginSourceMachine: " & ReqInfo.LoginSourceMachine
        Debug.Print "RequestString: " & ReqInfo.RequestString
        Debug.Print "State: " & ReqInfo.State
        Debug.Print "TimeStarted: " & ReqInfo.TimeStarted
        Debug.Print "Username: " & ReqInfo.UserName
    Next
End Sub

Sub ESB_AddToGroup()
    Dim sts As Long
    Dim GroupName As String
    Dim User As String

    GroupName = "Group1"

User = "user1"
sts = EsbAddToGroup(hCtx, GroupName, User)
Debug.Print "EsbAddToGroup sts: " & sts
End Sub

Sub ESB_GetGroupList()
Dim Items As Integer
Dim Group As String
Dim GroupName As String * ESB_USERNAMELEN
Dim sts As Long

Group = "group1"
sts = EsbGetGroupList(hCtx, Group, Items)
Debug.Print "EsbGetGroupList: sts = " & sts
For N = 1 To Items
    sts = EsbGetNextItem(hCtx, ESB_GROUPNAME_TYPE, ByVal GroupName)
    Debug.Print "EsbGetGroupList: sts = " & sts
    Debug.Print "User Name = " & GroupName
    MsgBox ("User Name = " & GroupName)
Next
End Sub

Sub ESB_GetDatabaseState()
Dim sts As Long
Dim AppName As String
Dim DbName As String
Dim DbState As ESB_DBSTATE_T
AppName = "Sample"
DbName = "Basic"

sts = EsbGetDatabaseState(hCtx, AppName, DbName, DbState)
Debug.Print "EsbGetDatabaseState: sts = " & sts
End Sub

Sub ESB_CreateLocalContext()
Dim sts As Long
Dim User As String
Dim Password As String
Dim hCtx As Long

'*******************************
' Create Local Context
'*******************************
sts = EsbCreateLocalContext(hInst, User, Password, hCtx)
End Sub

Sub ESB_Import()
Dim sts As Long
Dim Rules As ESB_OBJDEF_T
Dim Data As ESB_OBJDEF_T
Dim User As ESB_MBRUSER_T
Dim ErrorName As String
Dim AbortOnError As Integer
Dim hLocalCtx As Long
Need to create a local context, if files are not on the server

sts = EsbCreateLocalContext(hInst, "", "", hLocalCtx)
Debug.Print "EsbCreateLocalContext sts: " & sts
Data.hCtx = hLocalCtx
Data.Type = ESB_OBJTYPE_TEXT
Data.AppName = ""
Data.DbName = ""
Data.FileName = "F:\testArea\VBAPI\calcdat.txt"

Rules file resides at the server

'Rules.hCtx = hCtx
'Rules.Type = ESB_OBJTYPE_RULES
'Rules.AppName = "Demo"
'Rules.DbName = "Basic"
'Rules.FileName = "Test"

Data file resides at the server

'Data.hCtx = hCtx
'Data.Type = ESB_OBJTYPE_TEXT
'Data.AppName = "Demo"
'Data.DbName = "Basic"
'Data.FileName = "Data"

Specify file to redirect errors
to if any

ErrorName = "IMPORT.ERR"

Abort on the first error

AbortOnError = ESB_YES

Import

sts = EsbImport(hCtx, Rules, Data, User, ErrorName, AbortOnError)
Debug.Print "EsbImport sts: " & sts
End Sub

Sub ESB_VerifyFilter()
  Dim sts As Long
  Dim AppName As String
  Dim DbName As String
  Dim Row As String

  AppName = "Sample"
  DbName = "Basic"

  sts = EsbVerifyFilter(hCtx, AppName, DbName)
Debug.Print "EsbVerifyFilter sts: " & sts

' Initialize Filter Row
Row = "@IDESCENDANTS(Scenario)"
sts = EsbVerifyFilterRow(hCtx, Row) ' Initialize Filter Row
Debug.Print "EsbVerifyFilterRow sts: " & sts

Row = "@IDESCENDANTS(AAAA)"
sts = EsbVerifyFilterRow(hCtx, Row)
Debug.Print "EsbVerifyFilterRow sts: " & sts

sts = EsbVerifyFilterRow(hCtx, ByVal 0&)
Debug.Print "EsbVerifyFilterRow sts: " & sts
End Sub

Sub Test()
strComputer = "."
Const ForReading = 1
Const ForWriting = 2
Const ForAppending = 8
'==============================================================
Const Data_Path = "F:\Testarea\temp\"
Const FileName = "process.txt"

Set fso = CreateObject("Scripting.FileSystemObject")
If Not fso.FileExists(Data_Path & FileName) Then
    Set f = fso.OpenTextFile(Data_Path & FileName, 2, True)
Else
    Set f = fso.OpenTextFile(Data_Path & FileName, 8)
End If

Set objWMIService = GetObject("winmgmts:" & "{impersonationLevel=impersonate}" & strComputer & "\root\cimv2")
Set colProcessList = objWMIService.ExecQuery("Select * from Win32_Process")
For Each objProcess In colProcessList
    f.WriteLine "Process " & objProcess.Name
Next
End Sub

Sub ESB_CreateGLDrillThru()
Dim sts As Long
Dim url As ESB_DURLINFO_T
Dim cppDrillRegions(0 To 1) As String

'***************************************************************
' Need to create a local context, if files are not on the server
'***************************************************************
url.bIsLevel0 = 0

cppDrillRegions(0) = "sales"
cppDrillRegions(1) = "cogs"
url.cpURLXML = "Testing"
url.cpURLName = "VB URL7"
url.iURLXMLSize = 8

sts = EsbCreateDrillThruURL(hCtx, cppDrillRegions, url)
Debug.Print "EsbCreateDrillThruURL sts: " & sts

End Sub

Sub ESB_UpdateGLDrillThru()
    Dim sts As Long
    Dim url As ESB_DURLINFO_T
    Dim cppDrillRegions(0 To 1) As String
    Dim bMerge As Integer

    '***************************************************************
    ' Need to create a local context, if files are not on the server
    '***************************************************************
    url.bIsLevel0 = 0
    bMerge = ESB_TRUE

    cppDrillRegions(0) = "qtr1"
    url.cpURLXML = "Testing"
    url.cpURLName = "VB URL7"
    url.iURLXMLSize = 8

    sts = EsbUpdateDrillThruURL(hCtx, cppDrillRegions, url, bMerge)

    Debug.Print "EsbUpdateDrillThruURL sts: " & sts
End Sub

Sub ESB_DeleteGLDrillThru()
    Dim URLName As String

    URLName = "VB URL7"
    sts = EsbDeleteDrillThruURL(hCtx, URLName)

    Debug.Print "EsbDeleteDrillThruURL sts: " & sts
End Sub

Sub ESB_GetGLDrillThru()
    Dim URLName As String
    Dim url As ESB_DURLINFO_T
    Dim intX As Integer
    Dim cppDrillRegions As Variant

    URLName = "VB URL2"
    sts = EsbGetDrillThruURL(hCtx, URLName, url, cppDrillRegions)

    Debug.Print "EsbGetDrillThruURL sts: " & sts

    If sts = 0 Then
        Debug.Print "URL Name: " & url.cpURLName
        Debug.Print "URL XML: " & url.cpURLXML

        For intX = LBound(cppDrillRegions) To UBound(cppDrillRegions)
            Debug.Print "URL Region: " & cppDrillRegions(intX)
        Next
    End If
End Sub
Sub ESB_ListGLDrillThru()
    Dim intX As Integer
    Dim URLNames As Variant

    sts = EsbListDrillThruURLs(hCtx, URLNames)

    If sts = 0 Then
        Debug.Print "EsbListDrillThruURL sts: " & sts
        For intX = LBound(URLNames) To UBound(URLNames)
            Debug.Print "URL Name: " & URLNames(intX)
        Next
    End If
End Sub

Sub ESB_GetCellDrillThruReports()
    Dim intX As Integer
    Dim mbrs(0 To 4) As String
    Dim pURLXMLLens As Variant
    Dim pURLXMLs As Variant

    mbrs(0) = "sales"
    mbrs(1) = "jan"
    mbrs(2) = "New York"
    mbrs(3) = "actual"
    mbrs(4) = "100-10"

    sts = EsbGetCellDrillThruReports(hCtx, mbrs, pURLXMLLens, pURLXMLs)

    If sts = 0 Then
        Debug.Print "EsbGetCellDrillThruReports sts: " & sts
        For intX = LBound(pURLXMLLens) To UBound(pURLXMLLens)
            Debug.Print "URL XML: " & intX
            Debug.Print "URL XML Len: " & pURLXMLLens(intX)
            Debug.Print "URL XML String: " & pURLXMLs(intX)
        Next
    End If

    mbrs(0) = "profit"
    sts = EsbGetCellDrillThruReports(hCtx, mbrs, pURLXMLLens, pURLXMLs)

    If sts = 0 Then
        Debug.Print "EsbGetCellDrillThruReports sts: " & sts
        For intX = LBound(pURLXMLLens) To UBound(pURLXMLLens)
            Debug.Print "URL XML: " & intX
            Debug.Print "URL XML Len: " & pURLXMLLens(intX)
            Debug.Print "URL XML String: " & pURLXMLs(intX)
        Next
    End If
Next
End If
End Sub

Sub Main()
' Test
ESB_Init
' ESB_CreateLocalContext
' ESB_AutoLogin
ESB_Login
' ESB_LoginSetPassword
ESB_SetActive
ESB_CreateGLDrillThru
ESB_UpdateGLDrillThru
ESB_GetGLDrillThru
ESB_ListGLDrillThru
ESB_GetCellDrillThruReports
ESB_DeleteGLDrillThru
' ESB_GetGLDrillThru
' ESB_ListGLDrillThru
ESB_GetCellDrillThruReports
' ESB_SetUser
' ESB_GetUser
' ESB_GetMessage
' ESB_Import
' ESB_GetVersion
' ESB_GetDatabaseInfo
' ESB_GetDatabaseState
' ESB_GetDatabaseStats
' ESB_GetDatabaseAccess
' ESB_GetGroupList
' ESB_ListConnections
' ESB_ListRequests
' ESB_GetAssociatedAttributesInfo
' ESb_GetStoresInfo
' ESB_OtlGetMemberAlias
' ESB_AddAliasCombination
' ESB_CreateGroup
' ESB_LROAddObject
' ESB_LROGetCatalog
' ESB_LROListObjects
' ESB_LROPurgeObjects

' ESB_CopyObject
' ESB_PartitionReadDefFile
' ESB_PartitionWriteDefFile
' ESB_PartitionReplaceDefFile
' ESB_PartitionValidateDefinition
' ESB_PartitionValidateLocal
' ESB_PartitionReadOtlChangeFile

' ESB_AddToGroup
' ESB_GetGroupList
' ESB_VerifyFilter
ESB_Logout

1150 Using the Visual Basic Main API
ESB_Term
End Sub
In This Chapter

- Constant Definitions ................................................................................... 1153
- Constant and Structure Definitions for Linked Objects ............................................. 1155
- Constant and Structure Definitions for Partitions .................................................. 1157
- Standard C Language Types .......................................................................... 1157
- Standard Visual Basic Language Types .............................................................. 1159
- Visual Basic API Attributes Terminology .............................................................. 1164
- Visual Basic Main API Structures ..................................................................... 1165

Constant Definitions

The following constants are defined in the Essbase Visual Basic global text file ESB32.BAS, and in the C language header file ESBAPI.H

- “Maximum String Lengths” on page 1153
- “Information Flag Constants” on page 1154
- “Size Flag Constants” on page 1154
- “Dimension Tag Constants” on page 1154

Maximum String Lengths

The following constants define the maximum lengths of various string types in the Essbase VB API and must be used for variable declarations in a VB application:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESB_USERNAMELEN</td>
<td>Maximum length of a user or group name</td>
</tr>
<tr>
<td>ESB_PASSWORDLEN</td>
<td>Maximum length of a user password</td>
</tr>
<tr>
<td>ESB_SVRNAMELEN</td>
<td>Maximum length of a server name</td>
</tr>
<tr>
<td>ESB_APPNAMELEN</td>
<td>Maximum length of an application name</td>
</tr>
<tr>
<td>ESB_DBNAMELEN</td>
<td>Maximum length of a database name</td>
</tr>
</tbody>
</table>
### Information Flag Constants

The following constants define the available information flags used in the `DbReqFlags` (Data Load) field in the “ESB_DBREQINFO_T” on page 1173 structure.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESB_DBREQFLAG_CALCDEF</td>
<td>Default flag for <code>DbReqFlags</code> field. Used the default calc script. Value: 1.</td>
</tr>
</tbody>
</table>

### Size Flag Constants

The following constants define the maximum and minimum size for the `MaxMemIndex` and `IndexPageSize` fields in the “ESB_DBSTATE_T” on page 1173 structure.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESB_INDEXCACHEMIN_SIZE</td>
<td>Minimum index cache size for the <code>MaxMemIndex</code> field of the ESB_DBSTATE_T structure. Value: 1048576. No maximum value is defined.</td>
</tr>
<tr>
<td>ESB_INDEXPAGEMAX_SIZE</td>
<td>Maximum index page size for the <code>IndexPageSize</code> field of the ESB_DBSTATE_T structure. Value: 8192</td>
</tr>
<tr>
<td>ESB_INDEXPAGEMIN_SIZE</td>
<td>Minimum index page size for the <code>IndexPageSizeMin</code> field of the ESB_DBSTATE_T structure. Value: 1024</td>
</tr>
</tbody>
</table>

### Dimension Tag Constants

The following constants define the available information flags used in the `DimTag` field in the “ESB_DIMENSIONINFO_T” on page 1177 structure.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESB_TTYPE_NONE</td>
<td>No dimension type. Value for <code>DimTag</code> field of ESB_DIMENSIONINFO_T.</td>
</tr>
</tbody>
</table>
### Constants and Structure Definitions for Linked Objects

The following constants and structures are defined specifically for use with LROs:

- “Constants for LROs” on page 1155
- “ESB_CELLADDR_API_T” on page 1156
- “ESB_LRODESC_API_T” on page 1156
- “ESB_LROHANDLE_API_T” on page 1156
- “ESB_LROINFO_API_T” on page 1157

### Constants for LROs

The following constants define various values used by LROs functions and structures in the Essbase Visual Basic API.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESB_LRODESCLEN_API</td>
<td>79</td>
<td>Maximum length of an object description</td>
</tr>
<tr>
<td>ESB_LRONOTELEN_API</td>
<td>599</td>
<td>Maximum length of a cell note</td>
</tr>
<tr>
<td>ESB_ONAMELEN_API</td>
<td>511</td>
<td>Length of an object name consisting of file name and path</td>
</tr>
<tr>
<td>ESB_STORE_OBJECT_API</td>
<td>&amp;H0010</td>
<td>Value specifying to store a linked object on the server</td>
</tr>
<tr>
<td>ESB_NOSTORE_OBJECT_API</td>
<td>&amp;H0001</td>
<td>Value specifying not to store a linked object on the server</td>
</tr>
<tr>
<td>ESB_LRO_OBJ_API</td>
<td>1</td>
<td>Value specifying to update only the linked object file</td>
</tr>
<tr>
<td>ESB_LRO_CATALOG_API</td>
<td>2</td>
<td>Value specifying to update only the object's catalog entry</td>
</tr>
<tr>
<td>ESB_LRO_BOTH_API</td>
<td>3</td>
<td>Value specifying to update both the object file and the catalog entry</td>
</tr>
<tr>
<td>ESB_LROTYPE_CELLNOTE_API</td>
<td>0</td>
<td>Value specifying that a linked object is a cell note</td>
</tr>
<tr>
<td>ESB_LROTYPE_WINAPP_API</td>
<td>1</td>
<td>Value specifying that a linked object is a Windows application</td>
</tr>
<tr>
<td>ESB_LROTYPE_URL_API</td>
<td>2</td>
<td>Value specifying that a linked object is a URL</td>
</tr>
</tbody>
</table>

---

**Constant and Structure Definitions for Linked Objects 1155**
**ESB_CELLADDR_API_T**

This structure contains information about the address of a data cell. Essbase derives the cell address from the member combination and uses the address to keep track of objects linked to data cells. The `EsbLROAddObject` function returns the cell address in the object's description structure; you can use this information in subsequent API calls. You cannot modify fields in this structure through the API. The fields are described as follows:

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Long</td>
<td>cellOffset</td>
<td>Cell offset within a data block</td>
</tr>
<tr>
<td>Double</td>
<td>blkOffset</td>
<td>Block offset</td>
</tr>
<tr>
<td>Double</td>
<td>segment</td>
<td>Segment number</td>
</tr>
</tbody>
</table>

**ESB_LRODESC_API_T**

This structure contains information describing a specific object linked to a data cell in an Essbase database. The fields are described as follows:

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td>ObjType</td>
<td>The object type</td>
</tr>
<tr>
<td>Integer</td>
<td>status</td>
<td>The catalog entry status</td>
</tr>
<tr>
<td>Long</td>
<td>memCount</td>
<td>The number of member names in the member combination identifying the data cell</td>
</tr>
<tr>
<td>ESB_LROHANDLE_API_T</td>
<td>LinkID</td>
<td>A link to the object's identification structure</td>
</tr>
<tr>
<td>Long</td>
<td>updateDate</td>
<td>The last date the object was modified</td>
</tr>
<tr>
<td>Integer</td>
<td>accessLevel</td>
<td>The access level for the data cell associated with the linked object</td>
</tr>
<tr>
<td>String * ESB_USERNAMELEN</td>
<td>userName</td>
<td>The name of the last user to modify the object</td>
</tr>
<tr>
<td>String</td>
<td>memComb</td>
<td>The member combination that identifies the data cell associated with the linked object</td>
</tr>
<tr>
<td>String * ESB_LRONOTELEN_API</td>
<td>note</td>
<td>A cell note, associated by union</td>
</tr>
<tr>
<td>ESB_LROINFO_API_T</td>
<td>lroInfo</td>
<td>The LRO information structure, associated by union</td>
</tr>
</tbody>
</table>

**ESB_LROHANDLE_API_T**

This structure provides an identifier for a linked object. The identifier consists of a cell address and an internal object handle. You should not modify fields in this structure. The fields are described as follows:
### ESB_LROINFO_API_T

This structure contains descriptive information about a specific object linked to a data cell in an Essbase database. You might modify this structure when updating an object's source file name or object description. To do this, use `EsbLROGetCatalog` to retrieve the object's catalog entry, modify the `objName` and/or `objDesc` fields as needed, then use `EsbLROUpdateObject` to save your changes on the server. The fields are described as follows:

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String * ESB_ONAMELEN_API</td>
<td>objName</td>
<td>Source file name of object linked to a data cell. ESB_ONAMELEN_API specifies the maximum length of an object name; the default value is 511.</td>
</tr>
<tr>
<td>String * ESB_LRODESCLEN_API</td>
<td>objDesc</td>
<td>Description of an object linked to a data cell. ESB_LRODESCLEN_API specifies the maximum length of the description; the default value is 79.</td>
</tr>
</tbody>
</table>

### Constant and Structure Definitions for Partitions

“ESB_PART_CONNECT_INFO_T” on page 1185
“ESB_PART_DEFINED_T” on page 1186
“ESB_PART_INFO_T” on page 1186
“ESB_PART_REPL_T” on page 1188
“ESB_PARTOTL_QRY_FILTER_T” on page 1188
“ESB_PARTOTL_QUERY_T” on page 1189
“ESB_PARTSLCT_T” on page 1189

### Standard C Language Types

The following tables list the data types defined in `ESBAPI.H` for use in C language programs:

- Table 9, “Simple Data Types,” on page 1158
- Table 10, “Other Data Types,” on page 1158
- Table 11, “Pointer Types,” on page 1158
- Table 12, “Miscellaneous Types,” on page 1159
- Table 13, “API Definitions,” on page 1159
### Table 9  Simple Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Essbase Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>typedef char</td>
<td>ESB_CHAR_T</td>
</tr>
<tr>
<td>typedef short</td>
<td>ESB_SHORT_T</td>
</tr>
<tr>
<td>typedef long</td>
<td>ESB_LONG_T</td>
</tr>
<tr>
<td>typedef unsigned char</td>
<td>ESB_UCHAR_T</td>
</tr>
<tr>
<td>typedef unsigned short</td>
<td>ESB_USHORT_T</td>
</tr>
<tr>
<td>typedef unsigned long</td>
<td>ESB_ULONG_T</td>
</tr>
<tr>
<td>typedef float</td>
<td>ESB_FLOAT_T</td>
</tr>
<tr>
<td>typedef double</td>
<td>ESB_DOUBLE_T</td>
</tr>
</tbody>
</table>

### Table 10  Other Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Essbase Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>typedef unsigned char</td>
<td>ESB_BOOL_T</td>
<td>boolean</td>
</tr>
<tr>
<td>typedef char</td>
<td>*ESB_STR_T</td>
<td>string (array of char)</td>
</tr>
<tr>
<td>typedef void</td>
<td>*ESB_HINST_T</td>
<td>API instance handle</td>
</tr>
<tr>
<td>typedef void</td>
<td>*ESB_HCTX_T</td>
<td>API context handle</td>
</tr>
<tr>
<td>typedef void</td>
<td>ESB_VOID_T</td>
<td>void</td>
</tr>
<tr>
<td>typedef size_t</td>
<td>ESB_SIZE_T</td>
<td>size of a memory block</td>
</tr>
<tr>
<td>typedef unsigned short</td>
<td>ESB_ACCESS_T</td>
<td>Essbase access level</td>
</tr>
<tr>
<td>typedef unsigned long</td>
<td>ESB_LOGINID_T</td>
<td>Essbase login id</td>
</tr>
</tbody>
</table>

### Table 11  Pointer Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Essbase Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>typedef char</td>
<td>*ESB_PCHAR_T</td>
<td>pointer to char</td>
</tr>
<tr>
<td>typedef short</td>
<td>*ESB_PSHORT_T</td>
<td>pointer to short</td>
</tr>
<tr>
<td>typedef long</td>
<td>*ESB_PLONG_T</td>
<td>pointer to long</td>
</tr>
<tr>
<td>typedef unsigned char</td>
<td>*ESB_PUCHAR_T</td>
<td>pointer to unsigned char</td>
</tr>
<tr>
<td>typedef unsigned short</td>
<td>*ESB_PUSHORT_T</td>
<td>pointer to unsigned short</td>
</tr>
<tr>
<td>typedef unsigned long</td>
<td>*ESB_PULONG_T</td>
<td>pointer to unsigned long</td>
</tr>
<tr>
<td>typedef float</td>
<td>*ESB_PFLOAT_T</td>
<td>pointer to float</td>
</tr>
<tr>
<td>Data Type</td>
<td>Essbase Type</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>--------------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>typedef double</td>
<td>*ESB_PDOUBLE_T</td>
<td>pointer to double</td>
</tr>
<tr>
<td>typedef ESB_BOOL_T</td>
<td>*ESB_PBOOL_T</td>
<td>pointer to boolean</td>
</tr>
<tr>
<td>typedef ESB_STR_T</td>
<td>*ESB_PSTR_T</td>
<td>pointer to string</td>
</tr>
<tr>
<td>typedef ESB_VOID_T</td>
<td>*ESB_PVOID_T</td>
<td>pointer to void</td>
</tr>
<tr>
<td>typedef ESB_SIZE_T</td>
<td>*ESB_PSIZE_T</td>
<td>pointer to size of a memory block</td>
</tr>
<tr>
<td>typedef ESB_HINST_T</td>
<td>*ESB_PHINST_T</td>
<td>pointer to VB API instance handle</td>
</tr>
<tr>
<td>typedef ESB_HCTX_T</td>
<td>*ESB_PHCTX_T</td>
<td>pointer to VB API context handle</td>
</tr>
<tr>
<td>typedef ESB_ACCESS_T</td>
<td>*ESB_PACCESS_T</td>
<td>pointer to security access level</td>
</tr>
<tr>
<td>typedef ESB_LOGINID_T</td>
<td>*ESB_PLOGINID_T</td>
<td>pointer to login id</td>
</tr>
</tbody>
</table>

### Table 12  Miscellaneous Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Essbase Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>typedef long</td>
<td>ESB_STS_T</td>
<td>return value from API functions</td>
</tr>
<tr>
<td>typedef ESB_STS_T</td>
<td>(*ESB_FUNC_T)()</td>
<td>pointer to function</td>
</tr>
</tbody>
</table>

### Table 13  API Definitions

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>#define ESB_TRUE</td>
<td>1</td>
</tr>
<tr>
<td>#define ESB_FALSE</td>
<td>0</td>
</tr>
<tr>
<td>#define ESB_NULL</td>
<td>NULL</td>
</tr>
</tbody>
</table>

### Standard Visual Basic Language Types

The following tables describe C data types for use in Visual Basic applications whenever the VB API function refers to `ESB_xx...x_T` (except for user-defined types). Visual Basic does not allows the definition of new data types based on these data types.

- Table 14, “Simple Data Types,” on page 1160
- Table 15, “Bitmask Data Types,” on page 1161
- Table 16, “Additional Data Types,” on page 1162
- Table 17, “Pointer Types,” on page 1163
- Table 18, “Miscellaneous Types,” on page 1163
- Table 19, “Boolean Flags,” on page 1163
<table>
<thead>
<tr>
<th>Data Type</th>
<th>Essbase Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>As String * 1</td>
<td>ESB_CHAR_T</td>
</tr>
<tr>
<td>As Integer</td>
<td>ESB_SHORT_T</td>
</tr>
<tr>
<td>As Long</td>
<td>ESB_LONG_T</td>
</tr>
<tr>
<td>As String * 1</td>
<td>ESB_UCHAR_T</td>
</tr>
<tr>
<td>As Integer</td>
<td>ESB_USHORT_T</td>
</tr>
<tr>
<td>As Long</td>
<td>ESB_UULONG_T</td>
</tr>
<tr>
<td>As Long</td>
<td>ESB_FLOAT_T</td>
</tr>
<tr>
<td>As Long</td>
<td>ESB_DOUBLE_T</td>
</tr>
<tr>
<td>As Long</td>
<td>ESB_TIME_T</td>
</tr>
<tr>
<td>As Long</td>
<td>ESB_DATE_T</td>
</tr>
<tr>
<td>Data Type</td>
<td>Essbase Type</td>
</tr>
<tr>
<td>------------</td>
<td>---------------</td>
</tr>
<tr>
<td>As Integer</td>
<td>ESB_ACCESS_T</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Data Type</td>
<td>Essbase Type</td>
</tr>
<tr>
<td>-----------</td>
<td>--------------</td>
</tr>
</tbody>
</table>
| As Integer | ESB_OBJTYPE_T | File object type. Single object types are:  
ESB_OBJTYPE_NONE  
ESB_OBJTYPE_OUTLINE  
ESB_OBJTYPE_CALCSCRIPT  
ESB_OBJTYPE_REPORT  
ESB_OBJTYPE_RULES  
ESB_OBJTYPE_ALIAS  
ESB_OBJTYPE_STRUCTURE  
ESB_OBJTYPE_ASCBACKUP  
ESB_OBJTYPE_BINBACKUP  
ESB_OBJTYPE_EXCEL  
ESB_OBJTYPE_LOTUS2 (No longer supported)  
ESB_OBJTYPE_LOTUS3 (No longer supported)  
ESB_OBJTYPE_TEXT  
ESB_OBJTYPE_PARTITION  
ESB_OBJTYPE_LOTUS4 (No longer supported)  
ESB_OBJTYPE_WIZARD  
ESB_OBJTYPE_SELECTION  
ESB_OBJTYPE_LRO  
Combined object types are:  
ESB_OBJTYPE_BACKUP  
ESB_OBJTYPE_WORKSHEET  
ESB_OBJTYPE_DATA  
ESB_OBJTYPE_ALL |
| Note:  
The values for bitmap data types consist of bit values that are combined to provide additional values when appropriate. For example, a caller needing WRITE access to a database must have the READ and WRITE privileges, thus ESB_ACCESS_WRITE equals the bit values for ESB_PRIV_READ and ESB_PRIV_WRITE. Similarly, ESB_OBJTYPE_BACKUP is a combination of ESB_OBJTYPE_ASCBACKUP and ESB_OBJTYPE_BINBACKUP. |

### Table 16 Additional Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Essbase Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ByVal As String * 1</td>
<td>ESB_BOOL_T</td>
<td>boolean</td>
</tr>
<tr>
<td>ByVal As String</td>
<td>*ESB_STR_T</td>
<td>string (array of char)</td>
</tr>
<tr>
<td>ByVal As Long</td>
<td>*ESB_HINST_T</td>
<td>API instance handle</td>
</tr>
<tr>
<td>ByVal As Long</td>
<td>*ESB_HCTX_T</td>
<td>API context handle</td>
</tr>
<tr>
<td>As Any</td>
<td>ESB_VOID_T</td>
<td>void</td>
</tr>
<tr>
<td>ByVal As Long</td>
<td>ESB_SIZE_T</td>
<td>size of a memory block</td>
</tr>
</tbody>
</table>
### Table 17  Pointer Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Essbase Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>As Integer</td>
<td>*ESB_PSHORT_T</td>
<td>pointer to short</td>
</tr>
<tr>
<td>As Long</td>
<td>*ESB_PLONG_T</td>
<td>pointer to long</td>
</tr>
<tr>
<td>As Integer</td>
<td>*ESB_PUSHORT_T</td>
<td>pointer to unsigned short</td>
</tr>
<tr>
<td>As Long</td>
<td>*ESB_PULONG_T</td>
<td>pointer to unsigned long</td>
</tr>
<tr>
<td>As Long</td>
<td>*ESB_PFLOAT_T</td>
<td>pointer to float</td>
</tr>
<tr>
<td>As Long</td>
<td>*ESB_PDOUBLE_T</td>
<td>pointer to double</td>
</tr>
<tr>
<td>As Any</td>
<td>*ESB_PVOID_T</td>
<td>pointer to void</td>
</tr>
<tr>
<td>As Long</td>
<td>*ESB_PSIZE_T</td>
<td>pointer to size of a memory block</td>
</tr>
<tr>
<td>As Long</td>
<td>*ESB_PHINST_T</td>
<td>pointer to VB API instance handle</td>
</tr>
<tr>
<td>As Long</td>
<td>*ESB_PHCTX_T</td>
<td>pointer to VB API context handle</td>
</tr>
<tr>
<td>As Integer</td>
<td>*ESB_PACCESS_T</td>
<td>pointer to security access level</td>
</tr>
<tr>
<td>As Long</td>
<td>*ESB_PLOGINID_T</td>
<td>pointer to Essbase login id</td>
</tr>
</tbody>
</table>

### Table 18  Miscellaneous Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Essbase Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ByVal As Long</td>
<td>ESB_STS_T</td>
<td>return value from API functions</td>
</tr>
<tr>
<td>As Long</td>
<td>ESB_FUNC_T</td>
<td>pointer to function</td>
</tr>
</tbody>
</table>

### Table 19  Boolean Flags

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Essbase Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>chr$(1)</td>
<td>ESB_TRUE</td>
<td>Boolean TRUE to be used in data structures</td>
</tr>
<tr>
<td>chr$(0)</td>
<td>ESB_FALSE</td>
<td>Boolean FALSE to be used in data structures</td>
</tr>
<tr>
<td>1</td>
<td>ESB_YES</td>
<td>YES flag to be used in the list of the VB API function parameters</td>
</tr>
<tr>
<td>0</td>
<td>ESB_NO</td>
<td>NO flag to be used in the list of the VB API function parameters</td>
</tr>
<tr>
<td>ByVal 0&amp;</td>
<td>NULL</td>
<td>Null</td>
</tr>
</tbody>
</table>
## Visual Basic API Attributes Terminology

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bucketing Type</strong></td>
<td>When building a dimension, you can associate a zero-level attribute member of type ESB_ATTRMBRDT_DOUBLE with a range of data in a relational source. Bucketing type determines the upper or lower limit of the data range. See “ESB_ATTRSPECS_T” on page 1169.</td>
</tr>
<tr>
<td><strong>ESB_ATTRIBUTE_DIMENSION and ESB_ATTRIBUTE_MEMBER</strong></td>
<td>ESB_ATTRIBUTE_DIMENSION is an attribute dimension. ESB_ATTRIBUTE_MEMBER is a member of an attribute dimension. See “ESB_ATTRIBUTEQUERY_T” on page 1458. Also see EsbCheckAttributes.</td>
</tr>
<tr>
<td><strong>ESB_ATTRIBUTED_MEMBER</strong></td>
<td>ESB_ATTRIBUTED_MEMBER is a member (of a base dimension) which has an attribute member associated with it. See “ESB_ATTRIBUTEQUERY_T” on page 1458. Also see EsbCheckAttributes.</td>
</tr>
<tr>
<td><strong>ESB_BASE_DIMENSION and ESB_BASE_MEMBER</strong></td>
<td>ESB_BASE_DIMENSION is a standard dimension that has an attribute dimension associated with it. ESB_BASE_MEMBER is a member of a base dimension. See “ESB_ATTRIBUTEQUERY_T” on page 1458. Also see EsbCheckAttributes.</td>
</tr>
<tr>
<td><strong>ESB_STANDARD_DIMENSION and ESB_STANDARD_MEMBER</strong></td>
<td>ESB_STANDARD_DIMENSION is any dimension that is not an attribute dimension. ESB_STANDARD_MEMBER is a member of a standard dimension. See “ESB_ATTRIBUTEQUERY_T” on page 1458. Also see EsbCheckAttributes.</td>
</tr>
</tbody>
</table>
| **Long Name** | A zero-level attribute member that is not of type ESB_ATTRMBRDT_STRING is uniquely identified by a long name. A zero-level attribute member of type ESB_ATTRMBRDT_STRING must itself be unique. See the following structures:  
  - “ESB_ATTRSPECS_T” on page 1169  
  - “ESB_ATTRIBUTEINFO_T” on page 1168  
  Also see the following functions:  
    - EsbGetAttributeSpecifications  
    - EsbOtlGetAttributeSpecifications  
    - EsbOtlSetAttributeSpecifications  
  And, see Notes on Adding an Attribute Member. |
| **Short Name** | A zero-level attribute member that is not of type ESB_ATTRMBRDT_STRING is called a short name. It is provided to a function as a parameter of type ESB_STR_T. See EsbOtlFindAttributeMembers. |
Visual Basic Main API Structures

**ESB_APPDB_T**
An application and database name structure used to return matching application and database names. The fields are:

*Type ESB_APPDB_T*

```vbnet
    Type ESB_APPDB_T
        Name As String * ESB_APPNAMELEN
        Server As String * ESB_SVRNAMELEN
        status As Integer
        AppType As Integer
        nConnects As Integer
        nDbs As Integer
        ElapsedAppTime As Long
        storageType As Integer
        AppLocale As String * ESB_LOCALESTRING_LENGTH
    End Type
```

<table>
<thead>
<tr>
<th>VB Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>As String * ESB_APPNAMELEN</td>
<td>Name</td>
<td>The application name</td>
</tr>
<tr>
<td>As String * ESB_SVRNAMELEN</td>
<td>Server</td>
<td>The server name</td>
</tr>
</tbody>
</table>

**ESB_APPINFO_T**
This is an Application Info Structure used to get information about a specific application. Fields in this structure cannot be modified using the VB API. See also the “ESB_APPSTATE_T” on page 1167 structure, which contains additional application state parameters that can be modified. The fields are:

*Type ESB_APPINFO_T*

```vbnet
    Type ESB_APPINFO_T
        Name As String * ESB_APPNAMELEN
        Server As String * ESB_SVRNAMELEN
        status As Integer
        AppType As Integer
        nConnects As Integer
        nDbs As Integer
        ElapsedAppTime As Long
        storageType As Integer
        AppLocale As String * ESB_LOCALESTRING_LENGTH
    End Type
```

<table>
<thead>
<tr>
<th>VB Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>As String * ESB_APPNAMELEN</td>
<td>Name</td>
<td>The application name</td>
</tr>
<tr>
<td>As String * ESB_SVRNAMELEN</td>
<td>Server</td>
<td>The server name</td>
</tr>
<tr>
<td>VB Data Type</td>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------</td>
<td>-------------</td>
</tr>
<tr>
<td>As Integer</td>
<td>Status</td>
<td>Application load status. The values are as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESB_STATUS_NOTLOADED</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESB_STATUS_LOADING</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESB_STATUS_LOADED</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESB_STATUS_UNLOADING</td>
</tr>
<tr>
<td>As Integer</td>
<td>AppType</td>
<td>The type of application. Valid values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESB_APP_UNICODE - 0x0003 - The program is a Unicode client program. The function fails if the server is not in Unicode mode. This is the default value.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESB_APP_NONUNICODE - 0x0002 - The program is a non-Unicode mode client program.</td>
</tr>
<tr>
<td>As Integer</td>
<td>nConnects</td>
<td>The number of users currently connected to the application</td>
</tr>
<tr>
<td>As Integer</td>
<td>nDbs</td>
<td>Number of databases in this application</td>
</tr>
<tr>
<td>As Long</td>
<td>ElapsedAppTime</td>
<td>Elapsed number of seconds since application loading</td>
</tr>
<tr>
<td>As Integer</td>
<td>StorageType</td>
<td>The storage type. The valid values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- 0 - the default</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- 1 - multidimensional</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- 2 - DB2 relational</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- 3 - Oracle relational</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- 4 - aggregate storage (ASO)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- 1000 - Undefined</td>
</tr>
<tr>
<td>As String</td>
<td>AppLocale</td>
<td>The application locale description, of type ESB_LOCALESTRING_LENGTH.</td>
</tr>
</tbody>
</table>

**ESB_APPINFOEX_T**

This structure is used in EsbGetApplicationInfoEx(). The fields are:

```vbnet
Type ESB_APPINFOEX_T
    Name As String * ESB_APPNAMELEN
    Server As String * ESB_SVRNAMELEN
    AppType As Integer
    AppLocale As String * ESB_LOCALESTRING_LENGTH
    storageType As Integer
    status As Integer
    nConnects As Integer
    ElapsedAppTime As Long
End Type
```

<table>
<thead>
<tr>
<th>VB Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>As String * ESB_APPNAMELEN</td>
<td>Name</td>
<td>Application name</td>
</tr>
<tr>
<td>VB Data Type</td>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------</td>
<td>-------------</td>
</tr>
<tr>
<td>As String * ESB_SVRNAMELEN</td>
<td>Server</td>
<td>Network server name</td>
</tr>
<tr>
<td>As Integer</td>
<td>AppType</td>
<td>The type of application. Valid values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESB_APP_UNICODE - 0x0003 - The program is a Unicode client program. The function fails if the server is not in Unicode mode. This is the default value.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESB_APP_NONUNICODE - 0x0002 - The program is a non-Unicode mode client program.</td>
</tr>
<tr>
<td>As String * ESB_LOCALESTRING_LENGTH</td>
<td>AppLocale</td>
<td>The application locale description, of type ESB_LOCALESTRING_LENGTH.</td>
</tr>
<tr>
<td>As Integer</td>
<td>StorageType</td>
<td>The storage type. The valid values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- 0 - the default</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- 1 - multidimensional</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- 2 - DB2 relational</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- 3 - Oracle relational</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- 4 - aggregate storage (ASO)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- 1000 - Undefined</td>
</tr>
<tr>
<td>As Integer</td>
<td>Status</td>
<td>Application load status</td>
</tr>
<tr>
<td>As Integer</td>
<td>nConnects</td>
<td>Number of users connected</td>
</tr>
<tr>
<td>As Long</td>
<td>ElapsedApp Time</td>
<td>Elapsed application time: number of seconds the application has been loaded</td>
</tr>
</tbody>
</table>

**ESB_APPSTATE_T**

This Application State Structure gets and sets the state parameters for a specific Application. All fields in this structure can be modified using the VB API, with the exception that some fields do not apply to aggregate storage databases. See also the "ESB_APPINFO_T" on page 1165 structure, which contains additional Application information that cannot be modified. The fields are:

```vbnet
type ESB_APPSTATE_T
    Description As String * ESB_DESCLEN
    Loadable As Integer
    Autoload As Integer
    Access As Integer
    Connects As Integer
    Commands As Integer
    Updates As Integer
    LockTimeout As Long
    lroSizeLimit As Long
    Security As Integer
End Type
```
<table>
<thead>
<tr>
<th>VB Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>As String * ESB_DESCLEN</td>
<td>Description</td>
<td>Application description (up to 80 characters)</td>
</tr>
<tr>
<td>As String * 1</td>
<td>Loadable</td>
<td>Flag to indicate whether the application can be loaded (ESB_TRUE if the application is loadable).</td>
</tr>
<tr>
<td>As String * 1</td>
<td>Autoload</td>
<td>Flag to indicate whether the application is loaded automatically when Essbase Server is started (ESB_TRUE if the application will be automatically loaded).</td>
</tr>
<tr>
<td>As Integer</td>
<td>Access</td>
<td>Default access to databases in the application (lowest possible level of access for all users). Values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESB_PRIV_NONE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESB_PRIV_DBDESIGN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESB_PRIV_CALC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESB_PRIV_WRITE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESB_PRIV_READ</td>
</tr>
<tr>
<td>As String * 1</td>
<td>Connects</td>
<td>Flag to indicate whether users can connect to the application (ESB_TRUE if users can connect).</td>
</tr>
<tr>
<td>As String * 1</td>
<td>Commands</td>
<td>Flag to indicate whether users can issue commands to the application (ESB_TRUE if the application is accepting user commands).</td>
</tr>
<tr>
<td>As String * 1</td>
<td>Updates</td>
<td>Flag to indicate whether users can update data in the application (ESB_TRUE if the application is accepting user update commands).</td>
</tr>
<tr>
<td>As Long</td>
<td>LockTimeout</td>
<td>Time-out period (in seconds) after which block-level locks are automatically removed. This field does not apply to aggregate storage databases.</td>
</tr>
<tr>
<td>As Long</td>
<td>LroSizeLimit</td>
<td>Limit on the size of LRO files. This limit is set for each application and enables the administrator or program to protect the server from overly large linked files. Essbase itself does not limit the size or have a default value. This limit does not apply to LRO URLs (limited to 512 characters) or to LRO cell notes (limited to 599 characters). This field does not apply to aggregate storage databases.</td>
</tr>
<tr>
<td>As String * 1</td>
<td>Security</td>
<td>Flag to indicate whether application security is enabled (ESB_TRUE if security is enabled).</td>
</tr>
</tbody>
</table>

**ESB_ATTRIBUTEINFO_T**

This structure has information on attributes.

```vb
Type ESB_ATTRIBUTEINFO_T
    MbrName As String * ESB_MBRNAMELEN
    DimName As String * ESB_MBRNAMELEN
    Attribute As Variant
End Type
```

<table>
<thead>
<tr>
<th>VB Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>As String * ESB_MBRNAMELEN</td>
<td>MbrName</td>
<td>Attribute member name from &quot;ESB_MEMBERINFO_T&quot; on page 1182 or &quot;ESB_MBRINFO_T&quot; on page 1460, including a long name</td>
</tr>
<tr>
<td>As String * ESB_MBRNAMELEN</td>
<td>DimName</td>
<td>Attribute dimension name</td>
</tr>
</tbody>
</table>
### ESB_ATTRSPECS_T

This structure is used by `EsbOtlSetAttributeSpecifications()` to set attribute specifications for the outline, and by `EsbOtlGetAttributeSpecifications()` and `EsbGetAttributeSpecifications()` to get attribute specifications for the outline.

```vbnet
Type ESB_ATTRSPECS_T
    DefaultTrueString    As String * ESB_MBRNAMELEN
    DefaultFalseString   As String * ESB_MBRNAMELEN
    DefaultAttrCalcDimName As String * ESB_MBRNAMELEN
    DefaultSumMbrName    As String * ESB_MBRNAMELEN
    DefaultCountMbrName  As String * ESB_MBRNAMELEN
    DefaultAverageMbrName As String * ESB_MBRNAMELEN
    DefaultMinMbrName    As String * ESB_MBRNAMELEN
    DefaultMaxMbrName    As String * ESB_MBRNAMELEN
    GenNameBy            As Integer
    UseNameOf            As Integer
    Delimiter            As Integer
    DateFormat           As Integer
    BucketingType        As Integer
End Type
```

### VB Data Type

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>As Variant</td>
<td>Attribute value</td>
</tr>
</tbody>
</table>

### Visual Basic Main API Structures

<table>
<thead>
<tr>
<th>VB Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>As String * ESB_MBRNAMELEN</td>
<td>DefaultTrueString</td>
<td>The string used with the boolean attribute to indicate TRUE. The default value is ESB_DEFAULT_TRUESTRING (&quot;True&quot;).</td>
</tr>
<tr>
<td>As String * ESB_MBRNAMELEN</td>
<td>DefaultFalseString</td>
<td>The string used with the boolean attribute to indicate FALSE. The default value is ESB_DEFAULT_FALSESTRING (&quot;False&quot;).</td>
</tr>
<tr>
<td>As String * ESB_MBRNAMELEN</td>
<td>DefaultAttrCalcDimName</td>
<td>The name of the attribute calculations (aggregate) dimension. The default value is ESB_DEFAULTATTRIBUTECALCULATIONS (&quot;Attribute Calculations&quot;).</td>
</tr>
<tr>
<td>As String * ESB_MBRNAMELEN</td>
<td>DefaultSumMbrName</td>
<td>The name used with the attribute calculations (aggregate) dimension to indicate SUM. The default value is ESB_DEFAULT_SUM (&quot;Sum&quot;).</td>
</tr>
<tr>
<td>As String * ESB_MBRNAMELEN</td>
<td>DefaultCountMbrName</td>
<td>The name used with the attribute calculations (aggregate) dimension to indicate COUNT. The default value is ESB_DEFAULT_COUNT (&quot;Count&quot;).</td>
</tr>
<tr>
<td>As String * ESB_MBRNAMELEN</td>
<td>DefaultAverageMbrName</td>
<td>The name used with the attribute calculations (aggregate) dimension to indicate AVERAGE. The default value is ESB_DEFAULT_AVERAGE (&quot;Average&quot;).</td>
</tr>
<tr>
<td>As String * ESB_MBRNAMELEN</td>
<td>DefaultMinMbrName</td>
<td>The name used with the attribute calculations (aggregate) dimension to indicate MINIMUM. The default value is ESB_DEFAULT_MIN (&quot;Min&quot;).</td>
</tr>
<tr>
<td>As String * ESB_MBRNAMELEN</td>
<td>DefaultMaxMbrName</td>
<td>The name used with the attribute calculations (aggregate) dimension to indicate MAXIMUM. The default value is ESB_DEFAULT_MAX (&quot;Max&quot;).</td>
</tr>
<tr>
<td>VB Data Type</td>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| As Integer   | GenNameBy  | A constant identifier indicating whether to use the generation(s) of the zero-level member as the prefix or the suffix when generating a long name:  
  - ESB_GENNAMEBY_PREFIX (the default value)  
  - ESB_GENNAMEBY_SUFFIX |
| As Integer   | UseNameOf  | A constant identifier indicating which generation(s) of the zero-level member to use when generating a long name:  
  - ESB_USENAMEOF_NONE (the default value)  
  - ESB_USENAMEOF_PARENT  
  - ESB_USENAMEOF_GRANDPARENTANDPARENT  
  - ESB_USENAMEOF_ALLANCESTORS  
  - ESB_USENAMEOF_DIMENSION |
| As Integer   | Delimiter  | A constant identifier indicating the delimiter to use when generating a long name:  
  - ESB_DELIMITER_UNDERSCORE (the default value)  
  - ESB_DELIMITER_PIPE  
  - ESB_DELIMITER_CARET |
| As Integer   | DateFormat | A constant identifier indicating the format for a datetime attribute:  
  - ESB_DATEFORMAT_MMDDYYYY (the default value)  
  - ESB_DATEFORMAT_DDMMYYYY |
| As Integer   | BucketingType | A constant identifier indicating a numeric attribute's bucketing type:  
  - ESB_UPPERBOUNDINCLUSIVE (the default value)  
  - ESB_UPPERBOUNDNONINCLUSIVE  
  - ESB_LOWERBOUNDINCLUSIVE  
  - ESB_LOWERBOUNDNONINCLUSIVE |

**ESB_DBFILEINFO_T**

This structure contains information on an index or data file retrieved by EsbListDbFiles.

Type ESB_DBFILEINFO_T  
AppName         As String * ESB_APPNAMELEN  
DbName          As String * ESB_DBNAMELEN  
FilePath        As String * ESB_FILENAMELEN  
FileSize        As Long  
FileSequenceNum As Long  
FileCount       As Long  
FileType        As Integer  
FileOpen        As Integer  
End Type

<table>
<thead>
<tr>
<th>VB Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>As String * ESB_APPNAMELEN</td>
<td>AppName</td>
<td>Application name</td>
</tr>
</tbody>
</table>
## ESB_DBINFO_T

This database Info Structure gets information about a specific database. Fields in this structure cannot be modified using the VB API. See also “ESB_DBSTATE_T” on page 1173, which contains additional database state parameters that can be modified, and “ESB_DBSTATS_T” on page 1176. The fields are:

```vbnet
Type ESB_DBINFO_T
    ElapsedDbTime As Long
    DataCacheSize As Long
    IndexCacheSize As Long
    IndexPageSize As Long
    nDims As Long
    DbType As Integer
    status As Integer
    nConnects As Integer
    nLocks As Integer
    Data As Integer
    AppName As String * ESB_APPNAMELEN
    Name As String * ESB_DBNAMELEN
    Country As String * ESB_MBRNAMELEN
    Time As String * ESB_MBRNAMELEN
    Category As String * ESB_MBRNAMELEN
    Type As String * ESB_MBRNAMELEN
    CrPartition As String * ESB_MBRNAMELEN
End Type
```

### VB Data Type | Field | Description
---|---|---
As long | ElapsedDbTime | Number of seconds the database has been loaded
<table>
<thead>
<tr>
<th>VB Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>As long</td>
<td>DataCacheSize</td>
<td>Run-time data cache size (in KB) currently in use by database. Note that once you have changed the data cache size you must stop and restart the database in order for the new data cache size to take effect.</td>
</tr>
<tr>
<td>As long</td>
<td>IndexCacheSize</td>
<td>Run-time index cache size (in KB) currently in use by database</td>
</tr>
<tr>
<td>As long</td>
<td>IndexPageSize</td>
<td>Run-time index page size (in KB) currently in use by database</td>
</tr>
<tr>
<td>As Long</td>
<td>nDims</td>
<td>The number of dimensions in database</td>
</tr>
<tr>
<td>As Integer</td>
<td>DbType</td>
<td>Database type (normal or currency). This field can contain the following values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESB_DBTYPE_NORMAL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESB_DBTYPE_CURRENCY</td>
</tr>
<tr>
<td>As Integer</td>
<td>Status</td>
<td>The database load status (loaded or not loaded) - one of:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESB_STATUS_NOTLOADED</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESB_STATUS&gt;Loading</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESB_STATUS_LOADED</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESB_STATUS_UNLOADING</td>
</tr>
<tr>
<td>As Integer</td>
<td>nConnects</td>
<td>Number of users currently connected to the database</td>
</tr>
<tr>
<td>As Integer</td>
<td>nLocks</td>
<td>The number of data blocks currently exclusively locked</td>
</tr>
<tr>
<td>As Integer</td>
<td>Data</td>
<td>Flag to indicate the loading state of the database (either no data is loaded, data has been loaded but not calculated, or data is loaded and calculated). This field may contain one of the following values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESB_DBDATA_NONE /* no data */</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESB_DBDATA_LOADNOCALC /* data loaded without calc */</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESB_DBDATA_CLEAN /* data has been calculated */</td>
</tr>
<tr>
<td>As String *</td>
<td>AppName</td>
<td>Associated application name</td>
</tr>
<tr>
<td>ESB_APPNAMELEN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>As String *</td>
<td>Name</td>
<td>Database name</td>
</tr>
<tr>
<td>ESB_DBNAMELEN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>As String *</td>
<td>Country</td>
<td>The currency country dimension member, if any. If none, this field is an empty string.</td>
</tr>
<tr>
<td>ESB_MBRNAMELEN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>As String *</td>
<td>Time</td>
<td>The currency time dimension member, if any. If none, this field is an empty string.</td>
</tr>
<tr>
<td>ESB_MBRNAMELEN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>As String *</td>
<td>Category</td>
<td>The currency category dimension member, if any. If none, this field is an empty string.</td>
</tr>
<tr>
<td>ESB_MBRNAMELEN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>As String *</td>
<td>Type</td>
<td>The currency type dimension member (currency databases only). If none exists, this field is an empty string.</td>
</tr>
<tr>
<td>ESB_MBRNAMELEN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>As String *</td>
<td>CrPartition</td>
<td>The currency partition member (non-currency databases only)</td>
</tr>
<tr>
<td>ESB_MBRNAMELEN</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
ESB_DBREQINFO_T

This structure is used by EssGetDatabaseInfo. Essbase has three types of requests for which information exists, data load, calculation, and outline update. The following Essbase API constants identify each type of request:

- ESB_DBREQTYPE_DATLOAD 0 Data Load
- ESB_DBREQTYPE_CALC 1 Calculation
- ESB_DBREQTYPE_OTLUPD 2 Outline Update

The fields are:

```vba
type ESB_DBREQINFO_T
    DbReqType     As Long
    DbReqFlags    As Long
    StartTimeRec  As ESB_TIMERECORD_T
    EndTimeRec    As ESB_TIMERECORD_T
    User          As String * ESB_USERNAMELEN
end type
```

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>As Long</td>
<td>DbReqType</td>
<td>Type of database request</td>
</tr>
<tr>
<td>As Long</td>
<td>DbReqFlags</td>
<td>Bit map of information flags</td>
</tr>
<tr>
<td>As &quot;ESB_TIMERECORD_T&quot; on page 1191</td>
<td>StartTimeRec</td>
<td>Request to start time record</td>
</tr>
<tr>
<td>As &quot;ESB_TIMERECORD_T&quot; on page 1191</td>
<td>EndTimeRec</td>
<td>Request to end time record</td>
</tr>
<tr>
<td>As String * ESB_USERNAMELEN</td>
<td>User</td>
<td>User name</td>
</tr>
</tbody>
</table>

ESB_DBSTATE_T

This database State Structure gets and sets the state parameters for a specific database. All fields in this structure can be modified using the VB API. See also the “ESB_DBINFO_T” on page 1171 and “ESB_DBSTATS_T” on page 1176 structures, which contain additional database information that cannot be modified.

```vba
type ESB_DBSTATE_T
    Description         As String * ESB_DESCLEN
    Loadable            As Integer
    Autoload            As Integer
    Access              As Integer
    IndexType           As Integer
    MaxMem              As Long
    MaxCompMem          As Long
    MaxMemIndex         As Long
    IndexPageSize       As Long
    CalcNoAggMissing    As Integer
    CalcNoAvgMissing    As Integer
end type
```
The fields are:

<table>
<thead>
<tr>
<th>VB Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>As String * ESB_DESCLEN</td>
<td>Description</td>
<td>Database description (up to 80 characters)</td>
</tr>
<tr>
<td>As Integer</td>
<td>Loadable</td>
<td>Flag to indicate whether the database may be loaded (ESB_TRUE if the database is loadable)</td>
</tr>
<tr>
<td>As Integer</td>
<td>Autoload</td>
<td>Flag to indicate whether the database will automatically be loaded when the application is started (ESB_TRUE if the database will be automatically loaded)</td>
</tr>
<tr>
<td>As Integer</td>
<td>Access</td>
<td>Default access level to the database. See Table 15 for a list of values this field can contain.</td>
</tr>
<tr>
<td>As Integer</td>
<td>IndexType</td>
<td>Database index type (array or tree). Values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESB_INDEXTYPE_ARRAY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESB_INDEXTYPE_AVL</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note:</strong> For API Releases 4 and later, the IndexType field is obsolete.</td>
</tr>
<tr>
<td>As Long</td>
<td>MaxMem</td>
<td>Maximum memory reserved for non-compressed data blocks in the database (in bytes)</td>
</tr>
<tr>
<td>As Long</td>
<td>MaxCompMem</td>
<td>Maximum memory reserved for compressed data blocks in database (in bytes)</td>
</tr>
<tr>
<td>As Long</td>
<td>MaxMemIndex</td>
<td>Minimum index cache size. Value: 1048576. Set using the constant ESB_INDEXCACHEMIN_SIZE</td>
</tr>
<tr>
<td>As Long</td>
<td>IndexPageSize</td>
<td>Size of index page in which buffer pool is constructed in (in bytes).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Minimum index page size. Value: 1024. Set using the constant ESB_INDEXPAGEMIN_SIZE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Maximum page size for the IndexPageSize field. Value: 8192. Set using the constant ESB_INDEXPAGEMAX_SIZE</td>
</tr>
<tr>
<td>As Integer</td>
<td>CalcNoAgg Missing</td>
<td>Flag to suppress aggregation of members if all their children are missing (ESB_TRUE if missing values are not aggregated)</td>
</tr>
<tr>
<td>As Integer</td>
<td>CalcNoAvg Missing</td>
<td>Flag to suppress inclusion of missing members in calculating averages (ESB_TRUE if missing values are not included)</td>
</tr>
<tr>
<td>VB Data Type</td>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------</td>
<td>-------------</td>
</tr>
<tr>
<td>As Integer</td>
<td>CalcTwoPass</td>
<td>Flag to force two pass calculation when running full calculation of database (ESB_TRUE if two pass calculation is enabled)</td>
</tr>
<tr>
<td>As Integer</td>
<td>CalcCreate Block</td>
<td>Flag to force creation of data block on constant assignment calc equation (only valid for sparse dimensions). Set to ESB_TRUE if blocks are forcibly created.</td>
</tr>
<tr>
<td>As String * ESB_NAME *</td>
<td>CrDbName</td>
<td>The name of associated currency database (valid in non-currency databases)</td>
</tr>
<tr>
<td>As String * ESB_NAME *</td>
<td>CrTypeMember</td>
<td>The name of Currency Conversion type member (valid in non-currency databases)</td>
</tr>
<tr>
<td>As Integer</td>
<td>CrConvType</td>
<td>Currency Conversion type (whether currency conversions are calculated by multiplication or division). Values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESB_CRCTYPE_DIV</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESB_CRCTYPE_MULT</td>
</tr>
<tr>
<td>As Integer</td>
<td>DataCompress</td>
<td>Optional Flag to determine whether to compress blocks for this database.</td>
</tr>
<tr>
<td>As Long</td>
<td>RetrievalBuffer</td>
<td>Specifies the size, in kilobytes, of the server buffer that holds extracted row data cells before they are evaluated by the RESTRICT, TOP, or BOTTOM commands. The default is 2048 bytes.</td>
</tr>
<tr>
<td>As Long</td>
<td>RetrievalSortBuffer</td>
<td>Specifies the size, in kilobytes, of the server buffer that holds the data to be sorted during a retrieval. The default is 10240 bytes.</td>
</tr>
<tr>
<td>As Long</td>
<td>TimeOut</td>
<td>The timeout interval in seconds. This can only be set for COMMITTED access.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-1 is Indefinite wait.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 is Immediate access, no wait (the default).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>n is the specified interval in seconds.</td>
</tr>
<tr>
<td>As Long</td>
<td>CommitBlocks</td>
<td>The number of data blocks modified before performing the explicit commit (only used when isolation level is UNCOMMITTED).</td>
</tr>
<tr>
<td>As Long</td>
<td>CommitRows</td>
<td>The number of rows of the input file to data load before performing the explicit commit (only used when isolation level is UNCOMMITTED).</td>
</tr>
<tr>
<td>As Long</td>
<td>nVolumes</td>
<td>The number of disk volume settings for this database.</td>
</tr>
<tr>
<td>As Integer</td>
<td>DataCompressType</td>
<td>The data compression type used for write operations if the optional compression flag is set.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Bitmap - uses a bitmap to represent data cells (the default).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Run-Length Encoding - compresses any consecutive repetitive values.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- No Compression - does not compress the data.</td>
</tr>
<tr>
<td>As Integer</td>
<td>IsolationLevel</td>
<td>The isolation level:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- COMMITTED - write locks on all affected data blocks restrict access until the transaction commits.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- UNCOMMITTED - (default) write locks are acquired and released as needed during the transaction.</td>
</tr>
</tbody>
</table>
### ESB_DBSTATS_T

This database Statistics Structure returns run-time statistical information about a specific database. Fields in this structure cannot be modified using the VB API. See also the "ESB_DBSTATE_T" on page 1173 structure, which contains additional database state parameters that can be modified, and the “ESB_DBINFO_T” on page 1171 structure. The fields are:

**Type** ESB_DBSTATS_T

```vbnet
    nDims       As Long
    DeclaredBlockSize  As Long
    ActualBlockSize   As Long
    DeclaredMaxBlocks As Double
    ActualMaxBlocks   As Double
    NonMissingLeafBlocks As Double
    NonMissingNonLeafBlocks As Double
    NonMissingBlocks  As Double
    PagedOutBlocks   As Double
    PagedInBlocks    As Double
    InMemCompBlocks  As Double
    TotalBlocks      As Double
    NonExclusiveLockCount As Double
    ExclusiveLockCount As Double
    TotMemPagedInBlocks As Double
    TotMemBlocks     As Double
    TotMemIndex      As Double
    TotMemInMemCompBlocks As Double
    BlockDensity     As Double
    SparseDensity    As Double
    CompressionRatio As Double
    IndexType        As Integer
    ClusterRatio;     As Double
End Type
```

### VB Data Type

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>As Integer</td>
<td>PreImage</td>
</tr>
<tr>
<td></td>
<td>The flag to read previously committed data during read-only requests. This flag can only be set for COMMITTED access. The default is YES.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>VB Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>As Integer</td>
<td>IndexType</td>
<td>The database index type (array or tree). This field can contain the following values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESB_INDEXTYPE_ARRAY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESB_INDEXTYPE_AVL</td>
</tr>
<tr>
<td>As Long</td>
<td>nDims</td>
<td>Number of dimensions</td>
</tr>
<tr>
<td>As Long</td>
<td>DeclaredBlockSize</td>
<td>The declared data block size</td>
</tr>
<tr>
<td>As Long</td>
<td>ActualBlockSize</td>
<td>The actual data block size</td>
</tr>
<tr>
<td>As Double</td>
<td>DeclaredMaxBlocks</td>
<td>The declared maximum number of blocks in the database</td>
</tr>
<tr>
<td>VB Data Type</td>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>As Double</td>
<td>ActualMaxBlocks</td>
<td>The actual maximum number of blocks in the database</td>
</tr>
<tr>
<td>As Double</td>
<td>NonMissingLeafBlocks</td>
<td>The number of non-missing leaf (lowest level) blocks in the database</td>
</tr>
<tr>
<td>As Double</td>
<td>NonMissingNonLeafBlocks</td>
<td>The number of non-missing, non-leaf (upper level) blocks in the database</td>
</tr>
<tr>
<td>As Double</td>
<td>NonMissingBlocks</td>
<td>The total number of non-missing blocks in the database</td>
</tr>
<tr>
<td>As Double</td>
<td>PagedOutBlocks</td>
<td>The number of database blocks currently paged out to disk</td>
</tr>
<tr>
<td>As Double</td>
<td>PagedInBlocks</td>
<td>The total number of database blocks currently paged into memory</td>
</tr>
<tr>
<td>As Double</td>
<td>TotalBlocks</td>
<td>Total number of existing data blocks (not the maximum)</td>
</tr>
<tr>
<td>As Double</td>
<td>NonExclusiveLockCount</td>
<td>The number of database blocks currently non-exclusively locked</td>
</tr>
<tr>
<td>As Double</td>
<td>ExclusiveLockCount</td>
<td>The number of database blocks currently exclusively locked</td>
</tr>
<tr>
<td>As Double</td>
<td>TotMemBlocks</td>
<td>The total memory used for all database blocks</td>
</tr>
<tr>
<td>As Double</td>
<td>TotMemIndex</td>
<td>The total memory used for the database index</td>
</tr>
<tr>
<td>As Double</td>
<td>TotMemPagedInBlocks</td>
<td>The total memory used for all paged-in (uncompressed) database blocks</td>
</tr>
<tr>
<td>As Double</td>
<td>BlockDensity</td>
<td>The average database block density (calculated using all currently loaded blocks)</td>
</tr>
<tr>
<td>As Double</td>
<td>SparseDensity</td>
<td>Average density of the sparse dimensions in the database</td>
</tr>
<tr>
<td>As Double</td>
<td>CompressionRatio</td>
<td>Average data block compression ratio on the disk</td>
</tr>
<tr>
<td>As Double</td>
<td>InMemCompBlocks</td>
<td>The number of database blocks currently paged into compressed memory</td>
</tr>
<tr>
<td>As Double</td>
<td>TotMemInMemCompBlocks</td>
<td>The total memory used for database blocks currently paged into compressed memory</td>
</tr>
<tr>
<td>As Double</td>
<td>ClusterRatio</td>
<td>A measure of the fragmentation of the page file. A value close to 1 indicates the degree of fragmentation is low. A value close to zero indicates a high degree of fragmentation that could affect calculation and query performance.</td>
</tr>
</tbody>
</table>

### ESB_DIMENSIONINFO_T

This structure is used in `EsbGetDimensionInfo()`. The fields are:

```vbnet
Type ESB_DIMENSIONINFO_T
    DimName As String * ESB_MBRNAMELEN
    DimNumber As Long
    DimType As Integer
    DimTag As Integer
    DeclaredDimSize As Long
    ActualDimSize As Long
    Description As String * ESB_DESCLEN
    DimDataType As Integer
End Type
```
<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>As String * ESB_MBRNAMELEN</td>
<td>DimName</td>
<td>Dimension name</td>
</tr>
<tr>
<td>As Long</td>
<td>DimNumber</td>
<td>Dimension number of the member</td>
</tr>
</tbody>
</table>
| As Integer | DimType | Dimension type. Values:  
|  |  | * ESB_DIMTYPE_DENSE  
|  |  | * ESB_DIMTYPE_SPARSE |
| As Integer | DimTag | Dimension tag type. Values:  
|  |  | * ESB_TTYPE_ATTRCALC  
|  |  | * ESB_TTYPE_ATTRIBUTE  
|  |  | * ESB_TTYPECATEGORY  
|  |  | * ESB_TTYPE_CNAME  
|  |  | * ESB_TTYPE_CTIME  
|  |  | * ESB_TTYPE_NONE  
|  |  | * ESB_TTYPE_PARTITION  
|  |  | * ESB_TTYPE_TYPE |
| As Long | DeclaredDimSize | Declare dimension size |
| As Long | ActualDimSize | Actual dimension size |
| As String * ESB_DESCLEN | Description | Reserved: Not currently supported |
| As Integer | DimDataType | Attribute dimension data type. Values:  
|  |  | * ESB_ATTRMBRDT_BOOL  
|  |  | * ESB_ATTRMBRDT_DATETIME  
|  |  | * ESB_ATTRMBRDT_DOUBLE  
|  |  | * ESB_ATTRMBRDT_STRING |

**ESB_DIMSTATS_T**

This Dimension Statistical Structure gets information about a specific database dimension. Fields in this structure cannot be modified using the VB API. An array of these structures is generated when getting database statistics structure (EsbGetDbStats) to provide information about each dimension in the database. The fields are:

```vbnet
Type ESB_DIMSTATS_T
    DeclaredDimSize As Long
    ActualDimSize As Long
    DimType As Integer
    DimName As String * ESB_MBRNAMELEN
End Type
```
### ESB_DURLINFO_T

A data structure used to capture URL information. The fields are:

```vbnet
type ESB_DURLINFO_T
    blsLevel0  as integer  'consider level-0 members along symmetric regions
    iURLXMLSize as integer  'URL XML size
    cpURLName   as string * 1024  'URL identifier
    cpURLXML    as string * 8192  'URL XML
end type
```

<table>
<thead>
<tr>
<th>Visual Basic Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>As Integer</td>
<td>blsLevel0</td>
<td>If 1, then URL definition is restricted to level-0 data; if 0, there is no restriction</td>
</tr>
<tr>
<td>As Integer</td>
<td>iURLXMLSize</td>
<td>Size of URL XML</td>
</tr>
<tr>
<td>As String * 1024</td>
<td>cpURLName</td>
<td>Name of URL definition</td>
</tr>
<tr>
<td>As String * 8192</td>
<td>cpURLXML</td>
<td>Content of URL XML</td>
</tr>
</tbody>
</table>

**Note:** The regions list is passed as a separate argument, `symRegions()`, within each Visual Basic drill-through function.

### ESB_GLOBAL_T

This structure contains global server system parameters used for administrative purposes. All of the fields in this structure can be modified using the VB API. The fields are:

```vbnet
type ESB_GLOBAL_T
    Security   as integer
    Logins     as integer
    Access     as integer
    Validity   as integer
    Currency   as integer
    PwMin      as integer
end type
```

Visual Basic Main API Structures 1179
InactivityTime As Long
InactivityCheck As Long
End Type

<table>
<thead>
<tr>
<th>VB Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>As String * 1</td>
<td>Security</td>
<td>Flag to indicate whether global security is enabled (default is ESB_TRUE, security enabled)</td>
</tr>
<tr>
<td>As String * 1</td>
<td>Logins</td>
<td>Flag to indicate whether user logins are enabled (default is ESB_TRUE, indicating logins are enabled)</td>
</tr>
<tr>
<td>As Integer</td>
<td>Access</td>
<td>The default access level for newly-created applications (default is ESB_ACCESS_NONE). For a list of possible values, see Table 15.</td>
</tr>
<tr>
<td>As Integer</td>
<td>Validity</td>
<td>The default password validity period (default is 365 days)</td>
</tr>
<tr>
<td>As String * 1</td>
<td>Currency</td>
<td>Flag to indicate whether currency option is supported (this flag is read only). Set to ESB_TRUE if the currency option is enabled.</td>
</tr>
<tr>
<td>As Integer</td>
<td>PwMin</td>
<td>The minimum permitted password length (default is 6 characters)</td>
</tr>
<tr>
<td>As Long</td>
<td>InactivityTime</td>
<td>Maximum length of time, in seconds, the user can be inactive before automatic logout from all applications and the Agent. Default value: 3600 seconds. Minimum value: 300 seconds. To disable auto logout, set InactivityTime to 0.</td>
</tr>
<tr>
<td>As Long</td>
<td>InactivityCheck</td>
<td>Frequency of checks for auto logout, in seconds. Default value: 300 seconds. Minimum value: 30 seconds. Must be smaller than InactivityTime setting or InactivityCheck is set to the value of InactivityTime and a warning message occurs. To disable auto logout, set InactivityCheck to 0.</td>
</tr>
</tbody>
</table>

**ESB_INIT_T**

This structure is passed to the VB API initialization function EsbInit() and contains fields that let API developers customize their usage of the API. If any of the fields of the structure are set to zero, the API defaults are used. The fields are:

Type ESB_INIT_T

- Version As Long
- MaxHandles As Integer
- LocalPath As String * ESB_PATHLEN
- MessageFile As String * ESB_PATHLEN
- HelpFile As String * ESB_PATHLEN
- ClientError As Integer
- ErrorStack As Integer
- vbCallbackFuncAddress As Long

End Type

<table>
<thead>
<tr>
<th>VB Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>As Long</td>
<td>Version</td>
<td>Version of Essbase API used to compile the application. Used for backward compatibility.</td>
</tr>
<tr>
<td>As Integer</td>
<td>MaxHandles</td>
<td>The maximum number of simultaneous context handles required by the application (between 1 and 10)</td>
</tr>
<tr>
<td>As String * 1</td>
<td>LocalPath</td>
<td>The default local path name to use for file and object operations on the client</td>
</tr>
<tr>
<td>ESB_PATHLEN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VB Data Type</td>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>As String *</td>
<td>MessageFile</td>
<td>The qualified path name of the message database file, ESSBASE.MDB</td>
</tr>
<tr>
<td>ESB_PATHLEN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>As String *</td>
<td>HelpFile</td>
<td>Fully-qualified path name of the user-defined application help file, used</td>
</tr>
<tr>
<td>ESB_PATHLEN</td>
<td></td>
<td>for help for the AutoLogin dialog box. The login help context must be</td>
</tr>
<tr>
<td></td>
<td></td>
<td>defined in the help file. See Chapter 3, “Integrating Essbase With Your</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Product.” By default, clicking the Help button displays the Essbase System</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Login help topic shipped with the Oracle Essbase Spreadsheet Add-in User’s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Guide online help. If ESSBASEPATH is not defined, the help file name is set</td>
</tr>
<tr>
<td></td>
<td></td>
<td>to null.</td>
</tr>
<tr>
<td>As String *</td>
<td>ClientError</td>
<td>ESB_FALSE to use a default error handler, ESB_TRUE to use EsbGetMessage to</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>retrieve messages</td>
</tr>
<tr>
<td>As Integer</td>
<td>ErrorStack</td>
<td>A size of the message stack used by EsbGetMessage, default is 100</td>
</tr>
<tr>
<td>As Long</td>
<td>vbCallbackFuncAddress</td>
<td>AddressOf custom Visual Basic callback function. For more information, see “Visual Basic API Message Handling” on page 1123.</td>
</tr>
</tbody>
</table>

**ESB_LOCKINFO_T**

This structure contains information about data blocks exclusively locked, as returned by the EsbListLocks() function. Fields in this structure cannot be modified using the VB API. The fields are:

Type ESB_LOCKINFO_T

    LoginId   As Long
    Time      As Long
    nLocks    As Integer
    userName  As String * ESB_USERNAMELEN

**ESB_MBRALT_T**

This structure contains information about alternate member name. Fields in this structure cannot be modified using the VB API. The fields are:

Type ESB_MBRALT_T

    MbrName   As String * ESB_MBRNAMELEN
AltName   As String * ESB_MBRNAMELEN
End Type

<table>
<thead>
<tr>
<th>VB Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>As String * ESB_MBRNAMELEN</td>
<td>MbrName</td>
<td>The member name</td>
</tr>
<tr>
<td>As String * ESB_MBRNAMELEN</td>
<td>AltName</td>
<td>The associated alias name (ESB_MBRNAME_T)</td>
</tr>
</tbody>
</table>

**ESB_MBRUSER_T**

This structure contains information about an SQL user name and password. Fields in this structure cannot be modified using the VB API. The fields are:

Type ESB_MBRUSER_T

User          As String * ESB_USERNAMELEN
Password      As String * ESB_PASSWORDLEN
End Type

<table>
<thead>
<tr>
<th>VB Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>As String * ESB_USERNAMELEN</td>
<td>User</td>
<td>SQL database user name</td>
</tr>
<tr>
<td>As String * ESB_PASSWORDLEN</td>
<td>Password</td>
<td>SQL database user password</td>
</tr>
</tbody>
</table>

**ESB_MEMBERINFO_T**

This structure contains information about a specific member. Fields in this structure cannot be modified using the VB API. The fields are:

Type ESB_MEMBERINFO_T

CrMbrName     As String * ESB_MBRNAMELEN
MbrName       As String * ESB_MBRNAMELEN
DimName       As String * ESB_MBRNAMELEN
ParentMbrName As String * ESB_MBRNAMELEN
ChildMbrName  As String * ESB_MBRNAMELEN
PrevMbrName   As String * ESB_MBRNAMELEN
NextMbrName   As String * ESB_MBRNAMELEN
Description   As String * ESB_DESCLEN
MbrNumber     As Long
DimNumber     As Long
status        As Integer
Level         As Integer
Generation    As Integer
UnaryCalc     As Integer
MbrTagType    As Integer
CurrConvert   As Integer
Attribute     As Variant
IsAttributed  As Integer
End Type
<table>
<thead>
<tr>
<th>VB Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>As String * ESB_MBRNAMELEN</td>
<td>CrMbrName</td>
<td>The name of the tagged currency database member. For the Time dimension, gives the name of the tagged time member; for the Country dimension, gives the name of the tagged currency member; for the Accounts dimension, gives the name of the tagged category member.</td>
</tr>
<tr>
<td>As String * ESB_MBRNAMELEN</td>
<td>MbrName</td>
<td>The member name</td>
</tr>
<tr>
<td>As String * ESB_MBRNAMELEN</td>
<td>DimName</td>
<td>The member's dimension name (ESB_MBRNAMELEN)</td>
</tr>
<tr>
<td>As String * ESB_MBRNAMELEN</td>
<td>ParentMbrName</td>
<td>The specified member's parent name or an empty string if the member has no parent</td>
</tr>
<tr>
<td>As String * ESB_MBRNAMELEN</td>
<td>ChildMbrName</td>
<td>The specified member's first child member name</td>
</tr>
<tr>
<td>As String * ESB_MBRNAMELEN</td>
<td>PrevMbrName</td>
<td>The specified member's previous sibling member name</td>
</tr>
<tr>
<td>As String * ESB_MBRNAMELEN</td>
<td>NextMbrName</td>
<td>The specified member's next sibling member name</td>
</tr>
<tr>
<td>As String * ESB_DESCLEN</td>
<td>Description</td>
<td>The member description</td>
</tr>
<tr>
<td>As Long</td>
<td>MbrNumber</td>
<td>The member number</td>
</tr>
<tr>
<td>As Long</td>
<td>DimNumber</td>
<td>The member's dimension number</td>
</tr>
<tr>
<td>As Integer</td>
<td>Status</td>
<td>The member's share status is derived by performing a logical AND between the contents of this field and each of the constant values of the form ESB_MBRSTS_xxx:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESB_MBRSTS_NOTSET</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESB_MBRSTS_NEVER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESB_MBRSTS_LABEL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESB_MBRSTSREFER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESB_MBRSTS_REFNME</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESB_MBRSTS_SHARE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESB_MBRSTS_VIRTSTORE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESB_MBRSTS_VIRTNOSTORE</td>
</tr>
<tr>
<td>As Integer</td>
<td>Level</td>
<td>The member level number (zero-based), counting up from the lowest descendent of the specified member</td>
</tr>
<tr>
<td>As Integer</td>
<td>Generation</td>
<td>The member generation number (one-based), counting down from the specified member's dimension member</td>
</tr>
<tr>
<td>As Integer</td>
<td>UnaryCalc</td>
<td>The default unary rollup for this member (one of the constant values of the form ESB_UCAL_xxx). Can be add, subtract, multiply, divide, percent, or none</td>
</tr>
<tr>
<td>As Integer</td>
<td>MbrTagType</td>
<td>A 16 bit mask for member's tagged types (masks are of the form ESB_ATYPE_xxx)</td>
</tr>
<tr>
<td>As String *1</td>
<td>CurrConvert</td>
<td>Currency Conversion. This field can contain the values ESB_TRUE and ESB_FALSE</td>
</tr>
</tbody>
</table>
## ESB_OBJDEF_T

This structure contains information about a specific file object. Fields in this structure cannot be modified using the VB API. The fields are:

- **Type ESB_OBJDEF_T**
  - `hCtx` As Long
  - `Type` As Long
  - `AppName` As String * ESB_APPNAMELEN
  - `DbName` As String * ESB_DBNAMELEN
  - `FileName` As String * ESB_PATHLEN

### VB Data Type

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>hCtx</code></td>
<td>VB API context handle</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Type</code></td>
<td>Object type. See Table 15, “Bitmask Data Types,” on page 1161 for a list of object types.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>AppName</code></td>
<td>Application name</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>DbName</code></td>
<td>Database name</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>FileName</code></td>
<td>Object's file name. A local file name when the following apply:</td>
</tr>
<tr>
<td>- <code>hCtx</code> is a local context handle</td>
<td></td>
</tr>
<tr>
<td>- <code>AppName</code> and <code>DbName</code> are empty strings</td>
<td></td>
</tr>
<tr>
<td>- <code>FileName</code> points to the full path name of a local file.</td>
<td></td>
</tr>
</tbody>
</table>

## ESB_OBJINFO_T

This structure contains information about a specific file object. Fields in this structure cannot be modified using the API. The fields are:

- **ESB_OBJINFO_T**
  - `hCtx` As Long
  - `Type` As Long
  - `AppName` As String * ESB_APPNAMELEN
  - `DbName` As String * ESB_DBNAMELEN
  - `FileName` As String * ESB_PATHLEN

### VB Data Type

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>hCtx</code></td>
<td>VB API context handle</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Type</code></td>
<td>Object type. See Table 15, “Bitmask Data Types,” on page 1161 for a list of object types.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>AppName</code></td>
<td>Application name</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>DbName</code></td>
<td>Database name</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>FileName</code></td>
<td>Object's file name. A local file name when the following apply:</td>
</tr>
<tr>
<td>- <code>hCtx</code> is a local context handle</td>
<td></td>
</tr>
<tr>
<td>- <code>AppName</code> and <code>DbName</code> are empty strings</td>
<td></td>
</tr>
<tr>
<td>- <code>FileName</code> points to the full path name of a local file.</td>
<td></td>
</tr>
</tbody>
</table>
Type ESB_OBJINFO_T

AppName As String * ESB_APPNAMELEN
DbName As String * ESB_DBNAMELEN
Name As String * ESB_OBJNAMELEN
Type As Long
FileSize As Long
TimeStamp As Long
TimeModified As ESB_TIMERECORD_T
User As String * ESB_USERNAMELEN
Locked As Integer

End Type

<table>
<thead>
<tr>
<th>VB Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>As String * ESB_OBJNAMELEN</td>
<td>Name</td>
<td>Object name</td>
</tr>
<tr>
<td>As Long</td>
<td>Type</td>
<td>Object type. See Table 15, “Bitmask Data Types,” on page 1161 for a list of object types.</td>
</tr>
<tr>
<td>As String * ESB_APPNAMELEN</td>
<td>AppName</td>
<td>Application name</td>
</tr>
<tr>
<td>As String * ESB_DBNAMELEN</td>
<td>DbName</td>
<td>Database name</td>
</tr>
<tr>
<td>As Long</td>
<td>FileSize</td>
<td>Object’s allocated file size (in bytes)</td>
</tr>
<tr>
<td>As String * 1</td>
<td>Locked</td>
<td>Flag to indicate whether object is locked (ESB_TRUE indicates the object is locked)</td>
</tr>
<tr>
<td>As String * ESB_USERNAMELEN</td>
<td>User</td>
<td>Name of the user who has the object locked (if locked), otherwise undefined</td>
</tr>
<tr>
<td>As Long</td>
<td>TimeStamp</td>
<td>Date and time object was locked (if locked), otherwise undefined</td>
</tr>
<tr>
<td>As ESB_TIMERECORD_T</td>
<td>Time Modified</td>
<td>Date and time of last modification</td>
</tr>
</tbody>
</table>

ESB_PART_CONNECT_INFO_T

This structure specifies a database.

Type ESB_PART_CONNECT_INFO_T

HostName As String * ESB_SVRNAMELEN
AppName As String * ESB_APPNAMELEN
DbName As String * ESB_DBNAMELEN

End Type

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>HostName</td>
<td>Host name.</td>
</tr>
<tr>
<td>String</td>
<td>AppName</td>
<td>Application name.</td>
</tr>
<tr>
<td>String</td>
<td>DbName</td>
<td>Database name.</td>
</tr>
</tbody>
</table>
ESB_PART_DEFINED_T

This structure specifies a shared partition.

Type ESB_PART_DEFINED_T

usType    As Integer  ' ESB_PARTITION_OP_REPLICATED, _LINKED, or _TRANSPARENT
Direction As Integer  ' ESB_PARTITION_DATA_SOURCE or _TARGET
HostDatabase As ESB_PART_CONNECT_INFO_T

End Type

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td>usType</td>
<td>One of the Operation Type constants listed below.</td>
</tr>
<tr>
<td>Integer</td>
<td>usDirection</td>
<td>One of the Direction constants listed below.</td>
</tr>
</tbody>
</table>

"ESB_PART_CONNECT_INFO_T" on page 1185

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HostDatabase</td>
<td>The host server.</td>
</tr>
</tbody>
</table>

Operation Type Constants

define ESB_PARTITION_OP_REPLICATED    0x0001
#define ESB_PARTITION_OP_LINKED         0x0002
#define ESB_PARTITION_OP_TRANSPARENT    0x0004
#define ESB_PARTITION_OP_ALLTYPES       (ESB_PARTITION_OP_REPLICATED | ESB_PARTITION_OP_LINKED | ESB_PARTITION_OP_TRANSPARENT)

Direction Constants

define ESB_PARTITION_DATA_SOURCE      0x0001
#define ESB_PARTITION_DATA_TARGET       0x0002
#define ESB_PARTITION_DATA_BOTH         (ESB_PARTITION_DATA_SOURCE | ESB_PARTITION_DATA_TARGET)

ESB_PART_INFO_T

This structure holds the shared partition information.

Type ESB_PART_INFO_T

OperationType    As Integer
DataDirection    As Integer
MetaDirection    As Integer
usReserved       As Integer
LastMetaUpdateTime As Long
LastRefreshTime   As Long
AreaUpdatable    As Integer
IncrRefreshAllowed As Integer
LastUpdateTime   As Long
SvrName          As String * ESB_SVRNAMELEN
AppName          As String * ESB_APPNAMELEN
```plaintext
**Data Type** | **Field** | **Description**
--- | --- | ---
Integer | OperationTypes | One of the Operation Type constants listed below.
Integer | DirectionTypes | One of the Direction constants listed below.
Integer | MetaDirectionTypes | One of the MetaDirection constants listed below.
Integer | usReserved | Reserved for future use - is set to 0.
Long | LastMetaUpdateTime | Last time meta data was updated.

**The following fields only apply to replication data targets**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Long LastRefreshTime</td>
<td>Last time data at target was refreshed.</td>
</tr>
<tr>
<td>Integer AreaUpdatable</td>
<td>Are changes allowed to replicated data?</td>
</tr>
</tbody>
</table>

**The following fields only apply to replication data sources**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer IncrRefreshAllowed</td>
<td>Can we refresh only the changed data?</td>
</tr>
<tr>
<td>Long LastUpdateTime</td>
<td>Time of last change to data in the partition.</td>
</tr>
</tbody>
</table>

**The following fields only apply to remote connections.**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String SvrName</td>
<td>Host for the other side of the partition definition.</td>
</tr>
<tr>
<td>StringAppName</td>
<td>Application for the other side of the partition definition.</td>
</tr>
<tr>
<td>String DbName</td>
<td>Database for other side of the partition definition; meta data change information.</td>
</tr>
</tbody>
</table>

**Operation Type Constants**

```c
#define ESB_PARTITION_OP_REPLICATED 0x0001
#define ESB_PARTITION_OP_LINKED 0x0002
#define ESB_PARTITION_OP_TRANSPARENT 0x0004
#define ESB_PARTITION_OP_ALLTYPES = ESB_PARTITION_OP_REPLICATED + ESB_PARTITION_OP_LINKED + ESB_PARTITION_OP_TRANSPARENT
```

**Direction Constants**

```c
#define ESB_PARTITION_DATA_SOURCE 0x0001
#define ESB_PARTITION_DATA_TARGET 0x0002
#define ESB_PARTITION_DATA_BOTH = ESB_PARTITION_DATA_SOURCE + ESB_PARTITION_DATA_TARGET
```

**MetaDirection Constants**

Global Const ESB_PARTITION_META_SOURCE = 0x0001  'Source metadata partitions
Global Const ESB_PARTITION_META_TARGET = 0x0002  'Target metadata partitions
Global Const ESB_PARTITION_META_BOTH = ESB_PARTITION_META_SOURCE + ESB_PARTITION_META_TARGET
```
**ESB_PART_REPL_T**

This structure queries shared partitions.

**Type** `ESB_PART_REPL_T`

```plaintext
    AreaCount   As Long
    UpdatedOnly As Integer
End Type
```

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Long</td>
<td><code>AreaCount</code></td>
<td>Number of partitions to refresh from (-1 == ALL)</td>
</tr>
<tr>
<td>Integer (Boolean)</td>
<td><code>UpdatedOnly</code></td>
<td>Refreshes only the cells modified at the source since the last refresh operation.</td>
</tr>
</tbody>
</table>

**ESB_PARTOTL_QRY_FILTER_T**

This structure further defines the meta data retrieval criteria.

**Type** `ESB_PARTOTL_QRY_FILTER_T`

```plaintext
    TimeStamp           As Long
    DimFilter           As Long
    MbrFilter           As Long
    MbrAttrFilter       As Long
End Type
```

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Long</td>
<td><code>TimeStamp</code></td>
<td>Query meta change happens after this time.</td>
</tr>
<tr>
<td>Long</td>
<td><code>DimFilter</code></td>
<td>Bitfield to select dimension changes.</td>
</tr>
<tr>
<td>Long</td>
<td><code>MbrFilter</code></td>
<td>Bitfield to select member changes.</td>
</tr>
<tr>
<td>Long</td>
<td><code>MbrAttrFilter</code></td>
<td>Bitfield to select member attribute changes.</td>
</tr>
</tbody>
</table>

**Member Attribute Change Constants (MbrAttrFilter)**

```plaintext
#define ESB_PARTITION_OTLMBRATTR_STATUS         0x0001  /* status changes */
#define ESB_PARTITION_OTLMBRATTR_ALIAS          0x0002  /* alias changes */
#define ESB_PARTITION_OTLMBRATTR_UCALC          0x0004  /* unary calc symbol changes */
#define ESB_PARTITION_OTLMBRATTR_ATYPE          0x0008  /* account type changes */
#define ESB_PARTITION_OTLMBRATTR_CCONVERT       0x0010  /* currency conversion flag */
#define ESB_PARTITION_OTLMBRATTR_CRMBRNAME      0x0020  /* tagged currency db member */
#define ESB_PARTITION_OTLMBRATTR_UDA            0x0040  /* user defined attribute changes */
#define ESB_PARTITION_OTLMBRATTR_CALC           0x0080  /* calc formula changes */
#define ESB_PARTITION_OTLMBRATTR_LEVEL          0x0100  /* level number changes */
#define ESB_PARTITION_OTLMBRATTR_GENERATION     0x0200  /* generation number changes */
#define ESB_PARTITION_OTLMBRATTR_ALL            (ESB_PARTITION_OTLMBRATTR_STATUS |
```
ESB_ALLCHG = (ESB_PARTITION_OTLMBR_ALL | ESB_DIMCHG_ALL)

### ESB_PARTOTL_QUERY_T

This structure queries metadata changes.

**Type** ESB_PARTOTL_QUERY_T

- **OperationType** As Integer ' ESB_PARTITION_OP_REPLICATED, _LINKED, _TRANSPARENT
- **HostDatabase** As ESB_PART_CONNECT_INFO_T
- **MetaFilter** As ESB_PARTOTL_QRY_FILTER_T

**Data Type** | **Field** | **Description**
--- | --- | ---
Integer | usOperationType | One of the Operation Type constants listed below.

**"ESB_PART_CONNECT_INFO_T" on page 1185** | **HostDatabase** | The host server database name.

**"ESB_PARTOTL_QRY_FILTER_T" on page 1188** | **MetaFilter** | Criteria to further define names.

### Operation Type Constants

- `#define ESB_PARTITION_OP_REPLICATED 0x0001`
- `#define ESB_PARTITION_OP_LINKED 0x0002`
- `#define ESB_PARTITION_OP_TRANSPARENT 0x0004`
- `#define ESB_PARTITION_OP_ALLTYPES (ESB_PARTITION_OP_REPLICATED | ESB_PARTITION_OP_LINKED | ESB_PARTITION_OP_TRANSPARENT)`

### ESB_PARTSLCT_T

This structure queries shared partitions for a given site.

**Type** ESB_PARTSLCT_T

- **OperationTypes** As Integer
- **DirectionTypes** As Integer
- **MetaDirectionTypes** As Integer

End Type
## Data Type | Field | Description
---|---|---
Integer | OperationTypes | One of the Operation Type Constants listed below.
Integer | DirectionTypes | One of the Direction Constants listed below.
Integer | MetaDirectionTypes | One of the MetaDirection Constants listed below.

### Operation Type Constants

```c
#define ESB_PARTITION_OP_REPLICATED 0x0001
#define ESB_PARTITION_OP_LINKED 0x0002
#define ESB_PARTITION_OP_TRANSPARENT 0x0004
#define ESB_PARTITION_OP_ALLTYPES = ESB_PARTITION_OP_REPLICATED
 + ESB_PARTITION_OP_LINKED
 + ESB_PARTITION_OP_TRANSPARENT)
```

### Direction Constants

```c
#define ESB_PARTITION_DATA_SOURCE 0x0001
#define ESB_PARTITION_DATA_TARGET 0x0002
#define ESB_PARTITION_DATA_BOTH = ESB_PARTITION_DATA_SOURCE
 + ESB_PARTITION_DATA_TARGET)
```

### MetaDirection Constants

```c
Global Const ESB_PARTITION_META_SOURCE = 0x0001 'Source metadata partitions
Global Const ESB_PARTITION_META_TARGET = 0x0002 'Target metadata partitions
Global Const ESB_PARTITION_META_BOTH = ESB_PARTITION_META_SOURCE
 + ESB_PARTITION_META_TARGET
```

### ESB_PROCSTATE_T

When you perform asynchronous operations (for example, a calculation), this structure is returned from calls to EsbGetProcessState(). This lets the caller determine the status of the asynchronous operation.

**Note:** In this release of the VB API, the State field is the only field implemented; all other fields are reserved for future use.

**Type** ESB_PROCSTATE_T

- Action As Integer
- State As Integer
- Reserved1 As Integer
- Reserved2 As Long
- Reserved3 As Long

### VB Data Type | Field | Description
---|---|---
As Integer | Action | Current process action (not used)
<table>
<thead>
<tr>
<th>VB Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>As Integer</td>
<td>State</td>
<td>Current process state (either done or in progress). Values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESB_STATE_DONE (0)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESB_STATE_INPROGRESS (1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESB_STATE_FINALSTAGE (5)</td>
</tr>
<tr>
<td>As Integer</td>
<td>Reserved1</td>
<td>Reserved for future use</td>
</tr>
<tr>
<td>As Long</td>
<td>Reserved2</td>
<td>Reserved for future use</td>
</tr>
<tr>
<td>As Long</td>
<td>Reserved3</td>
<td>Reserved for future use</td>
</tr>
</tbody>
</table>

**ESB_RATEINFO_T**

This structure contains information about currency rates. Fields in this structure cannot be modified using the VB API. The fields are:

```
Type ESB_RATEINFO_T
    MbrName       As String * ESB_MBRNAMELEN
    RateMbr       As String * ESB_RATEINFOLEN
End Type
```

<table>
<thead>
<tr>
<th>VB Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>As String * ESB_MBRNAMELEN</td>
<td>MbrName</td>
<td>The Member name (ESB_MBRNAMELEN)</td>
</tr>
<tr>
<td>As String * ESB_MBRNAMELEN</td>
<td>RateMbr</td>
<td>Array of rate member name (ESB_MBRNAME_T)</td>
</tr>
</tbody>
</table>

**ESB_TIMERECORD_T**

This structure is used in the “ESB_DBREQINFO_T” on page 1173 structure. The fields are:

```
Type ESB_TIMERECORD_T
    TimeValue   As Long
    Seconds     As Integer
    Minutes     As Integer
    Hours       As Integer
    Day         As Integer
    Month       As Integer
    Year        As Integer
    Weekday     As Integer
    Reserved    As Integer
End Type
```

<table>
<thead>
<tr>
<th>VB Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>As Long</td>
<td>TimeValue</td>
<td>Time value in seconds after 1/1/70</td>
</tr>
</tbody>
</table>
## ESB_USERAPP_T, ESB_GROUPAPP_T

This structure contains access privilege information for a user or group and a specific application. The `Access` and `MaxAccess` fields are the only fields in this structure that can be modified using the VB API. The fields are:

```vba
type ESB_USERAPP_T
    Access       As Integer
    MaxAccess    As Integer
    userName     As String  * ESB_USERNAMELEN
    AppName      As String  * ESB_APPNAMELEN
end type
```

<table>
<thead>
<tr>
<th>VB Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>As String</td>
<td>UserName</td>
<td>The user or group name (ESB_USERNAMELEN)</td>
</tr>
<tr>
<td>As String</td>
<td>AppName</td>
<td>The application name (ESB_APPNAMELEN)</td>
</tr>
<tr>
<td>As Integer</td>
<td>Access</td>
<td>The assigned access privilege to the application for the user or group. Values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESB_PRIV_NONE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESB_PRIV_APPLOAD</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESB_PRIV_APPDESIGN</td>
</tr>
<tr>
<td>As Integer</td>
<td>MaxAccess</td>
<td>The maximum access privilege to the application for the user or group from all sources</td>
</tr>
</tbody>
</table>

## ESB_USERDB_T, ESB_GROUPDB_T

This structure contains access privilege information for a user or group and a specific database. The `Access`, `MaxAccess`, and `Filter` fields are the only fields in this structure that can be modified using the VB API. The fields are:

```vba
type ESB_USERDB_T
    Access       As Integer
end type
```

<table>
<thead>
<tr>
<th>VB Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>As String</td>
<td>UserName</td>
<td>The user or group name (ESB_USERNAMELEN)</td>
</tr>
<tr>
<td>As String</td>
<td>AppName</td>
<td>The application name (ESB_APPNAMELEN)</td>
</tr>
<tr>
<td>As Integer</td>
<td>Access</td>
<td>The assigned access privilege to the application for the user or group. Values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESB_PRIV_NONE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESB_PRIV_APPLOAD</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESB_PRIV_APPDESIGN</td>
</tr>
<tr>
<td>As Integer</td>
<td>MaxAccess</td>
<td>The maximum access privilege to the application for the user or group from all sources</td>
</tr>
</tbody>
</table>

1192 Visual Basic Main API Declarations
MaxAccess As Integer  
AppName As String * ESB_APPNAMELEN  
DbName As String * ESB_DBNAMELEN  
userName As String * ESB_USERNAMELEN  
FilterName As String * ESB_FTRNAMELEN
End Type

<table>
<thead>
<tr>
<th>VB Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>As String * ESB_USERNAMELEN</td>
<td>UserName</td>
<td>User or group name (ESB_USERNAMELEN)</td>
</tr>
<tr>
<td>As String * ESB_APPNAMELEN</td>
<td>AppName</td>
<td>Application name (ESB_APPNAMELEN)</td>
</tr>
<tr>
<td>As String * ESB_DBNAMELEN</td>
<td>DbName</td>
<td>Database name (ESB_DBNAMELEN)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>As Integer</th>
<th>Access</th>
<th>The assigned access privilege to the database for the user or group. Values:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>• ESB_PRIV_NONE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESB_PRIV_READ</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESB_PRIV_WRITE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESB_PRIV_CALC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESB_PRIV_DBLOAD</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESB_PRIV_DBDESIGN</td>
</tr>
</tbody>
</table>

These values are a subset of the Table 15.

<table>
<thead>
<tr>
<th>As Integer</th>
<th>MaxAccess</th>
<th>Maximum access privilege to the database for the user or group from all sources</th>
</tr>
</thead>
<tbody>
<tr>
<td>As String * ESB_FTRNAMELEN</td>
<td>FilterName</td>
<td>Name of the assigned database filter, if any. If none, this field is an empty string.</td>
</tr>
</tbody>
</table>

**ESB_USERINFO_T, ESB_GROUPINFO_T**

This structure stores information about users or groups.

Type ESB_USERINFO_T

<table>
<thead>
<tr>
<th>LastLogin</th>
<th>As Long</th>
</tr>
</thead>
<tbody>
<tr>
<td>DbConnectTime</td>
<td>As Long</td>
</tr>
<tr>
<td>LoginId</td>
<td>As Long</td>
</tr>
<tr>
<td>Login</td>
<td>As Integer</td>
</tr>
<tr>
<td>Type</td>
<td>As Integer</td>
</tr>
<tr>
<td>Access</td>
<td>As Integer</td>
</tr>
<tr>
<td>MaxAccess</td>
<td>As Integer</td>
</tr>
<tr>
<td>Expiration</td>
<td>As Integer</td>
</tr>
<tr>
<td>FailCount</td>
<td>As Integer</td>
</tr>
<tr>
<td>Name</td>
<td>As String * ESB_USERNAMELEN</td>
</tr>
<tr>
<td>AppName</td>
<td>As String * ESB_APPNAMELEN</td>
</tr>
<tr>
<td>DbName</td>
<td>As String * ESB_DBNAMELEN</td>
</tr>
<tr>
<td>Description</td>
<td>As String * ESB_DESCLEN</td>
</tr>
<tr>
<td>EMailID</td>
<td>As String * ESB_DESCLEN</td>
</tr>
<tr>
<td>LockedOut</td>
<td>As Boolean</td>
</tr>
<tr>
<td>PwdChgNow</td>
<td>As Boolean</td>
</tr>
</tbody>
</table>
End Type
Some of the fields are specific to users and cannot be used for groups. The *Access*, *Expiration*, and *PwdChgNow* fields are the only fields in this structure that can be modified using the API. The fields are:

<table>
<thead>
<tr>
<th>VB Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>As Long</td>
<td>LastLogin</td>
<td>Date of the user's last successful login stated as Greenwich Mean Time (users only).</td>
</tr>
<tr>
<td>As Long</td>
<td>DbConnectTime</td>
<td>Local (server) time of database connection. Read-only. Cannot be set by EsbSetUser.</td>
</tr>
<tr>
<td>As Long</td>
<td>LoginId</td>
<td>User login identification tag (users only).</td>
</tr>
<tr>
<td>As Integer</td>
<td>Login</td>
<td>Flag to indicate whether logged in (users only).</td>
</tr>
<tr>
<td>As Integer</td>
<td>Type</td>
<td>The type of the structure (user or group). Values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ESB_TYPE_USER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ESB_TYPE_GROUP</td>
</tr>
<tr>
<td>As Integer</td>
<td>Access</td>
<td>User or group assigned default access privileges. This field can take any combination of the following bit values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ESB_ACCESS_SUPER /*Administrator, all bits set */</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ESB_PRIV_APPCREATE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ESB_PRIV_USERCREATE</td>
</tr>
<tr>
<td>As Integer</td>
<td>MaxAccess</td>
<td>User's maximum access privileges (users only). This combines individual access, and access levels conferred by group membership.</td>
</tr>
<tr>
<td>As Integer</td>
<td>Expiration</td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td>As Integer</td>
<td>FailCount</td>
<td>Count of the failed login attempts since the last successful login (users only).</td>
</tr>
<tr>
<td>As String</td>
<td>Name</td>
<td>User or group name (ESB_USERNAMELEN).</td>
</tr>
<tr>
<td>As String</td>
<td>AppName</td>
<td>Name of the currently connected application (if applicable) (ESB_APPNAMELEN).</td>
</tr>
<tr>
<td>As String</td>
<td>DbName</td>
<td>Name of the currently connected database (if applicable) (ESB_DBNAMELEN).</td>
</tr>
<tr>
<td>As String</td>
<td>Description</td>
<td>User/group description (ESB_DESCLEN).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For Future Use. Not settable by user.</td>
</tr>
<tr>
<td>As String</td>
<td>EMailID</td>
<td>User/group email address (ESB_DESCLEN).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For Future Use. Not settable by user.</td>
</tr>
<tr>
<td>As Boolean</td>
<td>LockedOut</td>
<td>Flag that user is locked out.</td>
</tr>
<tr>
<td>As Boolean</td>
<td>PwdChgNow</td>
<td>Flag that user must change password.</td>
</tr>
</tbody>
</table>

**ESB_USERINFOEX_T**

This structure stores information about users or groups.
Type ESB_USERINFOEX_T

LastLogin As Long
DbConnectTime As Long
LoginId As Long
Login As Integer
Type As Integer
Access As Integer
MaxAccess As Integer
Expiration As Integer
FailCount As Integer
Name As String * ESB_USERNAMELEN
AppName As String * ESB_APPNAMELEN
DbName As String * ESB_DBNAMELEN
Password As String * ESB_PASSWORDLEN ' Authentication Password
Description As String * ESB_DESCLEN
EMailID As String * ESB_DESCLEN
LockedOut As Boolean
PwdChgNow As Boolean
protocol As String * ESB_PROTOCOLNAMELEN ' External Authentication
Protocol connparam As String * ESB_CONNPARAMLEN ' External Authentication
Connection

End Type

Some of the fields are specific to users and cannot be used for groups. The Access, Expiration, and PwdChgNow fields are the only fields in this structure that can be modified using the API.

The fields are:

<table>
<thead>
<tr>
<th>VB Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>As Long</td>
<td>LastLogin</td>
<td>Date of the user's last successful login stated as Greenwich Mean Time (users only).</td>
</tr>
<tr>
<td>As Long</td>
<td>DbConnectTime</td>
<td>Local (server) time of database connection. Read-only. Cannot be set by EsbSetUser.</td>
</tr>
<tr>
<td>As Long</td>
<td>LoginId</td>
<td>User login identification tag (users only).</td>
</tr>
<tr>
<td>As Integer</td>
<td>Login</td>
<td>Flag to indicate whether logged in (users only).</td>
</tr>
<tr>
<td>As Integer</td>
<td>Type</td>
<td>The type of the structure (user or group). Values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESB_TYPE_USER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESB_TYPE_GROUP</td>
</tr>
<tr>
<td>As Integer</td>
<td>Access</td>
<td>User or group assigned default access privileges. This field can take any combination of the following bit values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESB_ACCESS_SUPER /*Administrator, all bits set */</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESB_PRIV_APPCREATE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESB_PRIV_USERCREATE</td>
</tr>
<tr>
<td>As Integer</td>
<td>MaxAccess</td>
<td>User's maximum access privileges (users only). This combines individual access, and access levels conferred by group membership.</td>
</tr>
<tr>
<td>As Integer</td>
<td>Expiration</td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td>VB Data Type</td>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>As Integer</td>
<td>FailCount</td>
<td>Count of the failed login attempts since the last successful login (users only).</td>
</tr>
<tr>
<td>As String * ESB_USERNAMELEN</td>
<td>Name</td>
<td>User or group name (ESB_USERNAMELEN).</td>
</tr>
<tr>
<td>As String * ESB_APPNAMELEN</td>
<td>AppName</td>
<td>Name of the currently connected application (if applicable) (ESB_APPNAMELEN).</td>
</tr>
<tr>
<td>As String * ESB_DBNAMELEN</td>
<td>DbName</td>
<td>Name of the currently connected database (if applicable) (ESB_DBNAMELEN).</td>
</tr>
<tr>
<td>As String * ESB_PASSWORDLEN</td>
<td>Password</td>
<td>Password of externally authenticated user. This is used only when setting an externally authenticated user to the Essbase authenticated mechanisms. This password is ignored in other situations, including retrieving information from the server on the externally authenticated user.</td>
</tr>
<tr>
<td>As String</td>
<td>Description</td>
<td>User/group description (ESB_DESCLEN). For Future Use. Not settable by user.</td>
</tr>
<tr>
<td>As String</td>
<td>EMailID</td>
<td>User/group email address (ESB_DESCLEN). For Future Use. Not settable by user.</td>
</tr>
<tr>
<td>As Integer</td>
<td>LockedOut</td>
<td>Flag that user is locked out.</td>
</tr>
<tr>
<td>As Integer</td>
<td>PwdChgNow</td>
<td>Flag that user must change password.</td>
</tr>
<tr>
<td>As String * ESB_PROTOCOLNAMELEN</td>
<td>protocol</td>
<td>The External authentication protocol.</td>
</tr>
<tr>
<td>As String * ESB_CONNPARAMLEN</td>
<td>connparam</td>
<td>The External authentication connection.</td>
</tr>
</tbody>
</table>

**ESB_VARIABLE_T**

*ESB_VARIABLE_T* is the primary substitution variable datatype. It identifies the substitution variable’s value and name, as well as the Esbbase database, application, and server where the variable is defined.

The server name is optional, but recommended. If not included, the current server is the default. The AppName is optional. The DbName is optional, but if it exists, then the AppName member is required. The VarName is required. The VarValue is required.

Type ESB_VARIABLE_T

```vbnet
    Server As String * ESB_SVRNAMELEN
    AppName As String * ESB_APPNAMELEN
    DbName As String * ESB_DBNAMELEN
    VarName As String * ESB_MBRNAMELEN
    VarValue As String * ESB_VARVALUELEN

End Type
```

<table>
<thead>
<tr>
<th>VB Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESB_SVRNAME_T</td>
<td>Server</td>
<td>Name of server where variable is defined (optional)</td>
</tr>
<tr>
<td>VB Data Type</td>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>----------</td>
<td>-------------------------------------------------------------------</td>
</tr>
<tr>
<td>ESB_APPNAME_T</td>
<td>AppName</td>
<td>Name of application to restrict variable to</td>
</tr>
<tr>
<td>ESB_DBNAME_T</td>
<td>DbName</td>
<td>Name of database to restrict variable to. If used, it requires that application be set.</td>
</tr>
<tr>
<td>ESB_MBRNAME_T</td>
<td>VarName</td>
<td>Name of substitution variable.</td>
</tr>
<tr>
<td>ESB_CHAR_T</td>
<td>VarValue[256]</td>
<td>Value of substitution variable.</td>
</tr>
</tbody>
</table>
In This Chapter

Visual Basic Main API Function Categories .......................................................... 1199
Visual Basic Main API Function Reference .......................................................... 1209

Visual Basic Main API Function Categories

- “VB Main API Alias Table Functions” on page 1199
- “VB Main API Application Functions” on page 1200
- “VB Main API Attributes Functions” on page 1201
- “VB Main API Database Functions” on page 1201
- “VB Main API Database Member Functions” on page 1202
- “VB Main API Drill-through Functions” on page 1202
- “VB Main API File Functions” on page 1203
- “VB Main API Group Administration Functions” on page 1203
- “VB Main API Initialization and Login Functions” on page 1204
- “VB Main API LRO Functions” on page 1204
- “VB Main API Location Aliases Functions” on page 1205
- “VB Main API Miscellaneous Functions” on page 1205
- “VB Main API Object Functions” on page 1206
- “VB Main API Reporting, Updating, and Calculation Functions” on page 1206
- “VB Main API Security Filter Functions” on page 1207
- “VB Main API Substitution Variables Functions” on page 1208
- “VB Main API User Administration Functions” on page 1208

VB Main API Alias Table Functions

Alias table functions manage database alias tables.
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EsbListAliases</td>
<td>Lists all the alias tables in the active database.</td>
</tr>
<tr>
<td>EsbLoadAlias</td>
<td>Loads an alias table for the active database from a structured text file.</td>
</tr>
<tr>
<td>EsbGetAlias</td>
<td>Gets the active alias table name from the active database for a user.</td>
</tr>
<tr>
<td>EsbSetAlias</td>
<td>Sets the active alias table in the active database for a user.</td>
</tr>
<tr>
<td>EsbDisplayAlias</td>
<td>Dumps the contents of an alias table in the active database.</td>
</tr>
<tr>
<td>EsbRemoveAlias</td>
<td>Removes an alias table from the active database.</td>
</tr>
<tr>
<td>EsbClearAliases</td>
<td>Clears all alias tables for the active database.</td>
</tr>
</tbody>
</table>

### VB Main API Application Functions

The application functions create new applications and modify, copy, get information about and otherwise manage existing applications.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EsbGetActive</td>
<td>Gets the names of the caller's current active application and database.</td>
</tr>
<tr>
<td>EsbSetActive</td>
<td>Sets the caller's active application and database.</td>
</tr>
<tr>
<td>EsbClearActive</td>
<td>Clears the user's current active application and database.</td>
</tr>
<tr>
<td>EsbListApplications</td>
<td>Lists all applications which are accessible to the caller.</td>
</tr>
<tr>
<td>EsbCreateApplication</td>
<td>Creates a new application, either on the client or the server.</td>
</tr>
<tr>
<td>EsbCreateStorageTypedApplication</td>
<td>Creates a new application with the option of Multi-Dimensional or Aggregate Storage.</td>
</tr>
<tr>
<td>EsbDeleteApplication</td>
<td>Deletes an existing application, either on the client or the server.</td>
</tr>
<tr>
<td>EsbRenameApplication</td>
<td>Renames an existing application, either on the client or the server.</td>
</tr>
<tr>
<td>EsbGetApplicationInfoEx</td>
<td>Gets information from one or more applications.</td>
</tr>
<tr>
<td>EsbCopyApplication</td>
<td>Copies an existing application, either on the client or the server, to a new application, including all associated databases and objects.</td>
</tr>
<tr>
<td>EsbGetApplicationState</td>
<td>Gets an application state structure, which contains user-configurable parameters for the application.</td>
</tr>
<tr>
<td>EsbSetApplicationState</td>
<td>Sets user-configurable parameters for the application using the application's state structure.</td>
</tr>
<tr>
<td>EsbGetApplicationInfo</td>
<td>Gets an application's information structure, which contains non-user-configurable parameters for the application.</td>
</tr>
<tr>
<td>EsbLoadApplication</td>
<td>Starts an application on the server.</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>EsbUnloadApplication</td>
<td>Stops an application on the server.</td>
</tr>
</tbody>
</table>

**VB Main API Attributes Functions**

These Visual Basic Main functions are for attributes.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EsbCheckAttributes</td>
<td>Returns the attribute type for given attribute dimensions, base dimensions, attribute members, and base members</td>
</tr>
<tr>
<td>EsbGetAssociatedAttributesInfo</td>
<td>Returns the attribute members associated with a given base member</td>
</tr>
<tr>
<td>EsbGetAttributeInfo</td>
<td>Returns attribute information for a given attribute member or dimension</td>
</tr>
<tr>
<td>EsbGetAttributeSpecifications</td>
<td>Retrieves attribute specifications for the outline</td>
</tr>
</tbody>
</table>

Click here to see the VB Outline API “VB Outline API Attributes Functions” on page 1466.

**VB Main API Database Functions**

Database functions carry out database management tasks, and retrieve and modify database information structures.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EsbClearDatabase</td>
<td>Clears all loaded data in the active database.</td>
</tr>
<tr>
<td>EsbCopyDatabase</td>
<td>Copies an existing database, either on the client or the server, to a new database, including all associated databases and objects.</td>
</tr>
<tr>
<td>EsbCreateDatabase</td>
<td>Creates a new database within an application, either on the client or the server.</td>
</tr>
<tr>
<td>EsbDeleteDatabase</td>
<td>Deletes an existing database from an application, either on the client or the server.</td>
</tr>
<tr>
<td>EsbGetCurrencyRateInfo</td>
<td>Gets a list of structures containing rate information for all members of the tagged currency partition dimension in the active database outline.</td>
</tr>
<tr>
<td>EsbGetDatabaseInfo</td>
<td>Gets a database's information structure, which contains non user-configurable parameters for the database.</td>
</tr>
<tr>
<td>EsbGetDatabaseNote</td>
<td>Gets a database's note-of-the-day message.</td>
</tr>
<tr>
<td>EsbGetDatabaseState</td>
<td>Gets a database's state structure, which contains user-configurable parameters for the database.</td>
</tr>
<tr>
<td>EsbGetDatabaseStats</td>
<td>Gets the active database's stats structure, which contains statistical information about the database.</td>
</tr>
<tr>
<td>EsbListCurrencyDatabases</td>
<td>Lists all currency databases within a specific application which are accessible to the caller.</td>
</tr>
</tbody>
</table>
### VB Main API Database Member Functions

These functions obtain information about database members and build database dimensions.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EsbQueryDatabaseMembers</td>
<td>Performs a report-style query to list a selection of database member information.</td>
</tr>
<tr>
<td>EsbCheckMemberName</td>
<td>Checks if a string is a valid member name within the active database outline.</td>
</tr>
<tr>
<td>EsbGetMemberInfo</td>
<td>Gets a structure containing information about a specific member in the active database outline.</td>
</tr>
<tr>
<td>EsbGetMemberCalc</td>
<td>Gets the calc equation for a specific member in the active database outline.</td>
</tr>
<tr>
<td>EsbGetDimensionInfo</td>
<td>Gets dimension information.</td>
</tr>
<tr>
<td>EsbBuildDimension</td>
<td>Allows the creation of a dimension in the active database from a data file and rules file.</td>
</tr>
<tr>
<td>EsbBuildDimFile</td>
<td>This function builds a data file to be used in the addition or removal of members from the outline in the active database.</td>
</tr>
<tr>
<td>EsbBuildDimStart</td>
<td>This function starts the process of the adding or removing members from the outline in the active database.</td>
</tr>
</tbody>
</table>

### VB Main API Drill-through Functions

The following Drill-through functions manage drill-through URLs for drilling through to information hosted on Oracle ERP and EPM applications.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EsbCreateDrillThruURL</td>
<td>Creates a drill-through URL, with the given link and the name, within the active database outline.</td>
</tr>
<tr>
<td>EsbDeleteDrillThruURL</td>
<td>Deletes a drill-through URL, with the given URL name, within the active database outline.</td>
</tr>
</tbody>
</table>
### VB Main API File Functions

File functions enable an application to use predefined report scripts, data files and calculation scripts against the active database. There are also functions for importing and exporting data to and from both text and binary files.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EsbArchiveBegin</td>
<td>Prepares database for archive by setting READ-ONLY status.</td>
</tr>
<tr>
<td>EsbArchiveEnd</td>
<td>After archive, returns database status to READ-WRITE.</td>
</tr>
<tr>
<td>EsbCalcFile</td>
<td>Executes a calc script against the active database from a file.</td>
</tr>
<tr>
<td>EsbExport</td>
<td>Allows the exporting of data from the current database to a text file.</td>
</tr>
<tr>
<td>EsbImport</td>
<td>Allows the importing of data from text files and other sources to the current database.</td>
</tr>
<tr>
<td>EsbListDbFiles</td>
<td>Retrieves information on specified index and data files</td>
</tr>
<tr>
<td>EsbReportFile</td>
<td>Sends a report specification to the active database from a file.</td>
</tr>
<tr>
<td>EsbSetDefaultCalcFile</td>
<td>Sets the default calc script for the active database from a calc script file.</td>
</tr>
<tr>
<td>EsbUpdateFile</td>
<td>Sends an update specification to the active database from a file.</td>
</tr>
</tbody>
</table>

### VB Main API Group Administration Functions

These functions create groups, set and modify group attributes, and obtain information about existing groups.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EsbListGroups</td>
<td>Lists all groups who have access to a particular Essbase Server.</td>
</tr>
<tr>
<td>EsbCreateGroup</td>
<td>Creates a new group.</td>
</tr>
<tr>
<td>EsbDeleteGroup</td>
<td>Deletes an existing group.</td>
</tr>
<tr>
<td>EsbRenameGroup</td>
<td>Renames an existing group.</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>EsbGetGroup</td>
<td>Gets a group information structure, which contains security information for the group.</td>
</tr>
<tr>
<td>EsbSetGroup</td>
<td>Sets a group information structure.</td>
</tr>
<tr>
<td>EsbGetGroupList</td>
<td>Gets the list of users who are members of a group (or the list of groups to which a user belongs).</td>
</tr>
<tr>
<td>EsbSetGroupList</td>
<td>Sets the list of users who are members of a group.</td>
</tr>
<tr>
<td>EsbAddToGroup</td>
<td>Adds a user to a list of group members.</td>
</tr>
<tr>
<td>EsbDeleteFromGroup</td>
<td>Removes a user from a list of group members</td>
</tr>
</tbody>
</table>

**VB Main API Initialization and Login Functions**

These functions initialize the API, and log in and out of the Essbase Server. They also obtain version information, and enable an application to create and delete local contexts.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EsbAutoLogin</td>
<td>Displays a dialog box which allows the user to log in to an Essbase Server, and optionally selects an active application and database.</td>
</tr>
<tr>
<td>EsbCreateLocalContext</td>
<td>Creates a local API context for use in local API operations</td>
</tr>
<tr>
<td>EsbDeleteLocalContext</td>
<td>Releases a local context previously created by EsbCreateLocalContext()</td>
</tr>
<tr>
<td>EsbGetAPIVersion</td>
<td>Gets version number of the API DLL in use</td>
</tr>
<tr>
<td>EsbGetVersion</td>
<td>Gets the full version number of the connected Essbase Server.</td>
</tr>
<tr>
<td>EsbInit</td>
<td>Initializes the API and message database.</td>
</tr>
<tr>
<td>EsbLogin</td>
<td>Logs a user in to the Essbase Server.</td>
</tr>
<tr>
<td>EsbLoginSetPassword</td>
<td>Logs in a user, and changes the password.</td>
</tr>
<tr>
<td>EsbLogout</td>
<td>Logs a user out from an Essbase Server.</td>
</tr>
<tr>
<td>EsbLogoutUser</td>
<td>Allows an Administrator or Application Manager to disconnect another user from an Essbase Server.</td>
</tr>
<tr>
<td>EsbShutdownServer</td>
<td>Allows an Administrator to remotely stop the Agent.</td>
</tr>
<tr>
<td>EsbTerm</td>
<td>Terminates the API and releases all system resources used by the API.</td>
</tr>
<tr>
<td>EsbValidateHCtx</td>
<td>Validates a specific API context handle (hCtx).</td>
</tr>
</tbody>
</table>

**VB Main API LRO Functions**

These functions create, retrieve and delete LROs and return information about them.
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EsbLROAddObject</td>
<td>Links a reporting object to a data cell in an Essbase database.</td>
</tr>
<tr>
<td>EsbLRDeleteCellObjects</td>
<td>Deletes all objects linked to a given data cell in an Essbase database.</td>
</tr>
<tr>
<td>EsbLRDeleteObject</td>
<td>Deletes a specific object linked to a data cell in an Essbase database.</td>
</tr>
<tr>
<td>EsbLRGetCatalog</td>
<td>Retrieves a list of LRO catalog entries for a given data cell in an Essbase database.</td>
</tr>
<tr>
<td>EsbLRGetMemberCombo</td>
<td>Retrieves the n'th member from the member combination list of the current LRO.</td>
</tr>
<tr>
<td>EsbLRGetObject</td>
<td>Retrieves an object linked to a data cell in an Essbase database.</td>
</tr>
<tr>
<td>EsbLRListObjects</td>
<td>Retrieves a list of all objects linked to cells in the active database for a given user name and/or modification date.</td>
</tr>
<tr>
<td>EsbLRPurgeObjects</td>
<td>Deletes all objects linked to cells in the active database for a given user name and/or modification date.</td>
</tr>
<tr>
<td>EsbLRUpdateObject</td>
<td>Stores an updated version of an LRO on the server.</td>
</tr>
</tbody>
</table>

### VB Main API Location Aliases Functions

These functions create, delete and list location aliases.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EsbCreateLocationAlias</td>
<td>Maps an alias name to the host name, application name, database name, user login name, and user password</td>
</tr>
<tr>
<td>EsbDeleteLocationAlias</td>
<td>Deletes an existing location alias</td>
</tr>
<tr>
<td>EsbGetLocationAliasList</td>
<td>Returns all location aliases and the names to which the location aliases are mapped</td>
</tr>
</tbody>
</table>

### VB Main API Miscellaneous Functions

These functions manage asynchronous processes, obtain state information, handle log files, and retrieve messages.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EsbGetProcessState</td>
<td>Gets the current state of an asynchronous process, such as a calculate or a data import.</td>
</tr>
<tr>
<td>EsbCancelProcess</td>
<td>Cancels an asynchronous process which has not yet completed.</td>
</tr>
<tr>
<td>EsbGetLogFile</td>
<td>Copies all or part of an application log file from the server to the client.</td>
</tr>
<tr>
<td>EsbDeleteLogFile</td>
<td>Deletes an application log file on the server.</td>
</tr>
<tr>
<td>EsbGetGlobalState</td>
<td>Gets the server global state structure which contains parameters for system administration.</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>EsbSetGlobalState</td>
<td>Sets the server global state structure which contains parameters for system administration.</td>
</tr>
<tr>
<td>EsbGetNextItem</td>
<td>Gets the server global state structure which contains system administration parameters.</td>
</tr>
<tr>
<td>EsbGetMessage</td>
<td>Retrieves the top message from the message stack filled during VB API function execution if ClientError in ESB_INIT_T structure has been set to ESB_TRUE during initialization.</td>
</tr>
<tr>
<td>EsbSetPath</td>
<td>Sets the ARBORPATH environment variable for the current process.</td>
</tr>
</tbody>
</table>

**VB Main API Object Functions**

These functions create, delete, move and copy objects. They also retrieve and display object information and control access to objects.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EsbGetLocalPath</td>
<td>Gets the full local file for an object file on the client.</td>
</tr>
<tr>
<td>EsbListObjects</td>
<td>Lists all objects of types specified.</td>
</tr>
<tr>
<td>EsbGetObjectInfo</td>
<td>Gets information about a specified object.</td>
</tr>
<tr>
<td>EsbGetObject</td>
<td>Copies an object from the server to a local file, and optionally locks it.</td>
</tr>
<tr>
<td>EsbPutObject</td>
<td>Copies an object from a local file to the server, and optionally unlocks it.</td>
</tr>
<tr>
<td>EsbLockObject</td>
<td>Locks an object on the server to prevent other users from updating it.</td>
</tr>
<tr>
<td>EsbUnlockObject</td>
<td>Unlocks a locked object on the server.</td>
</tr>
<tr>
<td>EsbCreateObject</td>
<td>Creates a new object.</td>
</tr>
<tr>
<td>EsbDeleteObject</td>
<td>Deletes an existing object.</td>
</tr>
<tr>
<td>EsbRenameObject</td>
<td>Renames an existing object.</td>
</tr>
<tr>
<td>EsbCopyObject</td>
<td>Copies an object.</td>
</tr>
</tbody>
</table>

**VB Main API Reporting, Updating, and Calculation Functions**

These functions carry out reporting (retrieving data), updating (loading data) and calculation (aggregating data) tasks against the active database.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EsbBeginCalc</td>
<td>Starts sending a calc script and optionally executes it against the active database.</td>
</tr>
<tr>
<td>EsbBeginReport</td>
<td>Starts sending a report specification to the active database.</td>
</tr>
<tr>
<td>EsbBeginUpdate</td>
<td>Starts sending an update specification to the active database.</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>EsbCalc</td>
<td>Sends and optionally executes a calc script against the active database as a single string.</td>
</tr>
<tr>
<td>EsbDefaultCalc</td>
<td>Executes the default calculation for the active database.</td>
</tr>
<tr>
<td>EsbEndCalc</td>
<td>Marks the end of a calc script being sent to the active database.</td>
</tr>
<tr>
<td>EsbEndReport</td>
<td>Marks the end of a report specification being sent to the active database.</td>
</tr>
<tr>
<td>EsbEndUpdate</td>
<td>Marks the end of an update specification being sent to the active database.</td>
</tr>
<tr>
<td>EsbGetDefaultCalc</td>
<td>Gets the default calc script for the active database.</td>
</tr>
<tr>
<td>EsbGetString</td>
<td>Gets a string of data from the active database.</td>
</tr>
<tr>
<td>EsbGetStringBuf</td>
<td>Gets data from the active database until it returns all available data or until the caller's buffer is full.</td>
</tr>
<tr>
<td>EsbReport</td>
<td>Sends a report specification to the active database as a single string.</td>
</tr>
<tr>
<td>EsbSendString</td>
<td>Sends a string of data to the active database.</td>
</tr>
<tr>
<td>EsbSetDefaultCalc</td>
<td>Sets the default calc script for a database.</td>
</tr>
<tr>
<td>EsbUpdate</td>
<td>Sends an update to the active database as a single string.</td>
</tr>
</tbody>
</table>

**VB Main API Security Filter Functions**

Security filter functions set filter contents, assign filters to user groups, display filter lists for data bases, and obtain other data about security filters.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EsbListFilters</td>
<td>Lists all filters for a database.</td>
</tr>
<tr>
<td>EsbGetFilter</td>
<td>Starts getting the contents of a filter.</td>
</tr>
<tr>
<td>EsbGetFilterRow</td>
<td>Gets the next row of a filter.</td>
</tr>
<tr>
<td>EsbSetFilter</td>
<td>Starts setting the contents of a filter.</td>
</tr>
<tr>
<td>EsbSetFilterRow</td>
<td>Gets the next row of a filter.</td>
</tr>
<tr>
<td>EsbGetFilterList</td>
<td>Gets the list of users who are assigned a filter.</td>
</tr>
<tr>
<td>EsbSetFilterList</td>
<td>Sets the list of users who are assigned a filter.</td>
</tr>
<tr>
<td>EsbDeleteFilter</td>
<td>Deletes an existing filter.</td>
</tr>
<tr>
<td>EsbRenameFilter</td>
<td>Renames an existing filter.</td>
</tr>
<tr>
<td>EsbCopyFilter</td>
<td>Copies an existing filter.</td>
</tr>
<tr>
<td>EsbVerifyFilter</td>
<td>Verifies the syntax of a series of filter row strings against a specified database.</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>EsbVerifyFilterRow</td>
<td>Verifies the syntax of a single filter row string against a specified database.</td>
</tr>
</tbody>
</table>

**VB Main API Substitution Variables Functions**

These functions create, retrieve and delete substitution variables and return information about them.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EsbCreateVariable</td>
<td>This function creates a new substitution variable or modifies an existing substitution variable if the variable name already exists with the identical server, application, and database values.</td>
</tr>
<tr>
<td>EsbDeleteVariable</td>
<td>This function deletes a substitution variable.</td>
</tr>
<tr>
<td>EsbGetVariable</td>
<td>This function retrieves the value of a substitution variable.</td>
</tr>
<tr>
<td>EsbGetVariable</td>
<td>This function lists all substitution variables that conform to the input criteria.</td>
</tr>
</tbody>
</table>

**VB Main API User Administration Functions**

User administration functions create users, assign their passwords, and their access to databases, applications, and calc scripts. Functions are also provided to retrieve information about user capabilities.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EsbListUsers</td>
<td>Lists all users who have access to a particular Essbase Server.</td>
</tr>
<tr>
<td>EsbCreateUser</td>
<td>Creates a new user.</td>
</tr>
<tr>
<td>EsbDeleteUser</td>
<td>Deletes an existing user.</td>
</tr>
<tr>
<td>EsbRenameUser</td>
<td>Renames an existing user.</td>
</tr>
<tr>
<td>EsbGetUser</td>
<td>Gets a user information structure, which contains security information for the user.</td>
</tr>
<tr>
<td>EsbSetUser</td>
<td>Sets a user information structure, which contains security information for the user.</td>
</tr>
<tr>
<td>EsbResetUser</td>
<td>Resets the user's security structure to its initial state.</td>
</tr>
<tr>
<td>EsbSetPassword</td>
<td>Sets a user's password, erasing the existing password.</td>
</tr>
<tr>
<td>EsbGetApplicationAccess</td>
<td>Gets a list of user application access structures, which contain information about user access to applications.</td>
</tr>
<tr>
<td>EsbSetApplicationAccess</td>
<td>Sets a list of user application access structures.</td>
</tr>
<tr>
<td>EsbGetDatabaseAccess</td>
<td>Gets a list of user database access structures.</td>
</tr>
<tr>
<td>EsbSetDatabaseAccess</td>
<td>Sets a list of user database access structures.</td>
</tr>
</tbody>
</table>
### Function Description

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EsbGetCalcList</td>
<td>Gets the list of calc scripts objects accessible to the user.</td>
</tr>
<tr>
<td>EsbSetCalcList</td>
<td>Sets the list of calc script objects which are available to a user.</td>
</tr>
<tr>
<td>EsbListConnections</td>
<td>Lists all users who are connected to the current application and database.</td>
</tr>
<tr>
<td>EsbListLocks</td>
<td>Lists all users who are connected to a specific application and database.</td>
</tr>
<tr>
<td>EsbRemoveLocks</td>
<td>Removes all data block locks on a database which are currently held by a user.</td>
</tr>
</tbody>
</table>

### Visual Basic Main API Function Reference

Consult the Contents pane for an alphabetical list of Visual Basic Main API functions, which are prefaced with Esb.

#### EsbAddToGroup

Adds a user to the list of group members.

**Syntax**

```vba
EsbAddToGroup Lib "ESBAPIN" (hCtx, GroupName, User)
ByVal hCtx As Long
ByVal GrpName As String
ByVal User As String
```

**Parameter Description**

- **hCtx**: VB API context handle.
- **GroupName**: Group name.
- **User**: Name of user to add to the group list.

**Notes**

In addition to adding the specified user to the list of members for the specified group, this function adds the group to the user’s own list of associated groups.

**Return Value**

None.

**Access**

This function requires the caller to have Create/Delete User privilege (ESB_PRIV_USERCREATE) for the logged in server.

**Example**

```vba
Declare Function EsbAddToGroup Lib "ESBAPIN" (ByVal hCtx As Long, ByVal GroupName As String, ByVal User As String) As Long
```
Sub ESB_AddToGroup ()
    Dim sts As Long
    Dim GroupName As String
    Dim User As String
    GroupName = "PowerUsers"
    User = "Jim Smith"
    '******************
    ' Add user to group
    '******************
    sts = EsbAddToGroup (hCtx, GroupName, User)
End Sub

See Also
- EsbDeleteFromGroup
- EsbGetGroupList
- EsbListGroups
- EsbSetGroupList

**EsbArchive**

No longer in use.

This function is retained for compatibility with earlier releases of Essbase. For current Essbase
archiving, see EsbArchiveBegin and EsbArchiveEnd. This function now returns the error
message ESB_STS_OBSOLETE.

See Also
- EsbRestore
- EsbGetProcessState
- EsbArchiveBegin
- EsbArchiveEnd

**EsbArchiveBegin**

Prepares the server for archiving by changing server mode to "read-only."

**Syntax**

```
EsbArchiveBegin (hCtx, AppName, DbName, FileName)
```

**Parameter**  **Description**

hCtx  API context handle.
AppName  Name of application to archive.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DbName</td>
<td>Name of database to archive.</td>
</tr>
<tr>
<td>FileName</td>
<td>Name of file to contain archive information.</td>
</tr>
</tbody>
</table>

**Notes**

- This function changes server mode to Read-Only. This allows the database administrator to back up all the files on the server and prevents writing to the files during the backup. The database files to back up are listed in the `app\db` directory specified by the `FileName` parameter.
- Any existing information in the specified file is overwritten by the archived data.

**Return Value**

None.

**Access**

The caller must have at least read access (ESB_PRIV_READ) to the database, and must select it as the active database using `EsbSetActive()`.

**Example**

```vbnet
Declare Function EsbArchiveBegin Lib "ESBAPIN" (ByVal hCtx As Long, ByVal AppName As String, ByVal DbName As String, ByVal FileName As String) As Long

Sub ESB_ArchiveBegin ()
    Dim sts As Long
    Dim AppName As String
    Dim DbName As String
    Dim FileName As String
    AppName = "Sample"
    DbName = "Basic"
    FileName = "Test.arc"
    sts = EsbArchiveBegin (hCtx, AppName, DbName, FileName)
    ****
    At this point, you can back up the server safely.
End Sub
```

**See Also**

- `EsbArchiveEnd`
- `EsbRestore`
- `EsbGetProcessState`

---

**EsbArchiveEnd**

Restores the server to "read-write" mode after archiving is complete.

**Syntax**

```
EsbArchiveEnd (hCtx, AppName, DbName)
ByVal hCtx As Long
```
ByValAppName As String
ByValDbName As String

Parameter Description

hCtx API context handle.

AppName Name of archived application.

DbName Name of archived database.

Notes

After calling EsbArchiveBegin(), a call to EsbArchiveEnd() is required to restore read-write mode.

Return Value

None.

Access

The caller must have at least read access (ESB_PRIV_READ) to the database, and must select it as the active database using EsbSetActive().

Example

Declare Function EsbArchiveEnd Lib "ESBAPIN" (ByVal hCtx As Long, ByValAppName As String, ByValDbName As String) As Long

Sub ESB_ArchiveEnd()
    Dim sts As Long
    DimAppName As String
    Dim DbName As String
    AppName = "Sample"
    DbName = "Basic"
    "**** Archive End ***
    sts = EsbArchiveEnd (hCtx,AppName,DbName)
End Sub

See Also

- EsbArchiveBegin
- EsbRestore

**EsbAutoLogin**

Displays a dialog box that allows the user to log in to an Essbase Server, and optionally select an active application and database.

Syntax

EsbAutoLogin (hInst, Server, User, Password,AppName, DbName, opt, pAccess, phCtx)
ByVal hInst As Long
ByVal Server As String
ByVal User As String
ByVal Password As String
ByVal **AppName** As String
ByVal **DbName** As String
ByVal **opt** As Integer
  pAccess As Integer
  phCtx As Long

**Parameter** | **Description**
--- | ---
hInst | VB API instance handle.
Server | Network server name string.
  The server name can be expressed as hostname, hostname:port, or as a URL representing the APS servlet endpoint with the Essbase failover cluster name; for example:
  http://myhost:13080/aps/Essbase?clusternam...E...Essbase=Cluster1
  For secure mode (SSL), the URL syntax is
  http[s]://host:port/aps/Essbase?
  ClusterName=logicalName&SecureMODE=yesORno
  For example,
  https://myhost:13080/aps/Essbase?clusternam...E...Essbase=Cluster1&SecureMODE=Yes
User | User name string.
Password | Password string.
AppName | Application name.
DbName | Database name.
Options | Options flag. Values:
  - ESB_AUTO_NODIALOG—Attempts to log the user in without displaying the dialog, using the default settings (from the above arguments).
  - ESB_AUTO_NOSELECT—Allows the user to log in without selecting an application and database (lower part of the dialog is not displayed).
  You can add ESB_AUTO_NODIALOG and ESB_AUTO_NOSELECT together to log in a user without a dialog box and not select an application and database:
  ESB_AUTO_NODIALOG + ESB_AUTO_NOSELECT
  - ESB_AUTO_DEFAULT—Allows the user to log in and select an application and database interactively in the dialog box.
pAccess | Address of variable to receive database access level.
phCtx | Address of variable to receive Essbase context handle.

**Notes**
- The dialog box returned by this function is automatically managed by the function, and provides features in the login dialog to change the user password, display the database note message, etc., and so provides a standardized and powerful login screen for all applications using the VB API.
Use this function instead of the EsbLogin function if you are programming in the Windows environments.

The function should be called after executing a successful call to EsbInit, and prior to making any other VB API calls which require a context handle argument.

This function is supported only in the Windows environments. It is not supported in UNIX environments.

The string arguments Server, User, Password,AppName or DbName must be included. They may optionally be an empty string. If any are not an empty string, the buffers they point to are updated on returning from the function with the actual values selected by the user from the dialog box. If any of the passed-in arguments point to valid strings, they are used as the default displayed values in the dialog. The buffers for these arguments must be large enough to contain any possible return value, not just the values passed in.

If the login is successful, the server and user names are automatically stored (in the file ESSBASE.INI) and are used as the defaults the next time this function is called (unless those arguments are specified in subsequent calls). The names of all servers which have been successfully connected to are also stored and displayed.

The auto login dialog box is a child window of the current active window (the window that has the focus). Therefore avoid destroying the active window or changing focus while the auto login dialog is displayed.

This function returns a value of ESB_STS_CANCEL if the user presses the Cancel button or the Esc key in the dialog box.

In Windows environments, if the end user clicks the Help button, the Essbase System Login help topic shipped with the Oracle Essbase Spreadsheet Add-in User's Guide online help is opened. You can redirect the Help button to point to a different help file by specifying a different help file name in the ESB_INIT_T structure.

Return Value

If successful, returns an Essbase context handle in phCtx, which can be passed as an argument in subsequent calls to other VB API functions. Also returns the user's access level to the selected application and database (if selected) in pAccess.

Access

Before calling this function, you must first initialize the VB API and obtain a valid instance handle by calling the EsbInit function.

Example

Declare Function EsbAutoLogin Lib "ESBAPIN" (ByVal hInst As Long, ByVal Server As String, ByVal User As String, ByVal Password As String, ByVal AppName As String, ByVal DbName As String, ByVal Opt As Integer, pAccess As Integer, phCtx As Long) As Long

Sub ESB_AutoLogin ()
Dim sts As Long
Dim Server As String * ESB_SVRNAMELEN
Dim User As String * ESB_USERNAMELEN
Dim Password As String * ESB_PASSWORDLEN
Dim AppName As String * ESB_APPNAMELEN
Dim DbName As String * ESB_DBNAMELEN
Dim pOption As Integer
Dim pAccess As Integer
Dim hCtx As Long

'***********************
' Initialize parameters
'***********************
Server = "Server"
User = "User"
Password = "Password"
AppName = ""
DbName = ""
pOption = ESB_AUTO_DEFAULT

' Login to Essbase Server
'****************************
sts = EsbAutoLogin (hInst, Server, User, Password, AppName, DbName, pOption, pAccess, hCtx)
End Sub

See Also

- EsbInit
- EsbListDatabases
- EsbLogin
- EsbLogout
- EsbSetActive

EsbBeginCalc

Starts sending a calc script and optionally executes it against the active database. This call must be followed by successive calls to EsbSendString() to send the calc script, and finally by a call to EsbEndCalc(). The Calc Script must be less than 64 KB long in total. The calculation can either be initiated, or the calc script can just be verified and any errors returned.

Syntax

EsbBeginCalc (hCtx, isCalculate)
ByVal hCtx As Long
ByVal isCalculate As Integer

Parameter Description

hCtx VB API context handle.

isCalculate Controls calculation of the calc script. If TRUE, the calc script is executed.

Notes

- If this function succeeds and the calculation is started, it will continue on the server as an asynchronous process after the return from this call. The caller must check at regular intervals for the completion of the calculation.
intervals to see if the process has completed by calling `EsbGetProcessState()` until it returns `ESB_STATE_DONE`.

- If the `Calculate` flag is set to FALSE, the database merely performs a syntax check of the calc script.

**Return Value**

None.

**Access**

This function requires the caller to have calc privilege (ESB_PRIV_CALC) to the active database.

**Example**

```visualbasic
Declare Function EsbBeginCalc Lib "ESBAPIN" (ByVal hCtx As Long, ByVal Calculate As Integer) As Long

Sub ESB_BeginCalc()
    Dim sts As Long
    Dim Script As String
    Dim Calculate As Integer
    Dim ProcState As ESB_PROCSTATE_T
    Script = "CALC ALL;"
    Calculate = ESB_YES
    '**************
    ' Begin Calc
    '**************
    sts = EsbBeginCalc (hCtx, Calculate)
    '**********************
    ' Send Calc script
    '**********************
    sts = EsbSendString (hCtx, Script)
    '***********
    ' End Calc
    '***********
    sts = EsbEndCalc (hCtx)
    '************************************
    ' Check process state until it is done
    '************************************
    sts = EsbGetProcessState (hCtx, ProcState)
    Do Until ProcState.State = ESB_STATE_DONE
        sts = EsbGetProcessState (hCtx, ProcState)
    Loop
End Sub
```

**See Also**

- `EsbCalc`
- `EsbCalcFile`
- `EsbDefaultCalc`
- `EsbEndCalc`
- `EsbGetDefaultCalc`
- `EsbGetProcessState`
- `EsbSendString`
- `EsbSetDefaultCalc`
**EsbBeginDataload**

Starts sending an update specification to the active database, and can unlock any data blocks locked for update. The update data can either be stored in the database, or just verified and any errors returned.

**Syntax**

Declare Function EsbBeginDataload Lib "esbapin" (  
ByVal hCtx As Long,  
ByVal isStore As Integer,  
ByVal isUnlock As Integer,  
ByVal isAbortOnError As Integer,  
pRules As ESB_OBJDEF_T) As Long

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>API context handle.</td>
</tr>
<tr>
<td>Store</td>
<td>Controls storage of data. If TRUE, data is stored in the server; if FALSE, no data is stored.</td>
</tr>
<tr>
<td>Unlock</td>
<td>Controls unlocking of data blocks. If TRUE, all relevant blocks which are locked will be unlocked (after data is stored, if necessary). If FALSE, no blocks are unlocked.</td>
</tr>
<tr>
<td>abortOnError</td>
<td>If TRUE, data load stops on the first error. Otherwise, data load continues.</td>
</tr>
<tr>
<td>pRules</td>
<td>Pointer to the rules file object definition structure.</td>
</tr>
</tbody>
</table>

**Notes**

- **EsbBeginDataload()** must be followed by at least one call to **EsbSendString()** to send the update specification, and then a call to **EsbEndDataload()**.
- Each string passed to **EsbSendString()** following **EsbBeginDataload()** must be terminated with a carriage return/linefeed character sequence ("\r\n").
- If both the **Store** and **Unlock** flags are set to FALSE, the database merely performs a syntax check of the update specification.
- Unlike **EsbBeginUpdate()**, which ignores input rows (records) after an improper input row, **EsbBeginDataload()** processes the remaining input rows, and commits them if appropriate.

**Return Value**

None.

**Access**

**EsbBeginDataload()** requires the caller to have write privilege (ESS_PRIV_WRITE) to the active database.

**EsbBeginReport**

Starts sending a report specification to the active database. This call must be followed by successive calls to **EsbSendString** to send the report specification, and finally by a call to **EsbEndReport**. The report data can either be output, or the report specification can just be
verified and any errors returned. Also, the corresponding data blocks in the database can optionally be locked by this call (lock for update).

Syntax

EsbBeginReport (hCtx, isOutput, isLock)
ByVal hCtx As Long
ByVal isOutput As Integer
ByVal isLock As Integer

Parameter Description

hCtx VB API context handle.

isOutput Controls output of data. If TRUE, data is output from the server, according to the specified report. If FALSE, no data is output.

isLock Controls block locking. If TRUE, all blocks which are accessed by the report specification are locked for update. If FALSE, no blocks are locked.

Notes

- This function must be followed by at least one call to EsbSendString(), followed by a call to EsbEndReport().
- If this function causes data to be output (Output flag is TRUE), the returned data can be read by calling EsbGetString().
- If this function causes blocks to be locked (Lock flag is TRUE), the caller is responsible for unlocking the locked blocks (e.g. by calling EsbUpdate() with the Unlock flag set to TRUE).
- If both the Output and Lock flags are set to FALSE, the database merely performs a syntax check of the report specification.

Return Value

None.

Access

This function requires the caller to have read privilege (ESB_PRIV_READ) to one or more members in the active database.

Example

Declare Function EsbBeginReport Lib "ESBAPIN" (ByVal hCtx As Long, ByVal Output As Integer, ByVal Lock As Integer) As Long

Sub ESB_BeginReport ()
    Dim sts As Long
    Dim pOutput As Integer
    Dim pLock As Integer
    Dim Query As String
    Dim RString as String * 256
    Const szRString = 256
    Query = "<Desc Year !"
    Output = ESB_YES
    Lock = ESB_NO
    ************
' Begin Report
'**************************
sts = EsbBeginReport (hCtx, pOutput, pLock)   '**************************
' Send report specification
'**************************
sts = EsbSendString (hCtx, Query)
'**************************
' End Report
'********************
sts = EsbEndReport (hCtx)
'********************
' Print out all strings
'********************
If sts = 0 Then
  sts = EsbGetString (hCtx, RString, szRString)
  Do While Mid$(RString, 1, 1) <> Chr$(0)
    Print RString
    sts = EsbGetString (hCtx, RString, szRString)
  Loop
End If
End Sub

See Also
- EsbBeginUpdate
- EsbEndReport
- EsbGetString
- EsbReport
- EsbReportFile
- EsbSendString

**EsbBeginUpdate**

Starts sending an update specification to the active database. This call must be followed by successive calls to EsbSendString to send the update specification, and finally by a call to EsbEndUpdate. The update data can either be stored in the database, or just verified and any errors returned. Also, any data blocks locked for update can be unlocked by this call.

**Syntax**

```
EsbBeginUpdate (hCtx, isStore, isUpdate)
ByVal hCtx As Long
ByVal isStore As Integer
ByVal isUpdate As Integer
```

**Parameter Description**

- **hCtx**: VB API context handle.
- **isStore**: Controls storage of data. If TRUE, data is stored in the server; if FALSE, no data is stored.
- **isUpdate**: Controls updating of data blocks. If TRUE, all relevant blocks which are locked will be unlocked (after data is stored, if necessary). If FALSE, no blocks are unlocked.
Notes

- This function must be followed by at least one call to `EsbSendString()`, followed by a call to `EsbEndUpdate()`.
- If both the `Store` and `Unlock` flags are set to FALSE, the database merely performs a syntax check of the update specification.

Return Value
None.

Access
This function requires the caller to have write privilege (ESB_PRIV_WRITE) to the active database.

Example

```Visual Basic
Declare Function EsbBeginUpdate Lib "ESBAPIN" (ByVal hCtx As Long, ByVal Store As Integer, ByVal Update As Integer) As Long

Sub ESB_BeginUpdate ()
    Dim sts As Long
    Dim Store As Integer
    Dim pUnlock As Integer
    Dim Query As String
    Query = "Year Market Scenario AcItemss Product 12345"
    Store = ESB_YES
    Unlock = ESB_NO
    '*************
    ' Begin Update
    '*************
    sts = EsbBeginUpdate (hCtx, Store, pUnlock)
    '****************
    ' Send update specification
    '****************
    sts = EsbSendString (hCtx, Query)
    '*************
    ' End Update
    '*************
    sts = EsbEndUpdate (hCtx)
End Sub
```

See Also

- `EsbBeginReport`
- `EsbEndUpdate`
- `EsbSendString`
- `EsbUpdate`
- `EsbUpdateFile`

**EsbBuildDimension**

Allows the creation of a dimension in the active database from a data file and rules file.
Syntax

**EsbBuildDimension** (hCtx, pRules, pData, pUser, ErrName)

ByVal hCtx As Long
    pRules As ESB_OBJDEF_T
    pData As ESB_OBJDEF_T
    pUser As ESB_MBRUSER_T
ByVal ErrName As String

Parameter Description

- **hCtx**  VB API context handle.
- **pRules**  Pointer to rules file object definition structure.
- **pData**  Pointer to data file object definition structure.
- **pUser**  Pointer to SQL user structure (if data source is SQL database). A NULL SQL user structure indicates a non SQL data source.
- **ErrName**  Name of the error output file to be created locally.

Notes

- If *pMbrUser* is not NULL, an SQL data source is assumed.
- If server VB API context is specified for the *hCtx* field in the rules or the data object definition structure, object must exist for the currently active application and database.
- If local VB API context is specified for the *hCtx* field in the rules or the data object definition structure, *FileName* in this structure must be a fully qualified path.
- See the description of **EsbImport** for information on importing data sources.

Return Value

None.

Access

This function requires the caller to have database design privilege for the specified database (ESB_PRIV_DBDESIGN).

Example

Declare Function EsbBuildDimension Lib "ESBAPIN" (ByVal hCtx As Long, Rules As ESB_OBJDEF_T,
    Data As ESB_OBJDEF_T, User As ESB_MBRUSER_T,
    ByVal ErrName As String) As Long

Sub ESB_BuildDimension ()
    Dim sts       As Long
    Dim Rules     As ESB_OBJDEF_T
    Dim Data      As ESB_OBJDEF_T
    Dim User      As ESB_MBRUSER_T
    Dim ErrorName As String
    ' Rules file resides at the server
    ' ****************************************
    Rules.hCtx    = hCtx
Rules.Type = ESB_OBJTYPE_RULES
Rules.FileName = "Test" '**************************************************************************
' Data file resides at the server
'**************************************************************************
Data.hCtx = hCtx
Data.Type = ESB_OBJTYPE_TEXT
Data.FileName = "Data"

'**************************************************************************
' Specify file to redirect errors
' to if any
'**************************************************************************
ErrorName = "BUILDDIM.ERR" '**************************************************************************
' Build Dimensions
'**************************************************************************
sts = EsbBuildDimension (hCtx, Rules, Data, User, ErrorName)

'*
'* When a SQL data source is defined in the rules file, define
'* the variables in the ESB_OBJDEF_T Data structure as follows:
'*
'*  Data.hCtx = hCtx
'*  Data.AppName = ""
'*  Data.DbName = ""
'*  Data.ObjType = ESB_OBJTYPE_NONE
'*  Data.FileName = ""
'*
'* Also, provide strings for the variables in the ESB_MBRUSER_T
'* User structure; for example:
'*  User.User = "Dbusernm"
'*  User.Password = "Dbpasswd"
'*
'* Use a blank string for User and Password, if the SQL source
'* does not require user and password information; for example:
'*
'*  User.User = ""
'*  User.Password = ""
'*
'* Also, define sts as follows:
'*  sts = EsbBuildDimension (hCtx, Rules, Data, User, ErrorName)
'*
'* ************************************************************************** End Sub

See Also

- EsbImport
- EsbBuildDimFile
- EsbBuildDimStart

EsbBuildDimFile

Builds a data file used to add or remove members from the active database outline. See EsbBuildDimension.
Syntax

`EsbBuildDimFile (hCtx, RulesObj, DataObj, MbrUser, ErrorName, fOverwriteErrorFile)`

ByVal hCtx As Long
   pRules As ESB_OBJDEF_T
   pData As ESB_OBJDEF_T
   pUser As ESB_MBRUSER_T
ByVal ErrName As String
ByVal ErrFileOverwrite As Integer

Parameter Description

- **hCtx**: API context handle.
- **RulesObj**: Pointer to rules file object definition structure.
- **DataObj**: Pointer to data file object definition structure.
- **MbrUser**: SQL user structure (if data source is SQL database). NULL structure indicates a non-SQL data source.
- **ErrorName**: Name of error output file on client.
- **fOverwriteErrorFile**: A Boolean value which determines whether this function overwrites an existing file of name *ErrorFile*.

Notes

- If *MbrUser* is not NULL, an SQL data source is assumed.
- The description of *EsbImport* provides information on importing data sources.
- The database must be the active database. See the description of *EsbSetActive*.
- *EsbBuildDimStart* must be called prior to using *EsbBuildDimFile()*.
- *EsbBuildDimFile()* can be called repeatedly prior to restructuring to add members via multiple rules and/or data file to the outline.
- The database must be restructured after completion of call(s) to *EsbBuildDimFile()*.
- The outline must be unlocked after restructuring.

Return Value

Returns zero (0) if successful.

Access

This function requires database design privilege ESB_PRIV_DBDESIGN for the specified database.

Example

```
Declare Function EsbBuildDimension Lib "ESBAPIN" (ByVal hCtx As Long, Rules As ESB_OBJDEF_T, Data As ESB_OBJDEF_T, User As ESB_MBRUSER_T, ByVal ErrName As String) As Long

Sub ESB_BuildDimFile()
   Dim sts As Long
```

Dim Rules As ESB_OBJDEF_T
Dim Data As ESB_OBJDEF_T
Dim User As ESB_MBRUSER_T
Dim ErrorName As String

'*********************************
' Rules file resides at the server
'*********************************
Rules.hCtx = hCtx
Rules.Type = ESB_OBJTYPE_RULES
Rules.FileName = "Test"

'*********************************
' Data file resides at the server
'*********************************
Data.hCtx = hCtx
Data.Type = ESB_OBJTYPE_TEXT
Data.FileName = "Data"

' For a non SQL data source provide
' empty strings in User structure
'*********************************
User.User = ""
User.Password = ""

'*********************************
' Specify file to redirect errors
' to if any
'*********************************
ErrorName = "BUILDDIM.ERR"

'***************
' Build Dimensions
'***************

sts = EsbBuildDimFile (hCtx, Rules, Data, User, ErrorName)
End Sub

See Also
- EsbImport
- EsbBuildDimension
- EsbBuildDimStart
- EsbOtlRestructure
- EsbUnlockObject

**EsbBuildDimStart**

Starts the process to add or remove members from the active database outline.

**Syntax**

EsbBuildDimStart (hCtx);
ByVal hCtx As Long
**Parameter Description**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>API context handle.</td>
</tr>
</tbody>
</table>

**Notes**

- The description of `EsbImport` provides information on importing data sources.
- The database must be the active database. See the description of `EsbSetActive`.
- The outline object must be locked prior to calling `EsbBuildDimStart`. See the description for `EsbLockObject`.

**Return Value**

Returns zero (0) if successful.

**Access**

This function requires database design privilege ESB_PRIV_DBDESIGN for the specified database.

**Example**

```vbscript
Public Sub ESB_BuildDimStart()

    Dim sts As Long

    sts = EsbBuildDimStart (hCtx)

End Sub
```

**See Also**

- `EsbImport`
- `EsbBuildDimension`
- `EsbBuildDimFile`
- `EsbLockObject`

---

**EsbCalc**

Sends and optionally executes a calc script against the active database as a single string. This function is equivalent to making a call to `EsbBeginCalc`, followed by calls to `EsbSendString()`, and finally to `EsbEndCalc()`. The calculation can either be initiated, or the calc script can just be verified and any errors returned.

**Syntax**

```vbscript
EsbCalc (hCtx, isCalculate, cscQuery)
ByVal hCtx     As Long
ByVal isCalculate As Integer
ByVal cscQuery    As String
```

**Parameter Description**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>VB API context handle.</td>
</tr>
</tbody>
</table>
Parameter  Description

isCalculate  Controls calculation of the calc script. If TRUE, the calc script is executed.

cscQuery   The calc script, as a single string (must be less than 64 KB).

Notes
- The calc script string must be less than 64 KB long.
- If this function succeeds and the calculation is started, it will continue on the server as an asynchronous process after the return from this call. The caller must check at regular intervals to see if the process has completed by calling `EsbGetProcessState()` until it returns `ESB_STATE_DONE`.
- If the `Calculate` flag is set to FALSE, the database merely performs a syntax check of the calc script.

Return Value
None.

Access
This function requires the caller to have calc privilege (ESB_PRIV_CALC) to the active database.

Example

```vbnet
Declare Function EsbCalc Lib "ESBAPIN" (ByVal hCtx As Long, ByVal Calculate As Integer, ByVal Script As String) As Long

Sub ESB_Calc()
    Dim sts As Long
    Dim Script As String
    Dim Calculate As Integer
    Dim ProcState As ESB_PROCSTATE_T
    Script = "CALC ALL;"
    Calculate = ESB_YES  '**********
    ' Calculate
    '**********
    sts = EsbCalc (hCtx, Calculate, Script)  '******************************************************************************
    ' Check process state till it is done
    '******************************************************************************
    sts = EsbGetProcessState (hCtx, ProcState)
    Do Until ProcState.State = ESB_STATE_DONE
        sts = EsbGetProcessState (hCtx, ProcState)
    Loop
End Sub
```

See Also
- `EsbBeginCalc`
- `EsbCalcFile`
- `EsbDefaultCalc`
- `EsbEndCalc`
- `EsbGetDefaultCalc`
- `EsbGetProcessState`
- `EsbSendString`
**EsbCalcFile**

Executes a calc script against the active database from a file.

**Syntax**

```plaintext
EsbCalcFile (hDestCtx, hSrcCtx,AppName, DbName, FileName, isCalculate)
```

**Parameter Description**

- **hDestCtx**  
  VB API context handle of target database on the server.

- **hSrcCtx**  
  VB API context handle for calc script file location. The calc script file can reside on the client or on the same server as the target database.

- **AppName**  
  Application name for calc script file location.

- **DbName**  
  Database name for calc script file location.

- **FileName**  
  Name of calc script file.

- **isCalculate**  
  Controls calculation of the calc script. If TRUE, the calc script is executed.

**Notes**

- The calc script must be less than 64 KB long.
- If this function succeeds and the calculation is started, it will continue on the server as an asynchronous process after the return from this call. The caller should check at regular intervals to see if the process has completed by calling `EsbGetProcessState()` until it returns `ESB_STATE_DONE`.

**Return Value**

None.

**Access**

This function requires the caller to have calc privilege (ESB_PRIV_CALC) to the active database.

**Example**

```plaintext
Declare Function EsbCalcFile Lib "ESBAPIN" (ByVal hDestCtx As Long, ByVal hSrcCtx As Long, ByVal AppName As String, ByVal DbName As String, ByVal FileName As String, ByVal isCalculate As Integer) As Long

Sub ESB_CalcFile ()
    Dim sts As Long
    Dim AppName As String
```
Dim DbName As String
Dim FileName As String
Dim Calculate As Integer
Dim hSrcCtx As Long
Dim ProcState As ESB_PROCSTATE_T   AppName = "Sample"   DbName = "Basic"   '***************************************
' Calc script is an object at the server *
'***************************************
hSrcCtx = hCtx
FileName = "calc"
Calculate = ESB_YES '**********
' Calc File
'**********
sts = EsbCalcFile (hCtx, hSrcCtx, AppName, DbName, FileName, Calculate)    '************************************
' Check process state till it is done
'************************************
sts = EsbGetProcessState (hCtx, ProcState)
Do Until ProcState.State = ESB_STATE_DONE
sts = EsbGetProcessState (hCtx, ProcState)
Loop
End Sub

See Also
- EsbBeginCalc
- EsbCalc
- EsbDefaultCalc
- EsbSetDefaultCalcFile
- EsbGetProcessState

EsbCancelProcess
Cancels an asynchronous process which has not yet completed.

Syntax
EsbCancelProcess (hCtx)
ByVal hCtx As Long

Parameter Description
hCtx VB API context handle.

Notes
- If you use this function to cancel a process, the database may be left in an inconsistent state, with only some of the data recalculated.
- Calling this function except after initiating a successful asynchronous database operation (e.g. a calculation) will generate an error.

Return Value
None.
**Access**

This function requires no special privilege.

**Example**

Declare Function EsbCancelProcess Lib "ESBAPIN" (ByVal hCtx As Long) As Long

Sub ESB_CancelProcess ()
    Dim sts As Long
    Dim CalcScript As String
    Dim Calculate As Integer
    Dim ProcState As ESB_PROCSTATE_T
    Dim Items As Integer   CalcScript = "CALC ALL;"
    Calculate = ESB_YES   '***********
    ' Begin Calc
    '***********
    sts = EsbBeginCalc (hCtx, Calculate)
    '**********************
    ' Send Calc script
    ' It is possible to send
    ' more than one string
    '**********************
    sts = EsbSendString (hCtx, CalcScript)   '*********
    ' End Calc
    '*********
    sts = EsbEndCalc (hCtx)   '**************************
    ' Check process state and cancel it if
    ' it takes too long
    '**************************
    sts = EsbGetProcessState (hCtx, ProcState)
    Items = 1
    Do While ProcState.State = ESB_STATE_INPROGRESS
        Items = Items + 1
        If Items = 1000 Then      '**************
            ' Cancel process
            '**************
            sts = EsbCancelProcess (hCtx)
        End If
    Exit Do
    sts = EsbGetProcessState (hCtx, ProcState)
    Loop
End Sub

**See Also**

- EsbBeginCalc
- EsbCalc
- EsbGetProcessState
- EsbImport

---

**EsbCheckAttributes**

Returns the attribute information for each specified member.
Syntax

**EsbCheckAttributes** (hCtx, Count, AttrNameArray(), AttrTypeArray)

**Paraneter**  **Description**

<table>
<thead>
<tr>
<th>hCtx</th>
<th>Context handle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>Number of given dimensions and members</td>
</tr>
<tr>
<td>AttrNameArray()</td>
<td>An array of names of given dimensions and members</td>
</tr>
<tr>
<td>AttrTypeArray</td>
<td>One of the following constant identifiers for the attribute type array:</td>
</tr>
<tr>
<td>l</td>
<td>ESB_ATTRIBUTE_DIMENSION</td>
</tr>
<tr>
<td>l</td>
<td>ESB_ATTRIBUTE_MEMBER</td>
</tr>
<tr>
<td>l</td>
<td>ESB_STANDARD_DIMENSION</td>
</tr>
<tr>
<td>l</td>
<td>ESB_STANDARD_MEMBER</td>
</tr>
<tr>
<td>l</td>
<td>ESB_BASE_DIMENSION</td>
</tr>
<tr>
<td>l</td>
<td>ESB_BASE_MEMBER</td>
</tr>
<tr>
<td>l</td>
<td>ESB_ATRIBUTED_MEMBER</td>
</tr>
<tr>
<td>l</td>
<td>ESB_INVALID_MEMBER</td>
</tr>
</tbody>
</table>

Notes

- Expects a count of names to be entered, and a list of member names.
- Accept a single member name or an array of member names and returns attribute type information for each member entered.
- Member names can be names of attribute dimensions or members and base dimensions or members.

Return Value

Returns sts = 0, when successful, and populates AttrTypeArray(). Returns an error if an invalid member name is passed in.

Access

This function requires no special privileges.

Example

```
' NOTE: 'Out' is a sub to print the output within quotes to a listbox or text box.
Sub ESB_CheckAttributes()
Dim hCtx as long
Dim sts as long
Dim MbrNameArr() As String
Dim AttrTypeArr As Variant
Dim Count As Integer
Dim index As Integer
Dim test As Integer
```
Count = InputBox("Enter the number of attribute members")
ReDim MbrNameArr(Count)
For index = 0 To Count - 1
    MbrNameArr(index) = InputBox("Enter attribute member name")
Next index

sts = EsbCheckAttributes(hCtx, Count, MbrNameArr, AttrTypeArr)
If sts = 0 Then
    For index = LBound(AttrTypeArr) To UBound(AttrTypeArr)
        test = AttrTypeArr(index)
        Select Case test
          Case ESB_STANDARD_MEMBER
            Out MbrNameArr(index) & " is of type  ESB_STANDARD_MEMBER"
          Case ESB_STANDARD_DIMENSION
            Out MbrNameArr(index) & " is of type  ESB_STANDARD_DIMENSION"
          Case ESB_BASE_MEMBER
            Out MbrNameArr(index) & " is of type  ESB_BASE_MEMBER"
          Case ESB_BASE_DIMENSION
            Out MbrNameArr(index) & " is of type  ESB_BASE_DIMENSION"
          Case ESB_ATTRIBUTE_MEMBER
            Out MbrNameArr(index) & " is of type  ESB_ATTRIBUTE_MEMBER"
          Case ESB_ATTRIBUTE_DIMENSION
            Out MbrNameArr(index) & " is of type  ESB_ATTRIBUTE_DIMENSION"
          Case ESBATTRIBUTED_MEMBER
            Out MbrNameArr(index) & " is of type  ESBATTRIBUTED_MEMBER"
          Case Else
            Out MbrNameArr(index) & " is of INVALID Type or Invalid Member Name "
        End Select
    Next index
Else
    Out "EsbCheckAttributes failed:" & sts: Exit Sub
End If
End Sub

See Also

- EsbGetAssociatedAttributesInfo
- EsbGetAttributeInfo
- EsbGetAttributeSpecifications
- EsbOtlAssociateAttributeDimension
- EsbOtlAssociateAttributeMember
- EsbOtlDisassociateAttributeDimension
- EsbOtlDisassociateAttributeMember
- EsbOtlFindAttributeMembers
- EsbOtlGetAssociatedAttributes
- EsbOtlGetAttributeInfo
- EsbOtlGetAttributeSpecifications
- EsbOtlQueryAttributes
- EsbOtlSetAttributeSpecifications
**EsbCheckMemberName**

Checks if a string is a valid member name within the active database outline.

**Syntax**

```vbnet
EsbCheckMemberName (hCtx, MemName,isOk)
```

- **hCtx** As Long
- **MemName** As String
- **isOk** As Integer

**Parameter Description**

- **hCtx**: VB API context handle.
- **MemName**: Member name to be verified.
- **isOk**: Address of variable to receive a valid member flag. Set to TRUE if member is valid.

**Return Value**

If successful, this function returns a flag, `pValid`, indicating if the name string `MbrName` is a valid member name in the active database outline.

**Access**

This function requires the caller to have access to the database, and to have selected it as their active database using `EsbSetActive()`.

**Example**

```vbnet
Declare Function EsbCheckMemberName Lib "ESBAPIN" (ByVal hCtx As Long, ByVal MbrName As String, isOk As Integer) As Long

Sub ESB_CheckMemberName ()
    Dim MbrName As String
    Dim Valid As Integer
    Dim sts As Long
    MbrName = "Year"
    ' ******************
    ' Check member name
    ' ******************
    sts = EsbCheckMemberName (hCtx, MbrName, Valid)
    If Valid = ESB_YES Then
        Print "Valid Member Name"
    End If
End Sub
```

**See Also**

- `EsbGetMemberInfo`
- `EsbQueryDatabaseMembers`
- `EsbVerifyFilter`
- `EsbSetActive`

---

1232 Visual Basic Main API Functions
**EsbClearActive**

Clears the user's current active application and database.

**Syntax**

```
EsbClearActive (hCtx)
ByVal hCtx As Long
```

**Parameter Description**

- **hCtx**: VB API context handle.

**Return Value**

None.

**Access**

This function requires no special privileges.

**Example**

```
Declare Function EsbClearActive Lib "ESBAPIN" (ByVal hCtx As Long) As Long

Sub ESB_ClearActive ()
    Dim sts As Long  '*************
    ' Clear Active
    '*************
    sts = EsbClearActive (hCtx)
End Sub
```

**See Also**

- EsbGetActive
- EsbSetActive

---

**EsbClearAliases**

Permanently removes all alias tables for the active database.

**Syntax**

```
EsbClearAliases (hCtx)
ByVal hCtx As Long
```

**Parameter Description**

- **hCtx**: VB API context handle.

**Notes**

- "Default" or currently active alias table can not be removed.
- Make sure that no one else is using the same database as the one you try to clear alias tables from by calling EsbListConnections().
Use `EsbSetAlias()` to set an active alias to "default" prior to using this VB API function.

**Return Value**
None.

**Access**
This function requires the caller to have access to the database, and to have selected it as their active database using `EsbSetActive()`.

**Example**

```vba
Declare Function EsbClearAliases Lib "ESBAPIN" (ByVal hCtx As Long) As Long

Sub ESB_ClearAliases()
    Dim sts As Long  '**************
    ' Remove Aliases
    '**************
    sts = EsbClearAliases (hCtx)
End Sub
```

**See Also**
- `EsbListAliases`
- `EsbRemoveAlias`
- `EsbSetAlias`
- `EsbListConnections`
- `EsbSetActive`

### EsbClearDatabase

Clears all loaded data in the active database.

**Syntax**

```
EsbClearDatabase (hCtx)
ByVal hCtx As Long
```

**Parameter Description**

- **hCtx** VB API context handle.

**Notes**
Data deleted using this function cannot be restored. Use it with care!

**Return Value**
None.

**Access**
This function requires the caller to have Write privilege (ESB_PRIV_WRITE) for the database, and to have selected it as their active database using `EsbSetActive()`.
Example

Declare Function EsbClearDatabase Lib "ESBAPIN" (ByVal hCtx As Long) As Long

Sub ESB_ClearDatabase ()
    Dim sts As Long
    '**************
    ' Clear Database
    '**************
    sts = EsbClearDatabase (hCtx)
End Sub

See Also

- EsbDeleteDatabase
- EsbUnloadDatabase
- EsbSetActive

EsbClrSpanRelationalSource

Clears the Boolean bSpanRelPart field informing Essbase that pertinent data exists in an attached relational store. Some other API functions, such as EsbQueryDatabaseMembers, read bSpanRelPart and access the relational store if bSpanRelPart is set.

Syntax

Declare Function EsbClrSpanRelationalSource Lib "esbapin" (ByVal hCtx As Long) As Long

Parameter Description

hCtx API context handle.

Notes

Several API functions have been enhanced to retrieve information from relational stores.

- **EsbQueryDatabaseMembers**—Returns member names from the relational store.
- **EsbGetMemberInfo**—Returns information on members in the relational store.
- **EsbCheckMemberName**—Checks in the relational store for valid member names.
- **EsbGetMemberCalc**—Recognizes a relational member passed as input and returns a null string for all relational members.

Return Value

None.

Access

This function requires the caller to have read privilege (ESS_PRIV_READ) to one or more members in the active database.

Example

Declare Function EsbClrRelationalSource Lib "ESBAPIN" (ByVal hCtx As Long) As Long
Sub ESB_ClrRelationalSource()
    Dim sts As Long

    '**************************
    ' Clear the bSpanRelPart field
    '**************************
    sts = EsbClrRelationalSource(hCtx)
End Sub

See Also

- EsbSetSpanRelationalSource

EsbCommitDatabase

No longer in use because commits are handled automatically by the Essbase Server. This function now returns the error message ESB_STS_OBSOLETE. See the Oracle Essbase Database Administrator’s Guide for details about committing data.

EsbCopyApplication

Copies an existing application, either on the client or the server, to a new application, including all associated databases and objects. If the application is copied on the server, the new application is started.

Syntax

EsbCopyApplication(hCtx, hSrcCtx, AppName, nAppName)

ByVal hCtx As Long
ByVal hSrcCtx As Long
ByVal AppName As String
ByVal nAppName As String

Parameter Description

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>VB API context handle.</td>
</tr>
<tr>
<td>hSrcCtx</td>
<td>Not used; should be same as hCtx.</td>
</tr>
<tr>
<td>AppName</td>
<td>Name of an existing application to copy.</td>
</tr>
<tr>
<td>nAppName</td>
<td>Name of a new application. See “Application Name Limits” on page 1727.</td>
</tr>
</tbody>
</table>

Notes

- Copying a client application copies the local application directory and contents.
- This function can only be used to copy a client application to a new application on the client, or a server application to a new application on the same server. Use EsbCopyObject() to copy an application between different servers.
Return Value
None.

Access
For a server application, the caller must have application Create/Delete/Edit privilege (ESB_PRIV_APPCREATE).

Example
Declare Function EsbCopyApplication Lib "ESBAPIN" (ByVal hCtx As Long, ByVal hSrcCtx As Long, ByVal SrcApp As String, ByVal DestApp As String) As Long

Sub ESB_CopyApplication ()
    Dim sts As Long
    Dim SrcApp As String
    Dim DestApp As String
    Dim hSrcCtx As Long   hSrcCtx = hCtx
    SrcApp = "Sample"   DestApp = "NewTest"   '*****************
    ' Copy Application
    '*****************
    sts = EsbCopyApplication (hCtx, hSrcCtx, SrcApp, DestApp)
End Sub

See Also
• EsbCopyDatabase
• EsbCopyObject

EsbCopyDatabase
Copies an existing database, either on the client or the server, to a new database, including all associated databases and objects. If the database is copied on the server, the new database is started.

Syntax
EsbCopyDatabase (hCtx, hSrcCtx, AppName, nAppName, DbName, nDbName)
ByVal hCtx As Long
ByVal hSrcCtx As Long
ByVal AppName As String
ByVal nAppName As String
ByVal DbName As String
ByVal nDbName As String

Parameter Description

hCtx         VB API context handle.
hSrcCtx      Not used - should be same as hCtx.
AppName      Name of a source application.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nAppName</td>
<td>Name of a destination application.</td>
</tr>
<tr>
<td>DbName</td>
<td>Name of an existing database to copy.</td>
</tr>
<tr>
<td>nDbName</td>
<td>Name of a new database. See “Database Name Limits” on page 1728.</td>
</tr>
</tbody>
</table>

**Notes**

- Copying a client database copies the local database directory and contents.
- This function can only be used to copy a client database to a new database on the client, or a server database to a new database on the same server. Use `EsbCopyObject()` to copy a database between different servers.

**Return Value**

None.

**Access**

For a server database, the caller must have database Create/Delete/Edit privilege (ESB_PRIV_DBCREATE).

**Example**

```vbnet
Declare Function EsbCopyDatabase Lib "ESBAPIN" (ByVal hCtx As Long, ByVal hSrcCtx As Long, ByVal SrcApp As String, ByVal DestApp As String, ByVal SrcDb As String, ByVal DestDb As String) As Long

Sub ESB_CopyDatabase ()
    Dim sts As Long
    Dim SrcApp As String
    Dim DestApp As String
    Dim SrcDb As String
    Dim DestDb As String
    Dim hSrcCtx As Long   hSrcCtx = hCtx
    SrcApp = "Sample"
    DestApp = "NewSamp"
    SrcDb = "Basic"
    DestDb = "NewBasic"   '***************
    ' Copy database
    '***************   sts = EsbCopyDatabase (hCtx, hSrcCtx, SrcApp, DestApp, SrcDb, DestDb)
End Sub
```

**See Also**

- `EsbCopyApplication`
- `EsbCopyObject`

**EsbCopyFilter**

Copies an existing filter.
**Syntax**

EsbCopyFilter (hCtx, hSrcCtx, AppName, nAppName, DbName, nDbName, FltName, nFltName)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>VB API context handle.</td>
</tr>
<tr>
<td>hSrcCtx</td>
<td>Not used - should be same as hCtx.</td>
</tr>
<tr>
<td>AppName</td>
<td>Source application name.</td>
</tr>
<tr>
<td>nAppName</td>
<td>Destination application name.</td>
</tr>
<tr>
<td>DbName</td>
<td>Source database name.</td>
</tr>
<tr>
<td>nDbName</td>
<td>Destination database name.</td>
</tr>
<tr>
<td>FltName</td>
<td>Source name of an existing filter to be copied.</td>
</tr>
<tr>
<td>nFltName</td>
<td>Destination name of the copied filter. See “Filter Name Limits” on page 1728.</td>
</tr>
</tbody>
</table>

**Notes**

- The source filter must already exist.
- To prevent overwriting an existing filter by mistake, the caller should check whether the destination filter already exists.

**Return Value**

None.

**Access**

This function requires the caller to have Database Design privilege (ESB_PRIV_DBDESIGN) for the specified database.

**Example**

Declare Function EsbCopyFilter Lib "ESBAPIN" (ByVal hCtx As Long, ByVal hSrcCtx As Long, ByVal SrcApp As String, ByVal DestApp As String, ByVal SrcDb As String, ByVal DestDb As String, ByVal SrcName As String, ByVal DestName As String) As Long

Sub ESB_CopyFilter ()
    Dim sts As Long
    Dim SrcApp As String
    Dim SrcDb As String
    Dim SrcName As String
    Dim DestApp As String
    Dim DestDb As String
    Dim st As Long

    st = EsbCopyFilter (hCtx, hSrcCtx, SrcApp, DestApp, SrcDb, DestDb, SrcName, DestName)
    If st = -1 Then
        MsgBox "Error: Unable to copy filter.
    Else
        MsgBox "Filter copied successfully.
    End If
Dim DestName As String
Dim hDestCtx As Long   hDestCtx = hCtx
SrcApp = "Sample"
SrcDb = "Basic"
SrcName = "Filter"
DestApp = "NewSamp"
DestDb = "NewBasic"
DestName = "NewFilter"   '************
' Copy Filter
'************
sts = EsbCopyFilter (hCtx, hDestCtx, SrcApp,  
DestApp, SrcDb, DestDb, SrcName, DestName)
End Sub

See Also

- EsbDeleteFilter
- EsbListFilters
- EsbRenameFilter
- EsbSetFilter

**EsbCopyObject**

Copies an object on the server or client object system.

**Syntax**

```vba
EsbCopyObject (hCtx, hDestCtx, ObjType, AppName, nAppName, DbName, nDbName,  
objName, nobjName)
ByVal hCtx As Long
ByVal hDestCtx As Long
ByVal ObjType As Long
ByVal AppName As String
ByVal nAppName As String
ByVal DbName As String
ByVal nDbName As String
ByVal objName As String
ByVal nObjName As String
```

**Parameter**  **Description**

- **hCtx**  VB API context handle for source object. Can be local context handle returned by EsbCreateLocalContext().
- **hDestCtx**  VB API context handle for destination object.
- **ObjType**  Object type (must be single type). Refer to Table 15 for a list of possible values.
- **AppName**  Source application name.
- **nAppName**  Destination application name.
- **DbName**  Source database name. If an empty string, uses the source application sub-directory.
- **nDbName**  Destination database name. If an empty string, uses the destination application sub-directory.
### Parameter Description

**objName**  Name of a source object to copy from.

**nobjName**  Name of a destination object to copy to. See “Object Name Limits” on page 1728.

### Notes

- Objects may be copied from client to server, server to client, within the same server, or between different servers. In all cases the destination object must either not already exist or it must be locked by the caller.

- Outline objects cannot be copied. Use the `EsbCopyDatabase()` function to copy a database, including its associated outline.

### Return Value

None.

### Access

This function requires the caller to have the appropriate level of access to the specified source application and/or database containing the object (depending on the object type), and to have Application or Database Design privilege (ESB_PRIV_APPDESIGN or ESB_PRIV_DBDESIGN) for the specified destination application or database.

### Example

Declare Function EsbCopyObject Lib "ESBAPIN" (ByVal hCtx As Long, ByVal hDestCtx As Long, ByVal ObjType As Integer, ByVal SrcApp As String, ByVal DestApp As String, ByVal SrcDb As String, ByVal DestDb As String, ByVal SrcObj As String, ByVal DestName As String) As Long

Sub ESB_CopyObject()
    Dim sts As Long
    Dim hDestCtx As Long
    Dim SrcApp As String
    Dim SrcDb As String
    Dim SrcObj As String
    Dim DestApp As String
    Dim DestDb As String
    Dim DestObj As String
    Dim ObjType As Integer
    hDestCtx = hCtx
    SrcApp = "Sample"
    SrcDb = "Basic"
    SrcObj = "Basic"
    DestApp = "NewSamp"
    DestDb = "NewBasic"
    DestObj = "NewBasic"
    ObjType = ESB_OBJTYPE_RULES  '**************
    ' Copy rules object
    '**************
    sts = EsbCopyObject (hCtx, hDestCtx, ObjType, SrcApp, DestApp, SrcDb, DestDb, SrcObj, DestObj)
End Sub
See Also

- `EsbCreateObject`
- `EsbDeleteObject`
- `EsbListObjects`
- `EsbRenameObject`
- `EsbLockObject`

**EsbCreateApplication**

Creates a new application, either on the client or the server. If the application is created on the server, it is also started.

**Syntax**

```plaintext
EsbCreateApplication (hCtx, AppName)
ByVal hCtx   As Long
ByVal AppName As String
```

**Parameter Description**

- **hCtx**  VB API context handle.
- **AppName**  Name of application to create. See "Application Name Limits" on page 1727.

**Notes**

- Creating a client application creates a directory to contain local application files.
- A newly created database or application is not automatically set to active. Call `EsbSetActive()` after calling `EsbCreateDatabase()` or `EsbCreateApplication()` to keep subsequent functions, such as `EsbRestructure()`, from operating on the wrong database or application (the application or database that is already active).

**Return Value**

None.

**Access**

For a server application, the caller must have application Create/Delete/Edit privilege (ESB_PRIV_APPCREATE).

**Example**

```plaintext
Declare Function EsbCreateApplication Lib "ESBAPIN" (ByVal hCtx As Long, ByVal AppName As String) As Long

Sub ESB_CreateApplication ()
    Dim sts As Long
    Dim AppName As String    AppName = "Sample"  '********************
    ' Create Application
    '********************
    sts = EsbCreateApplication (hCtx, AppName)
End Sub
```

1242 Visual Basic Main API Functions
See Also

- EsbCreateDatabase
- EsbCreateObject

**EsbCreateDatabase**

Creates a new database within an application, either on the client or the server. If the database
is created on the server, it is also started.

**Syntax**

```plaintext
EsbCreateDatabase (hCtx, AppName, DbName, DbType)
ByVal hCtx As Long
ByVal AppName As String
ByVal DbName As String
ByVal DbType As Integer
```

**Parameter Description**

- **hCtx**: VB API context handle.
- **AppName**: Name of an application to contain database.
- **DbName**: Name of a database to create. See “Database Name Limits” on page 1728.
- **DbType**: Type of database to create: (ESB_DBTYPE_NORMAL/ESB_DBTYPE_CURRENCY).

**Notes**

- Creating a client database creates a directory to contain local database files.
- A newly created database or application is not automatically set to active. Call
  `EsbSetActive()` after calling `EsbCreateDatabase()` or `EsbCreateApplication()` to keep
  subsequent functions, such as `EsbRestructure()`, from operating on the wrong database or
  application (the application or database that is already active).

**Return Value**

None.

**Access**

For a server database, the caller must have database Create/Delete/Edit privilege
(ESB_PRIV_DBCREATE).

**Example**

```plaintext
Declare Function EsbCreateDatabase Lib "ESBAPIN" (ByVal hCtx As Long, ByVal AppName As String, ByVal DbName As String, ByVal DbType As Integer) As Long

Sub ESB_CreateDatabase ()
    Dim sts As Long
    Dim AppName As String
    Dim DbName As String   AppName = "Sample"
    DbName = "Basic"   '***************
    ' Create database
```
EsbCreateDrillThruURL

Creates a drill-through URL, with the given link and the name, within the active database outline.

“Drill-through URL Limits” on page 1729.

Syntax

Declare Function EsbCreateDrillThruURL Lib "esbapin" (ByVal hCtx As Long, ByRef symRegions() As String, ByRef pUrl As ESB_DURLINFO_T) As Long

Parameter    Description

hCtx       Visual Basic API context handle

symRegions() Array containing the symmetric region specification

pUrl       URL definition

Return Value

- If successful, creates a drill-through URL in the active database outline.
- If unsuccessful, returns an error code.

Access

- Caller must have database Design privilege (ESB_PRIV_DBDESIGN) for the specified database.
- Caller must have selected the specified database as their active database using EsbSetActive().

Example

Sub ESB_CreateGLDrillThru()
    Dim sts As Long
    Dim url As ESB_DURLINFO_T
    Dim cppDrillRegions(0 To 1) As String

    '***************************************************************
    ' Need to create a local context, if files are not on the server
    '***************************************************************
    url.bIsLevel0 = 0
    cppDrillRegions(0) = "sales"
    cppDrillRegions(1) = "cogs"
    url.cpURLXML = "Testing"
    url.cpURLName = "VB URL7"
End Sub
url.iURLXMLSize = 8

sts = EsbCreateDrillThruURL(hCtx, cppDrillRegions, url)

Debug.Print "EsbCreateDrillThruURL sts: " & sts
End Sub

See also an extended example in “Drill-through Visual Basic API Example” on page 1132.

**EsbCreateExtUser**

Creates a new externally authenticated user.

**Syntax**

Declare Function EsbCreateExtUser Lib "esbapin" (  
ByVal hCtx As Long,  
ByVal UserName As String,  
ByVal Password As String,  
ByVal Protocol As String,  
ByVal Connparam As String) As Long

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>API context handle.</td>
</tr>
<tr>
<td>UserName</td>
<td>Name of user to create. See “User Name Limits” on page 1728.</td>
</tr>
<tr>
<td>Password</td>
<td>Security password for new user. See “Password Limits” on page 1728.</td>
</tr>
<tr>
<td>SecurityProvider</td>
<td>The name of the external authentication mechanism.</td>
</tr>
<tr>
<td>ProviderConnectionParameters</td>
<td>Parameters used by the external authentication mechanism, if any.</td>
</tr>
</tbody>
</table>

**Notes**

- The specified user must not already exist.
- The user’s access level and other parameters may be set with the **EsbSetUser()** function.
- Your program should ensure that the password has been entered correctly before calling this function; for example, by requiring the user to type it twice. Once entered, it is not possible to retrieve a password. However, a password can be changed using the **EsbSetPassword()** function.
- The Password parameter has been made redundant by changes for Shared Services. You can use an empty string for this parameter.

**Return Value**

None.

**Access**

This function requires the caller to have Create/Delete User privilege (ESS_PRIV_USERCREATE) for the logged in server.
EsbCreateFilter

Creates a new filter and starts setting its contents.

Syntax

Declare Function EsbCreateFilter Lib "esbapin" ( ByVal hCtx As Long,
ByValAppName As String,
ByVal DbName As String,
ByVal FltName As String,
ByVal isActive As Integer,
ByVal pAccess As Integer) As Long

Parameter Description

hCtx API context handle
AppName Application name
DbName Database name
FilterName Filter name. See “Filter Name Limits” on page 1728.
Active Filter active flag. If TRUE, the filter is set active, otherwise it is set inactive.
Access The default filter access level

Notes

- If the filter does not already exist, it will be created by this call.
- If the filter already exists, an error message is returned.
- This call must be followed by successive calls to EsbSetFilterRow() to set all the rows for the filter.

Return Value

None.

Access

This function requires the caller to have Database Manager permission (ESS_PRIV_DBDESIGN) for the specified database.

EsbCreateGroup

Creates a new group.

Syntax

EsbCreateGroup (hCtx, GrpName)
ByVal hCtx As Long
ByVal GrpName As String
### EsbCreateGroup

**Description**

Creates a new group using the given context handle and group name. The group must not already exist.

**Syntax**

\[
\text{Declare Function EsbCreateGroup Lib "ESBAPIN" (ByVal hCtx As Long, ByVal GroupName As String) As Long}
\]

**Parameter Description**

- **hCtx**: VB API context handle.
- **GroupName**: Name of a group to create. See “Group Name Limits” on page 1728.

**Notes**

- The specified group must not already exist.
- The group's access level may be set with the *EsbSetGroup()* function.

**Return Value**

None.

**Access**

This function requires the caller to have Create/Delete User privilege (ESB_PRIV_USERCREATE) for the logged in server.

**Example**

```
Declare Function EsbCreateGroup Lib "ESBAPIN" (ByVal hCtx As Long, ByVal GroupName As String) As Long

Sub ESB_CreateGroup ()
    Dim sts As Long
    Dim GroupName As String  GroupName = "PowerUsers"  '* *************
    ' Create Group
    '* *************
    sts = EsbCreateGroup (hCtx, GroupName)
End Sub
```

**See Also**

- EsbDeleteGroup
- EsbListGroups
- EsbRenameGroup
- EsbSetGroup

### EsbCreateLocalContext

**Description**

Creates a local VB API context for use in local VB API operations.

**Syntax**

```
EsbCreateLocalContext (hInst, User, Password, phCtx)
ByVal hInst   As Long
ByVal User    As String
ByVal Password As String
    phCtx   As Long
```

**Parameter Description**

- **hInst**: VB API instance handle.
Parameter Description

User Currently not used; should be an empty string.

Password Currently not used; should be an empty string.

phCtx Address of variable to receive Essbase Server local context handle.

Notes

- This function must be called if access to local VB API operations (for example local file/object functions) is desired. It should be called after calling EsbInit().

- It is only necessary to call the function once per client application - the context handle can be used for all local API operations.

- You should call EsbDeleteLocalContext() when the application has finished accessing local objects.

Return Value

If successful, a valid local context handle is returned in phLocalCtx.

Access

This function requires no special privileges.

Example

Declare Function EsbCreateLocalContext Lib "ESBAPIN" (ByVal hInst As Long, ByVal User As String, ByVal Password As String, ByVal hCtx As Long) As Long

Sub ESB_CreateLocalContext ()
  Dim sts As Long
  Dim User As String
  Dim Password As String
  Dim hCtx As Long  '********************
  ' Create Local Context
  '*********************
  sts = EsbCreateLocalContext (hInst, User, Password, hCtx)
End Sub

See Also

- EsbDeleteLocalContext
- EsbInit

EsbCreateLocationAlias

Creates a new location alias; that is, it maps an alias name string to an ordered set of the following five strings: host name, application name, database name, user login name, and user password.

Syntax

EsbCreateLocationAlias (hCtx, AliasName, HostName, _
AppName, DbName, Login, Password)

ByVal hCtx As Long
ByVal AliasName As String
ByVal HostName As String
ByVal AppName As String
ByVal DbName As String
ByVal Login As String
ByVal Password As String

Parameter  Description
hCtx      API context handle
AliasName  Location alias
HostName   Target host
AppName    Target application
DbName     Target database
Login      User login name
Password   User password

Return Value
Returns an error if a location alias with the name AliasName already exists.

Example
Public Sub LocationAliasTest()
    Dim status As Long
    Dim ListCount As Integer
    Dim Aliases As Variant
    Dim HostNames As Variant
    Dim AppNames As Variant
    Dim DbNames As Variant
    Dim UserNames As Variant

    status = EsbCreateLocationAlias(hCtx, "blah1", "LocalHost", "Demo", "Basic", _
                                   "admin", "password")
    If (status <> 0) Then
        MsgBox "Create routine Failed"
        Exit Sub
    End If

    status = EsbCreateLocationAlias(hCtx, "blah2", "LocalHost", "Demo", "Basic", _
                                     "admin", "password")
    If (status <> 0) Then
        MsgBox "Create routine Failed"
        Exit Sub
    End If

    status = EsbGetLocationAliasList(hCtx, ListCount, Aliases, HostNames, _
                                      AppNames, DbNames, UserNames)
    If (status <> 0) Then
        MsgBox "Get routine Failed"
        Exit Sub
    End If
End Sub
If (ListCount > 0) Then
    ' Retrieve the elements as Aliases(0) to Aliases(ListCount -1)
End If

status = EsbDeleteLocationAlias(hCtx, "blah1")
If (status <> 0) Then
    MsgBox "Delete routine Failed"
    Exit Sub
End If

status = EsbGetLocationAliasList(hCtx, ListCount, Aliases, HostNames, AppNames, DbNames, UserNames)
If (status <> 0) Then
    MsgBox "Get routine Failed"
    Exit Sub
End If
End Sub

See Also

- EsbDeleteLocationAlias
- EsbGetLocationAliasList

**EsbCreateObject**

Creates a new object on the server or client object system.

**Syntax**

```vba
EsbCreateObject (hCtx, ObjType,AppName, DbName, ObjName)
```

- **hCtx** VB API context handle. Can local context handle returned by EsbCreateLocalContext().
- **ObjType** Object type (must be single type). Refer to Table 15 for a list of possible values.
- **AppName** Application name.
- **DbName** Database name. If an empty string, uses the application sub-directory.
- **ObjName** Name of an object to create. See “Object Name Limits” on page 1728.

**Notes**

- To create an object, it must not already exist.
A newly created object on the server contains no data and merely acts as a place holder to prevent another user from creating the object. If you wish to update the created object, you should lock it using `EsbLockObject()`, then save it using `EsbPutObject()`.

**Return Value**

None.

**Access**

This function requires the caller to have Application or Database Design privilege (ESB_PRIV_APPDESIGN or ESB_PRIV_DBDESIGN) for the specified application or database to contain the object.

**Example**

```visual-basic
Declare Function EsbCreateObject Lib "ESBAPIN" (ByVal hCtx As Long, ByVal ObjType As Integer, ByVal AppName As String, ByVal DbName As String, ByVal ObjName As String) As Long

Sub ESB_CreateObject ()
    Dim sts As Long
    DimAppName As String
    Dim DbName As String
    Dim ObjName As String
    Dim ObjType As Integer   AppName = "Sample"
    DbName = "Basic"
    ObjName = "Basic"
    ObjType = ESB_OBJTYPE_RULES   '********************
    ' Create Rules Object
    '********************
    sts = EsbCreateObject (hCtx, ObjType, AppName, DbName, ObjName)
End Sub
```

**See Also**

- `EsbDeleteObject`
- `EsbListObjects`
- `EsbLockObject`
- `EsbPutObject`
- `EsbCopyObject`
- `EsbRenameObject`

**EsbCreateStorageTypedApplication**

Creates a new application with the option of data storage mode: block (multidimensional) or aggregate.

**Syntax**

```visual-basic
EsbCreateStorageTypedApplication (hCtx, AppName, StorageType)
ByVal hCtx As Long
ByVal AppName As String
ByVal StorageType As Integer
```
Parameter                      Description

hCtx                          VB API context handle.
AppName                       Name of application to create. See “Application Name Limits” on page 1727.
StorageType                   The data storage type of the new application.
                                 The valid values for StorageType are:
                                 • ESB_DEFAULT_DATA_STORAGE
                                 • ESB_MULTIDIM_DATA_STORAGE—Block storage (multidimensional), the default storage type
                                 • ESB_ASO_DATA_STORAGE—Aggregate storage

Notes
• The new application is created in non-Unicode mode.
• Creating a client application creates a directory to contain local application files.
• A newly created database or application is not automatically set to active. Call EsbSetActive() after calling EsbCreateDatabase(), EsbCreateApplication(), or EsbCreateStorageTypedApplication() to keep subsequent functions, such as EsbRestructure(), from operating on the wrong database or application (the application or database that is already active).

Return Value
None.

Access
For a server application, the caller must have application Create/Delete/Edit privilege (ESB_PRIV_APPCREATE).

Example
Declare Function EsbCreateStorageTypedApplication Lib "ESBAPIN" (ByVal hCtx As Long, ByVal AppName As String, ByVal StorageType As Integer) As Long

Sub ESB_CreateStorageTypedApplication ()
    Dim sts As Long
    Dim AppName As String
    Dim StorageType as Integer   AppName = "Sample"   '******************************
    ' Create Storage Typed Application
    '******************************
    sts = EsbCreateStorageTypedApplication (hCtx, AppName, StorageType)
End Sub

See Also
• EsbCreateApplication
• EsbCreateDatabase
• EsbCreateObject
• EsbGetApplicationInfo
EsbCreateUser

Creates a new user.

Syntax

```plaintext
EsbCreateUser (hCtx, userName, Password)
ByVal hCtx    As Long
ByVal userName As String
ByVal Password As String
```

Parameter Description

- **hCtx**: VB API context handle.
- **userName**: Name of user to create. See “User Name Limits” on page 1728.
- **Password**: Security password for new user. See “Password Limits” on page 1728.

Notes

- The specified user must not already exist.
- The user’s access level and other parameters may be set with the EsbSetUser() function.
- You program should ensure that the password has been entered correctly (for example, by requiring the user to type it twice) before calling this function. Once entered, it is not possible to retrieve a password. However, a password can be changed using the EsbSetPassword() function.

Return Value

None.

Access

This function requires the caller to have Create/Delete User privilege (ESB_PRIV_USERCREATE) for the logged in server.

Example

```vbnet
Declare Function EsbCreateUser Lib "ESBAPIN" (ByVal hCtx As Long, ByVal User As String, ByVal Password As String) As Long

Sub ESB_CreateUser ()
    Dim sts As Long
    Dim User As String
    Dim Password As String
    User = "Joseph"
    Password = "Password"  '***********
    ' Create user
    '***********
    sts = EsbCreateUser (hCtx, User, Password)
End Sub
```

See Also

- **EsbDeleteUser**
- **EsbListUsers**
EsbCreateVariable

Creates a new substitution variable or modifies an existing substitution variable if the variable name already exists with the identical server, application, and database values.

Syntax

EsbCreateVariable (hCtx, pVariable)
ByVal hCtx As Long
 pVariable As ESB_PVARIABLE_T

Parameter Description

hCtx API context handle.

pVariable Pointer to the structure containing the description of the substitution variable being created.

Notes

- The scope of the variable can apply to the server, the application, or the database. The scope is controlled through the "ESB_VARIABLE_T" on page 1196 structure. When the server, application, and database are all named, the substitution variable applies only to the specified database. When only the server and application are named, the substitution variable applies to all databases in the specified application. When only the server is named, the substitution variable applies to all applications and databases on the specified server.

- When a variable exists and a new variable is created with the same name and scope, the new value replaces the old value with no error message from Esbbase.

- On a given server, you can create multiple substitution variables with the same name but different scopes (application and database).

Return Value

If successful, returns zero.

Example

Declare Function EsbCreateVariable Lib "esbapin" (ByVal hCtx As Long, pVariable As ESB_VARIABLE_T) As Long

Sub Esb_CreateVariable()

Dim sts As Long
Dim oVariable As ESB_VARIABLE_T

' Create "QuarterName" Substitution Variable at the Sample application level
oVariable.Server = "Localhost"
oVariable.AppName = "Sample"
' ** Note that DbName has been left empty
oVariable.VarName = "QuarterName"
oVariable.VarValue = "Qtr1"

sts = EsbCreateVariable(hCtx, oVariable)

End Sub

See Also

- “ESB_VARIABLE_T” on page 1196
- EsbDeleteVariable
- EsbGetVariable
- EsbListVariables

**EsbDefaultCalc**

Executes the default calculation for the active database.

**Syntax**

`EsbDefaultCalc (hCtx)`

ByVal hCtx As Long

**Parameter Description**

hCtx VB API context handle.

**Notes**

- If this function succeeds and the calculation is started, it will continue on the server as an asynchronous process after the return from this call. The caller should check at regular intervals to see if the process has completed by calling EsbGetProcessState() until it returns ESB_STATE_DONE.
- To get and set the default calc script, use the functions EsbGetDefaultCalc(), EsbSetDefaultCalc() and EsbSetDefaultCalcFile().

**Return Value**

None.

**Access**

This function requires the caller to have calc privilege (ESB_PRIV_CALC) to the active database.

**Example**

Declare Function EsbDefaultCalc Lib "ESBAPIN" (ByVal hCtx As Long) As Long

Sub ESB_DefaultCalc ()
    Dim sts As Long
    Dim ProcState As ESB_PROCSTATE_T '****************************
        ' Run default calc script
        '****************************
    sts = EsbDefaultCalc (hCtx) '****************************
        ' Check process state till it is done
        '****************************
End Sub
sts = EsbGetProcessState (hCtx, ProcState)
    Do Until ProcState.State = ESB_STATE_DONE
        sts = EsbGetProcessState (hCtx, ProcState)
    Loop
End Sub

See Also
- EsbBeginCalc
- EsbCalc
- EsbGetDefaultCalc
- EsbSetDefaultCalc
- EsbSetDefaultCalcFile

EsbDeleteApplication

Deletes an existing application, either on the client or the server. If the application is running on the server, then it is first stopped.

Syntax
EsbDeleteApplication (hCtx, AppName)
ByVal hCtx As Long
ByVal AppName As String

Parameter Description
hCtx VB API context handle.
AppName Name of application to delete.

Notes
- Deleting a client application removes the local application directory and contents. It also removes all objects stored with the application, including all databases.
- To delete a server application, the connected user must have "create/delete applications" privilege.

Return Value
None.

Access
For a server application, the caller must have application Create/Delete/Edit privilege (ESB_PRIV_APPCREATE).

Example
Declare Function EsbDeleteApplication Lib "ESBAPIN" (ByVal hCtx As Long, ByVal AppName As String) As Long

Sub ESB_DeleteApplication ()
    Dim sts As Long
    Dim AppName As String
    AppName = "Sample"
EsbDeleteApplication

sts = EsbDeleteApplication (hCtx, AppName)
End Sub

See Also

- EsbDeleteDatabase
- EsbDeleteObject

EsbDeleteDatabase

Deletes an existing database from an application, either on the client or the server. If the database is running on the server, then it is first stopped.

Syntax

EsbDeleteDatabase (hCtx, AppName, DbName)
ByVal hCtx As Long
ByVal AppName As String
ByVal DbName As String

Parameter Description

- hCtx VB API context handle.
- AppName Name of an application containing database.
- DbName Name of a database to delete.

Notes

- Deleting a client database removes the local database directory and contents.
- Deleting a server database removes all objects associated with that database.

Return Value

None.

Access

For a server database, the caller must have database Create/Delete/Edit privilege (ESB_PRIV_DBCREATE).

Example

Declare Function EsbDeleteDatabase Lib "ESBAPIN" (ByVal hCtx As Long, ByVal AppName As String, ByVal DbName As String) As Long

Sub ESB_DeleteDatabase ()
  Dim sts As Long
  Dim AppName As String
  Dim DbName As String
  AppName = "Sample"
  DbName = "Basic"
  '****************
End Sub

Visual Basic Main API Function Reference 1257
' Delete database
'****************
sts = EsbDeleteDatabase (hCtx, AppName, DbName)
End Sub

See Also
- EsbDeleteApplication
- EsbDeleteObject

EsbDeleteDrillThruURL

Deletes a drill-through URL, with the given URL name, within the active database outline.

Syntax

Declare Function EsbDeleteDrillThruURL Lib "esbapin" (ByVal hCtx As Long, ByVal URLName As String) As Long

Parameter Description

- hCtx: Visual Basic API context handle
- URLName: Drill-through URL name

Return Value

- If successful, deletes the named drill-through URL in the active database outline.
- If unsuccessful, returns an error code.

Access

- Caller must have database Design privilege (ESB_PRIV_DBDESIGN) for the specified database.
- Caller must have selected the specified database as their active database using EsbSetActive().

Example

Sub ESB_DeleteGLDrillThru()
    Dim URLName As String

    URLName = "VB URL7"
    sts = EsbDeleteDrillThruURL(hCtx, URLName)

    Debug.Print "EsbDeleteDrillThruURL sts: " & sts
End Sub

See also an extended example in “Drill-through Visual Basic API Example” on page 1132.

EsbDeleteFilter

Deletes an existing filter.
Syntax

EsbDeleteFilter (hCtx, AppName, DbName, FltName)
ByVal hCtx As Long
ByVal AppName As String
ByVal DbName As String
ByVal FltName As String

Parameter Description

hCtx VB API context handle.
AppName Application name.
DbName Database name.
FltName Filter name.

Return Value
None.

Access
This function requires the caller to have Database Design privilege (ESB_PRIV_DBDESIGN) for the specified database.

Example

Declare Function EsbDeleteFilter Lib "ESBAPIN" (ByVal hCtx As Long, ByVal AppName As String, ByVal DbName As String, ByVal FltName As String) As Long

Sub ESB_DeleteFilter ()
    Dim sts As Long
    Dim AppName As String
    Dim DbName As String
    Dim FilterName As String
    AppName = "Sample"
    DbName = "Basic"
    FilterName = "Filter"    '**************
    ' Delete Filter
    '**************
    sts = EsbDeleteFilter (hCtx, AppName, DbName, FilterName)
End Sub

See Also

- EsbCopyFilter
- EsbListFilters
- EsbRenameFilter
- EsbSetFilter

EsbDeleteFromGroup

Removes a user from the list of group members.
Syntax

**EsbDeleteFromGroup** (hCtx, GrpName, User)
ByVal hCtx As Long
ByVal GrpName As String
ByVal User As String

Parameter Description

hCtx VB API context handle.

GrpName Group name.

User Name of a user to remove from the group list.

Notes

As well as deleting the specified user from the list of members for the specified group, this function also deletes the group from the user's own list of associated groups.

Return Value

None.

Access

This function requires the caller to have Create/Delete User privilege (ESB_PRIV_USERCREATE) for the logged in server.

Example

Declare Function EsbDeleteFromGroup Lib "ESBAPIN" (ByVal hCtx As Long, ByVal GroupName As String, ByVal User As String) As Long

Sub ESB_DeleteFromGroup ()
    Dim sts As Long
    Dim GroupName As String
    Dim User As String  
    GroupName = "PowerUsers"  
    User = "Jim Smith"
    '***********************
    ' Delete user from group
    '***********************
    sts = EsbDeleteFromGroup (hCtx, GroupName, User)
End Sub

See Also

- EsbAddToGroup
- EsbGetGroupList
- EsbListGroups
- EsbSetGroupList

**EsbDeleteGroup**

Deletes an existing group.
Syntax

**EsbDeleteGroup** (hCtx, GrName)
ByVal hCtx As Long
ByVal GrpName As String

**Parameter Description**

hCtx VB API context handle.

GrpName Name of a group to delete.

**Return Value**

None.

**Access**

This function requires the caller to have Create/Delete User privilege
(ESB_PRIV_USERCREATE) for the logged in server.

**Example**

Declare Function EsbDeleteGroup Lib "ESBAPIN" (ByVal hCtx As Long, ByVal GroupName As String) As Long

Sub ESB_DeleteGroup ()
    Dim sts As Long
    Dim GroupName As String   GroupName = "PowerUsers"   '*************
    ' Delete Group
    '*************
    sts = EsbDeleteGroup (hCtx, GroupName)
End Sub

**See Also**

- EsbCreateGroup
- EsbListGroups
- EsbRenameGroup

**EsbDeleteLocalContext**

Releases a local context previously created by EsbCreateLocalContext().

Syntax

**EsbDeleteLocalContext** (hCtx)
ByVal hCtx As Long

**Parameter Description**

hCtx VB API local context handle.
Notes
This function should only be used for local contexts. For login contexts, use the `EsbLogout()` function.

Return Value
None.

Access
This function requires no special privileges.

Example
Declare Function EsbDeleteLocalContext Lib "ESBAPIN" (ByVal hCtx As Long) As Long
Sub ESB_DeleteLocalContext ()
    Dim sts As Long   '*********************
    ' Delete Local Context
    '*********************
    sts = EsbDeleteLocalContext (hCtx)
End Sub

See Also
- `EsbCreateLocalContext`
- `EsbLogout`
- `EsbTerm`

**EsbDeleteLocationAlias**
Deletes an existing location alias.

Syntax

```
EsbDeleteLocationAlias (hCtx, AliasName)
```

- `hCtx` (ByVal) API context handle
- `AliasName` (ByVal) Location alias

Parameter Description

- **hCtx** 
  API context handle
- **AliasName** 
  Location alias

Return Value
Returns an error if a location alias with the name `AliasName` is not found.

Example

```
Public Sub LocationAliasTest()
    Dim status    As Long
    Dim ListCount As Integer
    Dim Aliases   As Variant
```

1262 Visual Basic Main API Functions
Dim HostNames As Variant
Dim AppNames  As Variant
Dim DbNames   As Variant
Dim UserNames As Variant

status = EsbCreateLocationAlias(hCtx, "blah1", "LocalHost", "Demo", "Basic", _
   "admin", "password")
If (status <> 0) Then
   MsgBox "Create routine Failed"
   Exit Sub
End If

status = EsbCreateLocationAlias(hCtx, "blah2", "LocalHost", "Demo", "Basic", _
   "admin", "password")
If (status <> 0) Then
   MsgBox "Create routine Failed"
   Exit Sub
End If

status = EsbGetLocationAliasList(hCtx, ListCount, Aliases, HostNames, _
   AppNames, DbNames, UserNames)
If (status <> 0) Then
   MsgBox "Get routine Failed"
   Exit Sub
End If

If (ListCount > 0) Then
   ' Retrieve the elements as Aliases(0) to Aliases(ListCount -1)
End If

status = EsbDeleteLocationAlias(hCtx, "blah1")
If (status <> 0) Then
   MsgBox "Delete routine Failed"
   Exit Sub
End If

status = EsbGetLocationAliasList(hCtx, ListCount, Aliases, HostNames, _
   AppNames, DbNames, UserNames)
If (status <> 0) Then
   MsgBox "Get routine Failed"
   Exit Sub
End If

End Sub

See Also
- EsbCreateLocationAlias
- EsbGetLocationAliasList

EsbDeleteLogFile
Deletes an application log file on the server.
Syntax

EsbDeleteLogFile (hCtx, AppName)
ByVal hCtx  As Long
ByVal AppName As String

Parameter Description

hCtx  VB API context handle
AppName  Application name. If AppName is NULL or "" (the empty string), EsbDeleteLogFile() deletes the esbase.log log file.

Return Value

None.

Access

This function requires the caller to have Application Design privilege (ESB_PRIV_APPDESIGN) for the specified application.

Example

Declare Function EsbDeleteLogFile Lib "ESBAPIN" (ByVal hCtx As Long, ByVal AppName As String) As Long
Sub ESB_DeleteLogFile ()
    Dim sts As Long
    Dim AppName As String   AppName = "Sample"   '****************
    ' Delete Log file
    '****************
    sts = EsbDeleteLogFile (hCtx, AppName)
End Sub

See Also

●  EsbGetLogFile

EsbDeleteObject

Deletes an existing object from the server or client object system.

Syntax

EsbDeleteObject (hCtx, ObjType, AppName, DbName, ObjName)
ByVal hCtx  As Long
ByVal ObjType As Long
ByVal AppName As String
ByVal DbName As String
ByVal ObjName As String

Parameter Description

hCtx  VB API context handle. Can be local context handle returned by EsbCreateLocalContext().
ObjType  Object type (must be single type). Refer to Table 15 for a list of possible values.
Parameter Description

AppName Application name.
DbName Database name. If an empty string, uses the application sub-directory.
ObjName Object name to delete.

Notes
- To delete an object, the object must not be locked.
- Outline objects cannot be deleted. Use the EsbDeleteDatabase() function to delete a database, including its associated outline.

Return Value
None.

Access
This function requires the caller to have application or database Design privilege (ESB_PRIV_APPDESIGN or ESB_PRIV_DBDESIGN) for the specified application or database containing the object.

Example
Declare Function EsbDeleteObject Lib "ESBAPIN" (ByVal hCtx As Long, ByVal ObjType As Integer, ByVal AppName As String, ByVal DbName As String, ByVal ObjName As String) As Long

Sub ESB_DeleteObject ()
    Dim sts As Long
    Dim AppName As String
    Dim DbName As String
    Dim ObjName As String
    Dim ObjType As Integer
    AppName = "Sample"
    DbName = "Basic"
    ObjName = "Basic"
    ObjType = ESB_OBJTYPE_RULES '********************
    ' Delete Rules Object
    '*******************
    sts = EsbDeleteObject (hCtx, ObjType, AppName, DbName, ObjName)
End Sub

See Also
- EsbCreateObject
- EsbCopyObject
- EsbListObjects
- EsbRenameObject
- EsbUnlockObject
**EsbDeleteUser**

Deletes an existing user.

**Syntax**

```
EsbDeleteUser (hCtx, userName)
ByVal hCtx As Long
ByVal userName As String
```

**Parameter Description**

- **hCtx**  
  VB API context handle.

- **userName**  
  Name of a user to delete.

**Notes**

The caller may not delete either their own user or the last administrator on the server.

**Return Value**

None.

**Access**

This function requires the caller to have Create/Delete User privilege (ESB_PRIV_USERCREATE) for the logged in server.

**Example**

```
Declare Function EsbDeleteUser Lib "ESBAPIN" (ByVal hCtx As Long, ByVal User As String) As Long

Sub ESB_DeleteUser ()
    Dim sts As long
    Dim User As String   User = "Joseph"   '************
    ' Delete user
    '************
    sts = EsbDeleteUser (hCtx, User)
End Sub
```

**See Also**

- [EsbCreateUser](#)
- [EsbListUsers](#)
- [EsbRenameUser](#)

---

**EsbDeleteVariable**

Deletes a substitution variable.

**Syntax**

```
EsbDeleteVariable (hCtx, pVariable)
ByVal hCtx As Long
    pVariable As ESB_PVARIABLE_T
```

---

1266 Visual Basic Main API Functions
Parameter Description

**hCtx**  
Context handle to the API.

**pVariable**  
The pointer to the structure containing the description of the substitution variable being deleted.

Return Value

If successful, returns zero.

Example

```vbnet
Declare Function EsbDeleteVariable Lib "esbapin" (ByVal hCtx As Long, pVariable As ESB_VARIABLE_T) As Long
Sub Esb_DeleteVariable ()
    Dim sts As Long
    Dim oVariable As ESB_VARIABLE_T

    ' Delete "QuarterName" substitution Variable at the Sample application level
    oVariable.Server = "Localhost"
    oVariable.AppName = "Sample"
    ' ** Note that DbName has been left empty
    oVariable.VarName = "QuarterName"
    oVariable.VarValue = "Qtr1"

    sts = EsbDeleteVariable(hCtx, oVariable)
End Sub
```

See Also

- “ESB_VARIABLE_T” on page 1196
- EsbCreateVariable
- EsbGetVariable
- EsbListVariables

### EsbDisplayAlias

Displays the contents of an alias table in the active database.

Syntax

```vbnet
EsbDisplayAlias (hCtx, AltName, pItems)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>VB API context handle.</td>
</tr>
<tr>
<td>AltName</td>
<td>Name of alias table.</td>
</tr>
<tr>
<td>pItems</td>
<td>Address of variable to receive Items of aliases.</td>
</tr>
</tbody>
</table>

Visual Basic Main API Function Reference 1267
Notes

Windows only: The information returned by this command can exceed the ability of Windows to allocate memory. The Windows memory limit is 64 K.

Return Value

This function returns the number of alias names in the table and generates an array of MBRALT structures accessible via EsbGetNextItem().

Access

This function requires the caller to have access to the database, and to have selected it as their active database using EsbSetActive().

Example

Declare Function EsbDisplayAlias Lib "ESBAPIN" (ByVal hCtx As Long, ByVal Name As String, Items As Integer) As Long

Sub ESB_DisplayAlias ()
      Dim pItems As Integer
      Dim Name As String
      Dim MbrAlt As ESB_MBRALT_T
      Dim sts As Long
      Name = "Default"  '**************
      ' Display Alias
      '**************
      sts = EsbDisplayAlias (hCtx, Name, pItems)  '***************************
      ' Get next Member/Alias Name
      ' combination from the list
      '***************************
      sts = EsbGetNextItem (hCtx, ESB_MBRALT_TYPE, MbrAlt)
      Next
End Sub

See Also

- EsbListAliases
- EsbGetNextItem

EsbEndCalc

Marks the end of a calc script being sent to the active database. This function must be called after sending the calc script (using EsbSendString()).

Syntax

EsbEndCalc (hCtx)
ByVal hCtx As Long

Parameter Description

hCtx VB API context handle.
Notes

- This function must be preceded by a call to `EsbBeginCalc()`, and at least one call to `EsbSendString()`.
- If the calls to `EsbBeginCalc()`, `EsbSendString()`, and `EsbEndCalc` succeed, the caller must call `EsbGetProcessState()` until it returns ESB_STATE_DONE to determine when the process has completed.

Return Value

None.

Access

This function requires the caller to have calc privilege (ESB_PRIV_CALC) to the active database.

Example

Declare Function EsbEndCalc Lib "ESBAPIN" (ByVal hCtx As Long) As Long

See the example for `EsbBeginCalc`.

See Also

- `EsbBeginCalc`
- `EsbCalc`
- `EsbSendString`

---

**EsbEndDataload**

Marks the end of an update specification being sent to the active database, and must be called after sending the update specification using `EsbSendString()`.

Syntax

Declare Function EsbEndDataload Lib "esbapin" (ByVal hCtx As Long, ByVal ErrorName As String) As Long

Parameter Description

- **hCtx**  
  API context handle.
- **ErrorName**  
  The name of the text file containing the list of errors. Possible errors (and error strings) in the text file are:
  - ESS_MBRERR_UNKNOWN (Unknown member [membername] in dataload, [number] records returned.)
  - ESS_MBRERR_DBACCESS (You have insufficient access privilege to perform a lock on this database.)
  - ESS_MBRERR_BADDATA (Invalid member [membername] in data column.)
  - ESS_MBRERR_DUPLICATE (Duplicate members from the same dimension on data record, [number] records completed.)
  - AD_MSGDL_ERRORLOAD (Unable to do dataload at Item/Record [number].)
Notes

- `EsbEndDataload()` must be preceded by a call to `EsbBeginDataload()`, and at least one call to `EsbSendString()`.
- `EsbEndDataload()` returns a text file containing the list of errors.

Return Value

Returns zero, if successful. Otherwise, returns an error code, as follows:

- If `abortOnError` is TRUE:
  - The error code for the first error condition is returned.
  - The error list is NULL.
- If `abortOnError` is FALSE:
  - An error file is returned, if the server can process the data and can continue.
  - Otherwise, in exceptional circumstances, the error code explaining why the server cannot continue is returned. For example:
    - AD_MSGDL_COLS (too many data values in a record)
    - AD_MSGDL_MISDIM (data value encountered before all dimensions selected)

Access

This function requires the caller to have write privilege (ESS_PRIV_WRITE) to the active database.

**EsbEndReport**

Marks the end of a report specification being sent to the active database. This function must be called after sending the report specification (using `EsbSendString()`) and before reading any returned data (using `EsbGetString()`).

**Syntax**

```
EsbEndReport (hCtx)
ByVal hCtx As Long
```

**Parameter Description**

- **hCtx** VB API context handle.

**Notes**

- This function must be preceded by a call to `EsbBeginReport()`, and at least one call to `EsbSendString()`.
- If the output flag is TRUE for the call to `EsbBeginReport()` that begins the report sequence, the call to `EsbEndReport()` must be followed by repeated calls to `EsbGetString()` until an empty string is returned.
**Return Value**

None.

**Access**

This function requires callers to have read privilege (ESB_PRIV_READ) to one or more members in the active database.

**Example**

Declares Function EsbEndReport Lib "ESBAPIN" (ByVal hCtx As Long) As Long

See the example for EsbBeginReport.

**See Also**

- EsbBeginReport
- EsbGetString
- EsbSendString

---

**EsbEndUpdate**

Marks the end of an update specification being sent to the active database. This function must be called after sending the update specification (using EsbSendString()).

**Syntax**

```
EsbEndUpdate (hCtx)
```

**Parameter Description**

- **hCtx** VB API context handle.

**Notes**

This function must be preceded by a call to EsbBeginUpdate(), and at least one call to EsbSendString().

**Return Value**

None.

**Access**

This function requires callers to have write privilege (ESB_PRIV_WRITE) to the active database.

**Example**

Declares Function EsbEndUpdate Lib "ESBAPIN" (By Val hCtx As Long) As Long

See the example for EsbBeginUpdate.

**See Also**

- EsbBeginUpdate

---

Visual Basic Main API Function Reference 1271
EsbExport

Exports a database to an ASCII file.

Syntax

EsbExport (hCtx, AppName, DbName, FilePath, Level, isColumns)
ByVal hCtx As Long
ByVal AppName As String
ByVal DbName As String
ByVal FilePath As String
ByVal Level As Integer
ByVal isColumns As Integer

Parameter Description

hCtx VB API context handle.
AppName Name of application to archive.
DbName Name of database to archive.
FilePath Full path name of server file to contain archive information.
Level Controls level of data to export. Should be one of:
  - ESB_DATA_ALL - export all levels of data
  - ESB_DATA_LEVEL0 - only export all data from level zero blocks
  - ESB_DATA_INPUT - only export data from input level blocks
isColumns Controls output of data blocks in column format.

Notes

If this function succeeds, the export will continue on the server as an asynchronous process after
the return from this call. The caller should check at regular intervals to see if the process has
completed by calling EsbGetProcessState() until it returns ESB_STATE_DONE.

Return Value

None.

Access

This function requires callers to have access to the database, and to have selected it as their active
database using EsbSetActive().

Example

Declare Function EsbExport Lib "ESBAPIN" (ByVal hCtx As Long, ByVal AppName As String,
ByVal DbName As String, ByVal FilePath As String, ByVal Level As Integer, ByVal Columns
As Integer) As Long
Sub ESB_Export()
    Dim sts As Long
    Dim AppName As String
    Dim DbName As String
    Dim PathName As String
    Dim Level As Integer
    Dim Columns As Integer
    Dim ProcState As ESB_PROCSTATE_T
    AppName = "Sample"
    DbName = "Basic"
    PathName = "c:\essbase\main.txt"
    Level = ESB_DATA_INPUT
    Columns = ESB_YES
    '****************************
    ' Export input level data only
    '****************************
    sts = EsbExport(hCtx, AppName, DbName,
    PathName, Level, Columns)
    '************************************
    ' Check process state till it is done
    '************************************
    sts = EsbGetProcessState(hCtx, ProcState)
    Do Until ProcState.State = ESB_STATE_DONE
        sts = EsbGetProcessState(hCtx, ProcState)
    Loop
End Sub

See Also

● EsbImport

EsbGetActive

Gets the names of the caller’s current active application and database.

Syntax

**EsbGetActive** (hCtx, AppName, szApp, DbName, szDb, pAccess)

ByVal hCtx As Long
ByVal AppName As String
ByVal szApp As Integer
ByVal DbName As String
ByVal szDb As Integer
ByVal pAccess As Integer

Parameter Description

<table>
<thead>
<tr>
<th>hCtx</th>
<th>VB API context handle.</th>
</tr>
</thead>
<tbody>
<tr>
<td>AppName</td>
<td>Buffer to receive an application name string.</td>
</tr>
<tr>
<td>szApp</td>
<td>Size of an application name string buffer.</td>
</tr>
<tr>
<td>DbName</td>
<td>Buffer to receive a database name string.</td>
</tr>
<tr>
<td>szDb</td>
<td>Size of the database name string buffer.</td>
</tr>
</tbody>
</table>
### Parameter Description

**pAccess**  
Address of variable to receive the user’s access level to the selected database.

### Notes

If application/database name length is greater than the size of the buffer, the name is truncated.

### Return Value

If successful, returns the user’s selected active application and database in `AppName` and `DbName`.

### Access

This function requires no special privileges.

### Example

```vbnet
Declare Function EsbGetActive Lib "ESBAPIN" (ByVal hCtx As Long, ByVal AppName As String, ByVal szApp As Integer, ByVal DbName As String, ByVal szDb As Integer, Access As Integer) As Long

Sub ESB_GetActive ()
    Dim AppName As String * ESB_APPNAMELEN
    Dim DbName As String * ESB_DBNAMELEN
    Dim sts As Long
    Dim szApp As Integer
    Dim szDb As Integer
    Dim pAccess As Integer

    szApp = ESB_APPNAMELEN
    szDb = ESB_DBNAMELEN
    '**********************************
    ' Get active Application & Database
    '**********************************
    sts = EsbGetActive (hCtx, AppName, szApp, DbName, szDb, Access)
End Sub
```

### See Also

- *EsbClearActive*
- *EsbSetActive*

### EsbGetAlias

Gets the active alias table name from the active database for a user.

### Syntax

```vbnet
EsbGetAlias (hCtx, AltName, szName)
ByVal hCtx As Long
ByVal AltName As String
ByVal szName As Integer
```

### Parameter Description

**hCtx**  
VB API context handle.
Parameter Description

AltName Buffer to receive a name of active alias an table.

szName Size of the buffer to receive a name of an active alias table.

Notes

If Alias name length is greater that the size of the buffer, the name is truncated.

Return Value

If successful, returns the name of the active alias table in AliasName.

Access

This function requires the caller to have access to the database, and to have selected it as their active database using EsbSetActive().

Example

Declare Function EsbGetAlias Lib "ESBAPIN" (ByVal hCtx As Long, ByVal Name As String, ByVal szName As Integer) As Long

Sub ESB_GetAlias ()
    Dim sts As Long
    Const szName = 80
    Dim pName As String * szName   '**********
    ' Get Alias
    '**********
    sts = EsbGetAlias (hCtx, pName, szName)
End Sub

See Also

- EsbListAliases
- EsbSetAlias
- EsbSetActive

EsbGetAPIVersion

Returns the current version of the Essbase API.

Syntax

EsbGetAPIVersion (lVersion)
lVersion As Long
Parameter Description

Version Version number of API. Hex value, in Visual Basic notation, with the following format:
   &H00000000
   For example, &H00040000 represents Release 4.0, and &H00030002 represents Release 3.2.
   ● First 4 numbers from right (low order word): release number between versions
   ● Remaining numbers (high order word): version number

Notes

You can use this function to check the API version when your program requires a particular version.

Example

Declare Function EsbGetAPIVersion Lib "ESBAPIN" (lVersion As Long) As Long
Sub ESB_GetAPIVersion()
   Dim sts As Long
   Dim Version As Long  '***************
   'Get API Version
   '***************
   sts = EsbGetAPIVersion(Version)
End Sub

See Also

● EsbGetObjectInfo

EsbGetApplicationAccess

Gets a list of user application access structures, which contain information about user access to applications.

Syntax

EsbGetApplicationAccess (hCtx, User,AppName, pItems)
   ByVal hCtx  As Long
   ByVal User  As String
   ByValAppName As String
   pItems  As Integer

Parameter Description

hCtx   VB API context handle.
User   User name. If an empty string, lists all users for the specified application.
AppName Application name. If an empty string, lists all applications for the specified user.
pItems Address of variable to receive Items of user application structures.
Notes

- If User is an empty string, all users are listed for the specified application. IfAppName is an empty string, all applications are listed for the specified user. However, User and AppName cannot both be an empty string.
- The Access field of the user application structure is used to represent the user’s granted access to the application, whereas the MaxAccess field represents the user’s highest access from all sources (e.g. via groups or default application access etc.).

Return Value

If successful, returns a Items of users/applications in pItems, and generates a list of user application structures accessible via EsbGetNextItem()

Access

This function requires the caller to have application Design privilege (ESB_PRIV_APPDESIGN) for the specified application, unless they are getting their own application access information.

Example

Declare Function EsbGetApplicationAccess Lib "ESBAPIN" (ByVal hCtx As Long, ByVal User As String, ByVal AppName As String, Items As Integer) As Long

Sub ESB_GetApplicationAccess ()
    Dim Items As Integer
    Dim AppName As String
    Dim User As String
    Dim UserApp As ESB_USERAPP_T
    Dim sts As Long   AppName = "Demo"
    User = "Joseph"   '******************************
    ' Get Application Access
    '******************************
    sts = EsbGetApplicationAccess (hCtx,
    User, AppName, Items)   For n = 1 To Items   '******************************
    ' Get next User Application Access
    ' structure from the list
    '******************************
    sts = EsbGetNextItem (hCtx,
    ESB_USERAPP_TYPE, UserApp)
    Next
End Sub

See Also

- EsbGetDatabaseAccess
- EsbListUsers
- EsbSetApplicationAccess
- EsbSetUser
- EsbGetNextItem
EsbGetApplicationInfo

Gets an application's information structure, which contains non user-configurable parameters for the application.

**Syntax**

```plaintext
EsbGetApplicationInfo (hCtx, AppName, pAppInfo, pItems)
ByVal hCtx As Long
ByVal AppName As String
ByVal pAppInfo As ESB_APPINFO_T
pItems As Integer
```

**Parameter Description**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>VB API context handle (logged in).</td>
</tr>
<tr>
<td>AppName</td>
<td>Application name. Required; cannot by NULL.</td>
</tr>
<tr>
<td>pAppInfo</td>
<td>Buffer to receive an application info structure.</td>
</tr>
<tr>
<td>pItems</td>
<td>Address of variable to receive Items of returned databases.</td>
</tr>
</tbody>
</table>

**Notes**

This function can only be called for applications on the server.

**Return Value**

If successful, this function returns an application info structure in `pAppInfo` and a number of databases in `pItems` and generates a list of database name strings that is accessible via `EsbGetNextItem()`.

**Access**

This function requires the caller to have access to the specified application.

**Example**

```plaintext
Declare Function EsbGetApplicationInfo Lib "ESBAPIN" (ByVal hCtx As Long, ByVal AppName As String, AppInfo As ESB_APPINFO_T, Items As Integer) As Long

Sub ESB_GetApplicationInfo ()
    Dim Items As Integer
    Dim AppName As String
    Dim DbName As String * ESB_DBNAMELEN
    Dim AppInfo As ESB_APPINFO_T
    Dim sts As Long   AppName = "Sample"   '*******************************
    ' Get Application info structure
    '*******************************
    sts = EsbGetApplicationInfo (hCtx, AppName, AppInfo, Items)   For n = 1 To Items       '****************************
    ' Get next Database name string
    ' from the list
    '****************************
    sts = EsbGetNextItem (hCtx, ESB_DBNAME_TYPE, ByVal DbName)
```
EsbGetApplicationInfoEx

Retrieves information from multiple databases, including non user-configurable parameters for the Application.

Syntax

EsbGetApplicationInfoEx (hCtx,AppName, pItems)
ByVal hCtx As Long
ByVal AppName As String
   pItems As Integer

Parameter Description

hCtx          VB API context handle (logged in).
AppName       Application name.
pItems        Address of variable to receive Items of returned databases.

Notes

- This function can only be called for applications on the server.
- The caller of this function must call EsbGetNextItem with the ESB_APPINFOEX_TYPE parameter, which returns the structure ESB_APPINFOEX_T. ESB_APPINFOEX_T and ESB_APPINFO_T are the same, except that ESB_APPINFOEX_ does not include database information.

Return Value

If successful, this function returns an array of application information structures in ppAppInfo.

Access

This function requires the caller to have access to the specified application.

Example

Declare Function EsbGetApplicationInfoEx Lib "ESBAPIN" (ByVal hCtx As Long, ByVal
AppName As String, pItems As Integer) As Long

Sub ESB_GetApplicationInfoEx()
   Dim sts As Long
   Dim AppName As String
   Dim Items As Integer
Dim AppInfoEx As ESB_APPINFOEX_T
AppName = ""
'***********************
'Get application info Ex
'***********************
sts = EsbGetApplicationInfoEx(hCtx, AppName,
Items)
For n = 1 To Items
    '***********************************
    ' Get next Application Info item
    ' from the list
    '***********************************
    sts = EsbGetNextItem(hCtx, ESB_APPINFOEX_TYPE,
AppInfoEx)
    Next
End Sub

See Also

- EsbGetApplicationInfo
- EsbGetApplicationState
- EsbGetDatabaseInfo
- EsbGetNextItem

**EsbGetApplicationState**

Gets an Application’s state structure, which contains user-configurable parameters for the Application.

**Syntax**

```vbnet
EsbGetApplicationState (hCtx, AppName, pAppState)
```

*ByVal hCtx As Long*
*ByVal AppName As String*
*ByVal pAppState As ESB_PAPPSTATE_T*

**Parameter Description**

- `hCtx` VB API context handle.
- `AppName` Application name.
- `pAppState` Buffer to receive an application state structure.

**Notes**

This function cannot be called for local applications; it can only be called for applications on the server.

**Return Value**

If successful, this function returns an application state structure in `pAppState`.

**Access**

This function requires the caller to have access to the specified application.
Example

Declare Function EsbGetApplicationState Lib "ESBAPIN" (ByVal hCtx As Long, ByVal AppName As String, AppState As ESB_APPSTATE_T As Long)

Sub ESB_GetApplicationState ()
    Dim sts As Long
    Dim AppName As String
    Dim AppState As ESB_APPSTATE_T
    AppName = "Sample"
    '********************************
    ' Get Application State structure
    '********************************
    sts = EsbGetApplicationState (hCtx, AppName, AppState)
End Sub

See Also

- EsbGetApplicationInfo
- EsbGetDatabaseState
- EsbSetApplicationState

EsbGetAssociatedAttributesInfo

Returns any attribute information associated with a given base member.

Syntax

EsbGetAssociatedAttributesInfo (hCtx, MbrName, AttrDimName, Count)

ByVal hCtx As Long
ByVal MbrName As String
ByVal AttrDimName As String
Count As Long

Parameter Description

hCtx          Context handle
MbrName       Base member name
AttrDimName   (Optional) attribute dimension name
Count         Number of attribute members returned

Notes

- This function retrieves more information for attribute members than EsbQueryDatabaseMembers.
- Set AttrDimName to NULL to return all attribute members that are associated with the base member.
- Optionally, provide an attribute dimension name to retrieve information only about the member of that dimension which is associated with the base member.
After you call `EsbGetAssociatedAttributesInfo()`, call `EsbGetNextItem()`, using “ESB_ATTRIBUTEINFO_T” on page 1168, to retrieve the attribute information structure(s) that you want.

There are two situations where attribute information returned from this function might be invalid:

1. In the Visual Basic API, the attribute data type of a given attribute dimension is derived from the data type of the attribute dimension's name. Because of this, attribute values might not be valid for attribute dimensions. Applications should ignore the value of the Attribute field in the returned ESB_ATTRIBUTEINFO_T structure if the name passed to `EsbGetAssociatedAttributesInfo()` is that of an attribute dimension. Test the MbrInfo.MbrName and MbrInfo.DimName fields to see if they are equal. If equal, they refer to a base dimension, and the attributes information should be ignored.

2. Date attributes contain time information (a time stamp) that is automatically processed (has date math performed upon it) by Visual Basic. This could lead to invalid values for date attributes (depending upon the time zone specified in a given client machine). To avoid this automatic processing, use the attribute name as opposed to the attribute value when displaying the date attribute information.

Return Value

Returns sts = 0 when successful, otherwise returns an error number.

Access

This function requires no special privileges.

Example

```vba
Sub ESB_GetAssociatedAttributesInfo()
    ' NOTE: 'Out' is a sub to print the output within quotes to a listbox or text box.
    Dim hCtx as long
    Dim sts as long
    Dim MbrName As String
    Dim AttrDimName As String
    Dim Count As Long
    Dim attribinfo As ESB_ATTRIBUTEINFO_T
    Dim index As Integer
    Dim tempstring As String

    MbrName = InputBox("Base member name", "Base Member Name")
    AttrDimName = InputBox("Attribute Dimension Name (Optional)", "Attribute Dimension Name")

    sts = EsbGetAssociatedAttributesInfo(hCtx, MbrName, AttrDimName, Count)
    If sts <> 0 Then
        MsgBox "Error in ESB_GetAssociatedAttributesInfo: " & sts: Exit Sub
    Else
        tempstring = "...count = " & Count & "..."
        out (tempstring)
        Out "Associated Attr info for " & "[" & MbrName & "]""```
Out "------------------------------------"

For index = 1 To Count
    sts = EsbGetNextItem(hCtx, ESB_ATTRIBUTEINFO_TYPE, Attribinfo)
    Out "Dim Name: " & Attribinfo.DimName
    Out "Mbr Name: " & Attribinfo.MbrName
    ' NOTE: use of select case statement to discern (and act upon) type of attribute returned
    Select Case VarType(Attribinfo.Attribute)
        Case vbDouble
            Out "Data Type : Numeric(Double)"
            Out "Data Value : " & Attribinfo.Attribute
            Out ""
        Case vbBoolean
            Out "Data Type : Boolean"
            Out "Data Value : " & Attribinfo.Attribute
            Out ""
        Case vbDate
            Out "Data Type : Date"
            ' Suggested way to get Date Attribute value for display
            Out "Data Value : " & Attribinfo.DimName
            Out ""
        Case vbString
            Out "Data Type : String"
            Out "Data Value : " & Attribinfo.Attribute
            Out ""
        End Select
    Out ""
Next index
End If
End Sub

See Also

- EsbCheckAttributes
- EsbGetAttributeInfo
- EsbGetAttributeSpecifications
- EsbOtlAssociateAttributeDimension
- EsbOtlAssociateAttributeMember
- EsbOtlDisassociateAttributeDimension
- EsbOtlDisassociateAttributeMember
- EsbOtlFindAttributeMembers
- EsbOtlGetAssociatedAttributes
- EsbOtlGetAttributeInfo
- EsbOtlGetAttributeSpecifications
- EsbOtlQueryAttributes
- EsbOtlSetAttributeSpecifications

EsbGetAttributeInfo

Returns attribute information for a given attribute member or dimension.
Syntax

EsbGetAttributeInfo (hCtx, AttrName, AttrInfo)
ByVal hCtx As Long
ByVal AttrName As String
    AttrInfo As ESB_ATTRIBUTEINFO_T

Parameter Description

hCtx    Context handle
AttrName Name of the attribute member or dimension
AttrInfo Attribute information

Notes

If a base member or dimension is passed in, information will be returned, but there is no attribute specific information to be displayed. Also, in the circumstance of a base dimension being passed in, the dimension and member name fields of the structure will hold identical values.

Return Value

Returns sts = 0 when successful and populates the ESB_ATTRIBUTEINFO_T structure. Otherwise returns an error number.

Access

This function requires no special privileges.

Example

Sub ESB_GetAttributeInfo()
    ' NOTE: 'Out' is a sub to print the output within quotes to a listbox or text box.
    Dim hCtx as long
    Dim sts as long
    Dim MbrName As String
    Dim OutAttrInfo As ESB_ATTRIBUTEINFO_T

    MbrName = InputBox("Member Name")
    sts = EsbGetAttributeInfo(hCtx, MbrName, OutAttrInfo)
    If sts = 0 Then
        Out "ESB_OtlGetAttributeInfo passed" & sts
        Out "MbrName : " & OutAttrInfo.MbrName
        Out "DimName : " & OutAttrInfo.DimName
        Out "Attribute : " & OutAttrInfo.Attribute
    Else
        Out "ESB_OtlGetAttributeInfo failed" & sts: Exit Sub
    End If
End Sub

See Also

- EsbCheckAttributes
- EsbGetAssociatedAttributesInfo
- EsbGetAttributeSpecifications
- EsbOtlAssociateAttributeDimension
EsbGetAttributeSpecifications

Retrieves attribute specifications for the outline.

Syntax

EsbGetAttributeSpecifications (hCtx, AttrSpecs)

ByVal hCtx As Long
AttrSpecs As As ESB_ATTRSPECS_T

Parameter Description

hCtx     Context handle
AttrSpecs Attribute specifications

Notes

- Set attribute specifications for the outline using EsbSetAttributeSpecifications().
- Attribute specifications are used to do the following:
  - Generate a long name
  - Indicate the format of a datetime attribute
  - Indicate a numeric attribute's bucketing type
  - Provide the name of the attribute calculations dimension and the names for the values used with it

Access

This function requires no special privileges.

Example

Sub ESB_GetAttributeSpecifications()
  ' NOTE: 'Out' is a sub to print the output within quotes to a listbox or text box.
  Dim OutAttrSpecs As ESB_ATTRSPECS_T
  Dim sts as long
  Dim hCtx as long
  Dim test As String

  sts = EsbGetAttributeSpecifications(hCtx, OutAttrSpecs)

  If sts <> 0 Then Out "ESB_GetAttributeSpecifications failed" & sts: Exit Sub
Out "ESB_GetAttributeSpecifications passed: " & sts
Out "DefaultTrueString : " & OutAttrSpecs.DefaultTrueString
Out "DefaultFalseString : " & OutAttrSpecs.DefaultFalseString
Out "DefaultAttrCalcDimName : " & OutAttrSpecs.DefaultAttrCalcDimName
Out "DefaultSumMbrName : " & OutAttrSpecs.DefaultSumMbrName
Out "DefaultCountMbrName : " & OutAttrSpecs.DefaultCountMbrName
Out "DefaultAverageMbrName : " & OutAttrSpecs.DefaultAverageMbrName
Out "DefaultMinMbrName : " & OutAttrSpecs.DefaultMinMbrName
Out "DefaultMaxMbrName : " & OutAttrSpecs.DefaultMaxMbrName

test = OutAttrSpecs.GenNameBy
Select Case test
    Case ESB_GENNAMEBY_PREFIX
        Out "GenNameBy : ESB_GENNAMEBY_PREFIX"
    Case ESB_GENNAMEBY_SUFIX
        Out "GenNameBy : ESB_GENNAMEBY_SUFIX"
    Case Else
        Out "GenNameBy : invalid"
End Select

test = OutAttrSpecs.UseNameOf
Select Case test
    Case ESB USENAMEOF_NONE
        Out "UseNameOf : ESB_USENAMEOF_NONE"
    Case ESB_USENAMEOF_PARENT
        Out "UseNameOf : ESB_USENAMEOF_PARENT"
    Case ESB_USENAMEOF_GRANDPARENTANDPARENT
        Out "UseNameOf : ESB_USENAMEOF_GRANDPARENTANDPARENT"
    Case ESB_USENAMEOF_ALLANCESTORS
        Out "UseNameOf : ESB_USENAMEOF_ALLANCESTORS"
    Case ESB_USENAMEOF_DIMENSION
        Out "UseNameOf : ESB_USENAMEOF_DIMENSION"
    Case Else
        Out "UseNameOf : invalid"
End Select

test = OutAttrSpecs.Delimiter
Select Case test
    Case ESB_DELIMITER_UNDERSCORE
        Out "Delimiter : ESB_DELIMITER_UNDERSCORE"
    Case ESB_DELIMITER_PIPE
        Out "Delimiter : ESB_DELIMITER_PIPE"
    Case ESB_DELIMITER_CARET
        Out "Delimiter : ESB_DELIMITER_CARET"
    Case Else
        Out "Delimiter : invalid"
End Select

test = OutAttrSpecs.DateFormat
Select Case test
    Case ESB_DATEFORMAT_MMDDYYYY
        Out "DateFormat : ESB_DATEFORMAT_MMDDYYYY"
    Case ESB_DATEFORMAT_DDMMYYYY
        Out "DateFormat : ESB_DATEFORMAT_DDMMYYYY"
    Case Else
        Out "Delimiter : invalid"
End Select
test = OutAttrSpecs.BucketingType  
Select Case test  
  Case ESB_UPPERBOUNDINCLUSIVE  
    Out "BucketingType : ESB_UPPERBOUNDINCLUSIVE"  
  Case ESB_LOWERBOUNDINCLUSIVE  
    Out "BucketingType : ESB_LOWERBOUNDINCLUSIVE"  
  Case ESB_UPPERBOUNDNONINCLUSIVE  
    Out "BucketingType : ESB_UPPERBOUNDNONINCLUSIVE"  
  Case ESB_LOWERBOUNDNONINCLUSIVE  
    Out "BucketingType : ESB_LOWERBOUNDNONINCLUSIVE"  
  Case Else  
    Out "BucketingType : invalid"  
End Select  
End Sub  

See Also  
- EsbCheckAttributes  
- EsbGetAssociatedAttributesInfo  
- EsbGetAttributeInfo  
- EsbOtlAssociateAttributeDimension  
- EsbOtlAssociateAttributeMember  
- EsbOtlDisassociateAttributeDimension  
- EsbOtlDisassociateAttributeMember  
- EsbOtlFindAttributeMembers  
- EsbOtlGetAssociatedAttributes  
- EsbOtlGetAttributeInfo  
- EsbOtlGetAttributeSpecifications  
- EsbOtlQueryAttributes  
- EsbOtlSetAttributeSpecifications  

EsbGetCalcList  
Gets the list of calc script objects that are accessible to a user. The programmer needs to use EsbGetNextItem() to access the list of available scripts.  

Syntax  
EsbGetCalcList (hCtx, UserName, AppName, DbName, isAllCalcs, pItems)  
ByVal hCtx As Long  
ByVal UserName As String  
ByVal AppName As String  
ByVal DbName As String  
  isAllCalcs As Integer  
  pItems As Integer  

Parameter Description  

hCtx VB API context handle.  
UserName User name.
### Parameter Description

**AppName**  Application name.

**DbName**  Database name. If an empty string, uses Application sub-directory

**isAllCalcs**  Integer that contains the AllowAllCalcs flag. If AllowAllCalcs is set to ESB_FALSE, the user can access all calc script objects. Otherwise, the user can only access those script objects specified in the CalcList argument.

**pItems**  Integer that contains the count of available calc script objects.

### Notes
- In order to access any calc script objects, the specified user must have at least calculate access to the appropriate database.
- If the value returned in `pAllCalcs` is TRUE, then the value returned in `pItems` is zero.

### Return Value
If successful, returns `sts=0` and returns the user’s AllowAllCalcs setting in `pAllCalcs`. If `isAllCalcs` is equal to ESB_FALSE, `pItems` contains the count of the available calc script objects. Access the list of calc script object names with `EsbGetNextItem()`.

If `isAllCalcs` is equal to ESB_TRUE, then `pItems` will return 0 and the programmer will need to call a combination for `EsbListObjects()` (using type ESB_OBJTYPE_CALCSCRIPT) and `EsbGetObjectInfo()` for each returned object.

### Access
This function requires callers to have Database Design privilege (ESB_PRIV_DBDESIGN) for the specified database, unless they are getting their own calc list.

### Example

Declare Function EsbGetCalcList Lib "ESBAPIN" (ByVal hCtx As Long, ByVal User As String, ByVal AppName As String, ByVal DbName As String, AllCalcs As Integer, Items As Integer) As Long

Sub ESB_GetCalcList()
    Dim Items As Integer
    Dim AppName As String
    Dim DbName As String
    Dim User As String
    Dim AllCalcs As Integer
    Dim ObjName As String * ESB_OBJNAMELEN
    Dim sts As Long
    Dim ObjType As Long
    Dim ObjectInfo As ESB_OBJINFO_T
    ObjType = ESB_OBJTYPE_CALCSCRIPT    AppName = "Sample"
    DbName = "Basic"
    User = "test_user" ' Has 'calculate' access to Sample->Basic ' If user passed in has access to everything,
        ' then Items will ALWAYS be set to '0'!
        ' In that case, use EsbListObjects()
        ' (of type ESB_OBJTYPE_CALCSCRIPT, and
        ' then EsbGetObjectInfo())!
    sts = EsbGetCalcList(hCtx, User, AppName, DbName, AllCalcs, Items)
If AllCalcs = ESB_NO Then
    frmMain.lstInfo.AddItem "Number of calc script items returned: " & Items
    frmMain.lstInfo.AddItem "--------------------------------------------------"
    For n = 1 To Items
        sts = EsbGetNextItem(hCtx, ESB_OBJNAME_TYPE, ByVal ObjName)
        If sts <> 0 Then MsgBox "Failure in EsbGetNextItem(): " & sts: Exit Sub
        sts = EsbGetObjectInfo(hCtx, ObjType, AppName, DbName, ObjName, ObjectInfo)
        If sts <> 0 Then MsgBox "Failure in EsbGetObjectInfo(): " & sts: Exit Sub
        frmMain.lstInfo.AddItem ObjectInfo.Name
        frmMain.lstInfo.AddItem ObjectInfo.Type
        frmMain.lstInfo.AddItem "----------"
    Next
Else
    frmMain.lstInfo.AddItem "You need to call EsbListObjects of type ESB_OBJTYPE_CALCSCRIPT"
End If
End Sub

See Also
- EsbListObjects
- EsbListUsers
- EsbSetCalcList
- EsbGetNextItem

**EsbGetCellDrillThruReports**

Gets the drill-through reports associated with a data cell as a list of URL XMLs, given the cell's member combination.

**Syntax**

Declare Function EsbGetCellDrillThruReports Lib "esbapin" (ByVal hCtx As Long, ByRef pMbrs() As String, ByRef ppURLXMLLen As Variant, ByRef ppURLXML As Variant) As Long

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>Visual Basic API context handle</td>
</tr>
<tr>
<td>pMbrs</td>
<td>List of member names (or Aliases)</td>
</tr>
<tr>
<td>ppURLXMLLen</td>
<td>Returns length of URL XML generated</td>
</tr>
<tr>
<td>ppURLXML</td>
<td>Returns pointers to the URL XML byte stream</td>
</tr>
</tbody>
</table>

**Notes**

The application database needs to be set to Active for this call. This function needs to be extended to support any additional information needed by the clients.

**Return Value**

- If successful, gets the list of URL XMLs.
- If unsuccessful, returns an error code.
**Access**

- Caller must have database Read privilege (ESB_PRIV_READ) for the specified database.
- Caller must have selected the specified database as their active database using EsbSetActive().

**Example**

```visualbasic
Sub ESB_GetCellDrillThruReports()
    Dim intX           As Integer
    Dim mbrs(0 To 4)   As String
    Dim pURLXMLLens    As Variant
    Dim pURLXMLs       As Variant

    mbrs(0) = "sales"
    mbrs(1) = "jan"
    mbrs(2) = "New York"
    mbrs(3) = "actual"
    mbrs(4) = "100-10"

    sts = EsbGetCellDrillThruReports(hCtx, mbrs, pURLXMLLens, pURLXMLs)
    If sts = 0 Then
        Debug.Print "EsbGetCellDrillThruReports sts: " & sts
        For intX = LBound(pURLXMLLens) To UBound(pURLXMLLens)
            Debug.Print "URL XML: " & intX
            Debug.Print "URL XML Len: " & pURLXMLLens(intX)
            Debug.Print "URL XML String: " & pURLXMLs(intX)
        Next
    End If

    mbrs(0) = "profit"
    sts = EsbGetCellDrillThruReports(hCtx, mbrs, pURLXMLLens, pURLXMLs)
    If sts = 0 Then
        Debug.Print "EsbGetCellDrillThruReports sts: " & sts
        For intX = LBound(pURLXMLLens) To UBound(pURLXMLLens)
            Debug.Print "URL XML: " & intX
            Debug.Print "URL XML Len: " & pURLXMLLens(intX)
            Debug.Print "URL XML String: " & pURLXMLs(intX)
        Next
    End If
End Sub
```

See also an extended example in “Drill-through Visual Basic API Example” on page 1132.

**EsbGetCurrencyRateInfo**

Gets a list of structures containing rate information for all members of the tagged currency partition dimension in the active database outline.
Syntax

EsbGetCurrencyRateInfo (hCtx, pItems)
ByVal hCtx As Long
    pItems As Integer

Parameter Description

hCtx  VB API context handle.

pItems  Address of variable to receive the Items of rate info structures.

Notes

This function can be called for regular databases with associated currency databases.

Return Value

If successful, this function returns a Items of structures in pItems and generates an array of
Currency info structures accessible via EsbGetNextItem().

Access

This function requires callers to have access to the database, and to have selected it as their active
database using EsbSetActive().

Example

Declare Function EsbGetCurrencyRateInfo Lib "ESBAPIN" (ByVal hCtx As Long, Items As Integer) As Long

Sub ESB_GetCurrencyRateInfo ()
    Dim Items As Integer
    Dim RateInfo As ESB_RATEINFO_T
    Dim sts As Long  '****************************
        ' Get Currency Rates Info
        '****************************
    sts = EsbGetCurrencyRateInfo (hCtx, Items)  For n = 1 To Items
        '****************************
        ' Get next Rates Info item
        'from the list
        '****************************
    sts = EsbGetNextItem (hCtx,
                ESB_RATEINFO_TYPE, RateInfo)
    Next
End Sub

See Also

- EsbListCurrencyDatabases
- EsbGetNextItem
- EsbSetActive
**EsbGetDatabaseAccess**

Gets a list of user database access structures, which contain information about user access to databases.

**Syntax**

```vba
EsbGetDatabaseAccess (hCtx, User, AppName, DbName, pItems)
```

- **hCtx**  As Long
- **User**  As String
- **AppName**  As String
- **DbName**  As String
- **pItems**  As Integer

**Parameter Description**

- **hCtx**  VB API context handle.
- **User**  User name. If an empty string, lists all users for the specified application and database.
- **AppName**  Application name. If an empty string, lists all applications and databases for the specified user.
- **DbName**  Database name. If an empty string, lists all databases for the specified user or application.
- **pItems**  Address of variable to receive Items of user database structures.

**Notes**

- If any of **User**, **AppName**, or **DbName** are an empty string, they will be treated as wild cards and all items of the appropriate type will be listed. If **AppName** is an empty string, **DbName** is assumed to also be an empty string. Any two of these arguments may be an empty string, but not all three.

- The **Access** field of the user database structure is used to represent the user's granted access to the database, whereas the **MaxAccess** field represents the user's highest access from all sources (e.g. via groups or default database access etc.).

- Filter access privileges are equivalent to ESB_PRIV_DBLOAD privileges.

**Return Value**

If successful, returns a Items of users/databases in **pItems**, and generates a list of user database structures accessible via `EsbGetNextItem()`.

**Access**

This function requires callers to have Database Design privilege (ESB_PRIV_DBDESIGN) for the specified database, unless they are getting their own database access information.

**Example**

Declare Function EsbGetDatabaseAccess Lib "ESBAPIN" (ByVal hCtx As Long, ByVal User As String, ByVal AppName As String, ByVal DbName As String, Items As Integer) As Long

Sub ESB_GetDatabaseAccess ()
    Dim Items As Integer
    Dim AppName As String
    Dim DbName As String
Dim User As String
Dim UserDb As ESB_USERDB_T
Dim sts As Long   AppName = "Sample"
DbName = "Basic"
User = "Joseph"   '********************
' Get Database Access
'********************
sts = EsbGetDatabaseAccess (hCtx,
User, AppName, DbName, Items)   For n = 1 To Items
' Get next User Database Access
' structure from the list
'********************
sts = EsbGetNextItem (hCtx,
ESB_USERDB_TYPE, UserDb)
Next
End Sub

**See Also**
- EsbGetApplicationAccess
- EsbGetUser
- EsbListUsers
- EsbSetDatabaseAccess
- EsbGetNextItem

**EsbGetDatabaseInfo**

Gets a database's information structure, which contains non user-configurable parameters for the database.

**Syntax**

```vbnet
EsbGetDatabaseInfo (hCtx, AppName, DbName, DbInfo, pItems)
```

**Parameter Description**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>VB API context handle</td>
</tr>
<tr>
<td>AppName</td>
<td>Application name</td>
</tr>
<tr>
<td>DbName</td>
<td>Database name</td>
</tr>
<tr>
<td>DbInfo</td>
<td>Buffer to receive a database info structure.</td>
</tr>
<tr>
<td>pItems</td>
<td>Number of &quot;ESB_DBREQINFO_T&quot; on page 1173 structures returned.</td>
</tr>
</tbody>
</table>

**Notes**

- This function can only get the information structure for a server database.
The caller of this routine must call EsbGetNextItem with the ESB_DBREQINFO_TYPE, which returns a structure of type ESB_DBREQINFO_T. These structures contain request information, including last calc, data load, and outline update.

**Return Value**

If successful, this function returns a pointer to a database infostructure in pDbInfo.

**Access**

This function requires the caller to have at least Read access(ESB_PRIV_READ) to the specified database.

**Example**

```visualbasic
Sub ESB_GetDatabaseInfo()
    Dim sts As Long
    Dim AppName As String
    Dim DbName As String
    Dim Items As Integer
    Dim n As Integer
    Dim DbInfo As ESB_DBINFO_T
    Dim DbReqInfo As ESB_DBREQINFO_T,
    Dim Items As Integer
    AppName = "Sample"
    DbName = "Basic"
    sts = EsbGetDatabaseInfo(hCtx, AppName, DbName, DbInfo, Items)
    If sts = 0 Then
        For n = 1 To Items
            sts = EsbGetNextItem(hCtx, ESB_DBREQINFO_TYPE, DbReqInfo)
            Next   End
    IfEnd Sub
```

**See Also**

- EsbGetApplicationInfo
- EsbGetDatabaseState
- EsbGetDatabaseStats

---

### EsbGetDatabaseInfoEx

Retrieves information for one or more databases, which contains non user-configurable parameters for the databases.

**Syntax**

```visualbasic
Declare Function EsbGetDatabaseInfoEx Lib "esbapin" (hCtx As Long, ByVal AppName As String, ByVal DbName As String, pItems As Integer) As Long
```
**Parameter**  | **Description**  
--- | ---  
`hCtx`  | API context handle.  
AppName  | Name of application for which to return database information. If NULL, returns information for all applications and databases.  
DbName  | Name of database for which to return database information. If NULL, returns information for all databases.  
pItems  | Number of information structures returned.  

**Notes**  
- The program must call `EsbGetNextItem()` with the `ESB_DBREQINFO_T` parameter.  
- This function can only get the information structure for server databases.  

**Return Value**  
If successful, this function returns a count of the number of accessible databases in `pCount`, and generates a list of the application and database names. The list is accessible through use of `EsbGetNextItem()`.  

**Access**  
This function requires the caller to have at least read access (ESS_PRIV_READ) to the specified database.

## EsbGetDatabaseNote  
Gets a database's note-of-the-day message. This message may be used to display useful information about the database (whether data has been loaded, when it was last calculated, etc.) to users before they connect to the database.  

**Syntax**  
```
EsbGetDatabaseNote (hCtx,AppName,DbName,DbNote,szDbNote)
```

**Parameter**  | **Description**  
--- | ---  
hCtx  | VB API context handle.  
AppName  | Application name.  
DbName  | Database name.  
DbNote  | Buffer to receive a database note string.  
`szDbNote`  | Size of the buffer.
Notes
- The database note string will always be less than 64 KB in length.
- If application/database name length is greater than the size of the buffer, the name will be truncated.
- The database’s note is set by EsbSetDatabaseNote().

Return Value
If successful, returns a database note string in DbNote.

Access
This function requires the caller to have access to the specified database.

Example
Declare Function EsbGetDatabaseNote Lib "ESBAPIN" (ByVal hCtx As Long, ByVal AppName As String, ByVal DbName As String, ByVal DbNote As String, ByVal szDbNote As Integer) As Long

Sub ESB_GetDatabaseNote ()
    Dim sts As Long
    Dim AppName As String
    Dim DbName As String
    Const szDbNote = 256
    Dim DbNote As String * szDbNote
    AppName = "Sample"
    DbName = "Basic"  '******************
    ' Get Database note
    '******************
    sts = EsbGetDatabaseNote (hCtx, AppName, DbName, DbNote, szDbNote)
End Sub

See Also
- EsbSetDatabaseNote

**EsbGetDatabaseState**
Gets a database's state structure, which contains user-configurable parameters for the database.

Syntax

EsbGetDatabaseState (hCtx, AppName, DbName, pDbState)

ByVal hCtx As Long
ByVal AppName As String
ByVal DbName As String
pDbState As ESB_DBSTATE_T

Parameter  Description

hCtx   VB API context handle.
AppName  Application name.
### EsbGetDatabaseState

**Parameter**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DbName</td>
<td>Database name.</td>
</tr>
<tr>
<td>pDbState</td>
<td>Buffer to receive a database state structure.</td>
</tr>
</tbody>
</table>

**Notes**

This function can get only a server database's state structure.

**Return Value**

If successful, this function returns a pointer to a database state structure in `pDbState`.

**Access**

To get a database's state structure, the connected user must have at least read access (ESB_PRIV_READ) to the database.

**Example**

Declare Function EsbGetDatabaseState Lib "ESBAPIN" (ByVal hCtx As Long, ByVal AppName As String, ByVal DbName As String, pDbState As ESB_DBSTATE_T) As Long

Sub ESB_GetDatabaseState()
    Dim sts As Long
    Dim AppName As String
    Dim DbName As String
    Dim DbState As ESB_DBSTATE_T
    AppName = "Sample"
    DbName = "Basic"
    '********************************
    '****** Get Database State *****
    '********************************
    sts = EsbGetDatabaseState(hCtx, AppName, DbName, DbState)
End Sub

**See Also**

- EsbGetApplicationState
- EsbGetDatabaseInfo
- EsbSetDatabaseState
- EsbGetDatabaseStats

### EsbGetDatabaseStats

Gets the active database's stats structure, which contains statistical information about the database.

**Syntax**

```
EsbGetDatabaseStats (hCtx, AppName, DbName, pDbStats, pItems)
```

* ByVal hCtx      As Long
* ByVal AppName   As String
* ByVal DbName    As String
  * pDbStats As ESB_PDBSTATS_T
  * pItems    As Integer

Visual Basic Main API Function Reference 1297
### Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>VB API context handle.</td>
</tr>
<tr>
<td>AppName</td>
<td>Application name.</td>
</tr>
<tr>
<td>DbName</td>
<td>Database name.</td>
</tr>
<tr>
<td>pDbStats</td>
<td>Buffer to receive a database stats structure.</td>
</tr>
<tr>
<td>pItems</td>
<td>Address of variable to receive Items of Dimension stats items.</td>
</tr>
</tbody>
</table>

### Notes
- This function can only be called for server databases.
- This function will load the database if it is not loaded.

### Return Value
If successful, this function returns a pointer to an allocated database stats structure in `pDbStats`, the number of dimensions in `pItems` and generates a list of Dimension stats structures accessible via `GetNextItem()`.

### Access
This function requires the caller to have access to the database, and to have selected it as their active database using `EsbSetActive()`.

### Example

```visualbasic
Declare Function EsbGetDatabaseStats Lib "ESBAPIN" (ByVal hCtx As Long, ByVal AppName As String, ByVal DbName As String, DbStats As ESB_DBSTATS_T, Items As Integer) As Long

Sub ESB_GetDatabaseStats ()
    Dim Items As Integer
    Dim AppName As String
    Dim DbName As String
    Dim DbStats As ESB_DBSTATS_T
    Dim DimStats As ESB_DIMSTATS_T
    Dim sts As Long
    AppName = "Sample"
    DbName = "Basic"
    ' Get Database stats
    '**************************
    sts = EsbGetDatabaseStats (hCtx, AppName, DbName, DbStats, Items) For n = 1 To Items
    ' Get next Dimension stats item from the list
    '**************************
    sts = EsbGetNextItem (hCtx, ESB_DIMSTATS_TYPE, DimStats)
Next
End Sub
```

### See Also
- `EsbGetDatabaseInfo`
- `EsbGetDatabaseState`
**EsbGetDefaultCalc**

Gets the default calc script for the active database.

**Syntax**

```
EsbGetDefaultCalc (hCtx, cscString, szString)
```

- **hCtx** As Long
- **cscString** As String
- **szString** As Integer

**Parameter Description**

- **hCtx** VB API context handle.
- **cscString** Buffer to receive a calc script string.
- **szString** Size of the buffer to receive a calc script string.

**Notes**

- The returned calc script string is less than 64 KB long.
- If the CalcScript length is greater that the size of the buffer, the script is truncated.

**Return Value**

If successful, this function returns the default calc script for the database in `CalcScript`.

**Access**

This function requires the caller to have at least read access (ESB_PRIV_READ) to the database, and to have selected it as their active database using `EsbSetActive()`.

**Example**

```
Declare Function EsbGetDefaultCalc Lib "ESBAPIN" (ByVal hCtx As Long, ByVal Script As String, ByVal szScript As Integer) As Long

Sub ESB_GetDefaultCalc ()
    Dim sts As Long
    Const szScript = 1024
    Dim Script As String * szScript   '*****************
    ' Get default calc
    '*****************
    sts = EsbGetDefaultCalc (hCtx, Script, szScript)
End Sub
```

**See Also**

- `EsbDefaultCalc`
- `EsbSetDefaultCalc`
- `EsbSetDefaultCalcFile`
- `EsbSetActive`
EsbGetDimensionInfo

Gets dimension information.

Syntax

EsbGetDimensionInfo (hCtx, Dimension, pItems)
ByVal hCtx As Long
ByVal Dimension As String
pItems As Integer

Parameter  Description
hCtx        API context handle.
Dimension    Member name of dimension for which to return information. If NULL, returns information about each
dimension.
pItems      Number of information structures returned.

Notes

- The caller must call EsbGetNextItem with the ESB_DIMINFO_TYPE parameter.
- Attributes:
  - The constant values ESB_TTYPE_ATTRIBUTE and ESB_TTYPE_ATTRCALC for
    theDimTag field of the “ESB_DIMENSIONINFO_T” on page 1177 structure indicate
    that the dimension is an attribute dimension.
  - TheDimDataType field of the “ESB_DIMENSIONINFO_T” on page 1177 structure
    indicates the type of attribute dimension.

Return Value

If successful, returns a reference to the number of dimension information structures.

Access

This function requires the caller to have database design privilege for the specified database
(ESB_PRIV_DBDESIGN).

Example

Declare Function EsbGetDimensionInfo Lib "ESBAPIN" (ByVal hCtx As Long, ByVal Dimension As String, pItems As Integer) As Long

Sub ESB_GetDimensionInfo()
    Dim sts As Long
    Dim Dimension As String
    Dim nDims As Integer
    Dim DimInfo As ESB_DIMENSIONINFO_T
    Dimension = "Year"
    sts = EsbGetDimensionInfo(hCtx, Dimension, nDims)  If Not sts Then
        For n = 1 To nDims
            '********************
            ' Get next Dimension Info
            ' from the list
sts = EsbGetNextItem(hCtx, ESB_DIMINFO_TYPE, DimInfo)
   Next
End If
End Sub

See Also
- EsbBuildDimension
- EsbGetApplicationInfo
- EsbGetApplicationInfoEx
- EsbGetDatabaseInfo

**EsbGetDrillThruURL**

Gets a list of drill-through URL names within the active database outline.

“Drill-through URL Limits” on page 1729.

**Syntax**

Declare Function EsbGetDrillThruURL Lib "esbapin" (ByVal hCtx As Long, ByVal URLName As String, pUrl As ESB_DURLINFO_T, ByRef symRegions As Variant) As Long

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>Visual Basic API context handle</td>
</tr>
<tr>
<td>URLName</td>
<td>Drill-through URL name</td>
</tr>
<tr>
<td>pUrl</td>
<td>URL definition</td>
</tr>
<tr>
<td>symRegions</td>
<td>List of symmetric regions</td>
</tr>
</tbody>
</table>

**Return Value**

- If successful, gets a list of drill-through URLs in the active database outline.
- If unsuccessful, returns an error code.

**Access**

- Caller must have database Read privilege (ESB_PRIV_READ) for the specified database.
- Caller must have selected the specified database as their active database using EsbSetActive().

**Example**

```vbnet
Sub ESB_GetGLDrillThru()
   Dim URLName     As String
   Dim url         As ESB_DURLINFO_T
   Dim intX        As Integer
   Dim cppDrillRegions As Variant

   URLName = "VB URL2"
   sts = EsbGetDrillThruURL(hCtx, URLName, url, cppDrillRegions)
End Sub
```
Debug.Print "EsbGetDrillThruURL sts: " & sts

If sts = 0 Then
    Debug.Print "URL Name: " & url.cpURLName
    Debug.Print "URL XML: " & url.cpURLXML
    For intX = LBound(cppDrillRegions) To UBound(cppDrillRegions)
        Debug.Print "URL Region: " & cppDrillRegions(intX)
    Next
End If
End Sub

See also an extended example in “Drill-through Visual Basic API Example” on page 1132.

**EsbGetFilter**

Starts getting the contents of a filter.

**Syntax**

```vbnet
EsbGetFilter (hCtx, AppName, DbName, FltName, pItems)
ByVal hCtx As Long
ByValAppName As String
ByVal DbName As String
ByVal FltName As String
    pItems As Integer
```

**Parameter Description**

- **hCtx** VB API context handle
- **AppName** Application name
- **DbName** Database name
- **FltName** Filter name
- **pItems** Address of variable to receive Items of user application structures.

**Notes**

This call must be followed by successive calls to **EsbGetFilterRow()** to fetch the rows for the filter.

**Return Value**

If successful, returns the filter active flag in **pActive**, and the default filter access level in **pAccess**.

**Access**

This function requires the caller to have Database Design privilege (ESB_PRIV_DBDESIGN) for the specified database.
Example

Declare Function EsbGetFilter Lib "ESBAPIN" (ByVal hCtx As Long, ByVal AppName As String, ByVal DbName As String, ByVal FltName As String, Active As Integer, pAccess As Integer) As Long

Sub ESB_GetFilter ()
    Dim sts As Long
    Dim AppName As String
    Dim DbName As String
    Dim FilterName As String
    Dim Active As Integer
    Dim pAccess As Integer
    Const szRow = 512
    Dim Row As String * szRow
    AppName = "Sample"
    DbName = "Basic"
    FilterName = "Filter"  '***********
    ' Get Filter
    '***********
    sts = EsbGetFilter (hCtx, AppName, DbName, FilterName, Active, pAccess)  '***********
    ' Get Filter Rows
    '***********
    sts = EsbGetFilterRow (hCtx, Row, szRow, pAccess)
    Do While Mid$(Row,1,1) <> chr$(0)
        sts = EsbGetFilterRow (hCtx, Row, szRow, pAccess)
    Loop
End Sub

See Also

- EsbGetFilterRow
- EsbListFilters
- EsbSetFilter

EsbGetFilterList

Gets the list of users who are assigned a filter.

Syntax

EsbGetFilterList (hCtx, AppName, DbName, FltName, pItems)
ByVal hCtx As Long
ByVal AppName As String
ByVal DbName As String
ByVal FltName As String
    pItems As Integer

Parameter Description

hCtx VB API context handle.
AppName Application name.
DbName Database name.
**Parameter**  | **Description**  
---|---  
FltName | Filter name.  
pItems | Address of variable to receive Items of users assigned this filter.  

**Return Value**  
If successful, returns a Items of the users assigned this filter in pItems, and generates an array of user name strings accessible via EsbGetNextItem().

**Access**  
This function requires the caller to have Database Design privilege (ESB_PRIV_DBDESIGN) for the specified database.

**Example**  
Declare Function EsbGetFilterList Lib "ESBAPIN" (ByVal hCtx As Long, ByVal AppName As String, ByVal DbName As String, ByVal FilterName As String, Items As Integer) As Long

Sub ESB_GetFilterList ()  
    Dim Items As Integer  
    Dim AppName As String  
    Dim DbName As String  
    Dim FilterName As String  
    Dim User As String * ESB_USERNAMELEN  
    Dim sts As Long   
    AppName = "Sample"  
    DbName = "Basic"   
    FilterName = "Filter"   
    ' Get Filter List  
    '****************  
    sts = EsbGetFilterList (hCtx, AppName, DbName, FilterName, Items)   
    For n = 1 To Items   
        ' Get next User Name String  
        ' from the list  
        '****************  
        sts = EsbGetNextItem (hCtx, ESB_FUSERNAME_TYPE, ByVal User)  
    Next  
End Sub  

**See Also**  
- EsbGetFilter  
- EsbListFilters  
- EsbSetFilterList  
- EsbGetNextItem

EsbGetFilterRow

Gets the next row of a filter.
### Syntax

**EsbGetFilterRow** (hCtx, FltRow, szRow, pAccess)

- **hCtx** As Long
- **FltRow** As String
- **szRow** As Integer
- **pAccess** As Integer

### Parameter Description

- **hCtx** VB API context handle.
- **FltRow** Buffer to receive the next row of the filter.
- **szRow** Size of the buffer to receive the next row of the filter.
- **pAccess** Address of variable to receive the access level for the filter row.

### Notes

- This function should be called repeatedly after calling **EsbGetFilter()**, until an empty string is returned.
- If the filter row string length is greater than the size of the buffer, the filter row is truncated.

### Return Value

If successful, returns the next filter row (if any) in **RowString**, and the row access level in **pAccess**.

### Access

This function requires the caller to have Database Design privilege (ESB_PRIV_DBDESIGN) for the specified database.

### Example

Declare Function EsbGetFilterRow Lib "ESBAPIN" (ByVal hCtx As Long, ByVal FltRow As String, ByVal szRow As Integer, Access As Integer) As Long

See the example for **EsbGetFilter**.

### See Also

- **EsbGetFilter**
- **EsbListFilters**

---

**EsbGetGlobalState**

Gets the server global state structure which contains parameters for system administration.

### Syntax

**EsbGetGlobalState** (hCtx, pGlobal)

- **hCtx** As Long
- **pGlobal** As ESB_GLOBAL_T
**EsbGetGlobalState**

Gets a group information structure, which contains security information for the group.

**Syntax**

```vbnet
EsbGetGlobalState (hCtx, GrpName, pUserInfo)
```

**Parameter Description**

- **hCtx** VB API context handle.
- **GrpName** Group name.
- **pUserInfo** Buffer to receive a group info structure.

**Notes**

In the `pGlobal` structure, the statement returns: global security status, global login status (enabled/disabled), global default access level, global password validity (in days), global currency enabled flag, global minimum password length, global auto-logout time (in seconds).

**Return Value**

If successful, returns the current state of the server global state structure in `pGlobal`.

**Access**

This function requires the caller to be an administrator.

**Example**

```vbnet
Declare Function EsbGetGlobalState Lib "ESBAPIN" (ByVal hCtx As Long, Global As ESB_GLOBAL_T) As Long

Sub ESB_GetGlobalState ()
    Dim sts As Long
    Dim pGlobal As ESB_GLOBAL_T   '*****************
    ' Get Global State
    '*****************
    sts = EsbGetGlobalState (hCtx, pGlobal)
End Sub
```

**See Also**

- [EsbSetGlobalState](#)
- [EsbGetGroup](#)
Return Value

If successful, returns the group information structure in pGroupInfo.

Access

This function requires the caller to have Create/Delete User privilege (ESB_PRIV_USERCREATE) for the logged in server.

Example

Declare Function EsbGetGroup Lib "ESBAPIN" (ByVal hCtx As Long, ByVal GroupName As String, UserInfo As ESB_USERINFO_T) As Long

Sub ESB_GetGroup ()
    Dim sts As Long
    Dim GroupName As String
    Dim GroupInfo As ESB_USERINFO_T   
    GroupName = "PowerUsers"
    '************************
    ' Get GroupInfo structure
    '************************
    sts = EsbGetGroup (hCtx, GroupName, GroupInfo)
End Sub

See Also

- EsbListGroups
- EsbSetGroup

**EsbGetGroupList**

Gets the list of users who are members of a group (or the list of groups to which a user belongs).

Syntax

EsbGetGroupList (hCtx, GrpName, pItems)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>VB API context handle.</td>
</tr>
<tr>
<td>GrpName</td>
<td>Group or user name.</td>
</tr>
<tr>
<td>pItems</td>
<td>Address of variable to receive Items of user names.</td>
</tr>
</tbody>
</table>

Notes

- This function can also be used to get the list of groups to which a user belongs, by using a user name as the GroupName argument.

Return Value

If successful, returns a Items of user names in pItems, and generates an array of user name strings accessible via EsbGetNextItem().
**Access**

This function requires the caller to have Create/Delete User privilege (ESB_PRIV_USERCREATE) for the logged in server, unless they are a user getting their own list of groups.

**Example**

Declare Function EsbGetGroupList Lib "ESBAPIN" (ByVal hCtx As Long, ByVal GroupName As String, Items As Integer) As Long

Sub ESB_GetGroupList ()
    Dim Items As Integer
    Dim Group As String
    Dim GroupName As String * ESB_USERNAMELEN
    Dim sts As Long
    Group = "User Group"  '**********************
    ' Get Group List
    '**********************
    sts = EsbGetGroupList (hCtx, Group, Items)  For n = 1 To Items
    '**************************
    ' Get next User Name String
    ' from the list
    '**************************
    sts = EsbGetNextItem (hCtx, ESB_GROUPNAME_TYPE, ByVal GroupName)
Next
End Sub

**See Also**

- EsbAddToGroup
- EsbDeleteFromGroup
- EsbListGroups
- EsbSetGroupList
- EsbGetNextItem

**EsbGetLocalPath**

Gets the full local file path for a specific object file on the client.

**Syntax**

```
EsbGetLocalPath (hCtx, ObjType, AppName, DbName, ObjName, isCreate, Path, szPath)
```

- **hCtx**: API context handle returned by EsbCreateLocalContext().
Parameter Description

ObjType Object type (must be single type). See Table 15, “Bitmask Data Types,” on page 1161 for a list of object types.

AppName Application name.

DbName Database name. If an empty string, uses the Application sub-directory.

ObjName Object name.

isCreate Create directories flag. If TRUE, the appropriate application and database sub-directories will be created if necessary. If FALSE, and the directories do not exist, an error will be generated.

Path Buffer to receive allocated local path name string.

szPath Size of the buffer to receive allocated local path name string.

Notes
If the Path string length is greater that the size of the buffer, the Path string is truncated.

Return Value
If successful, returns the full path name of the appropriate object file in Path.

Access
This function requires no special privileges.

Example
Declare Function EsbGetLocalPath Lib "ESBAPIN" (ByVal hCtx As Long, ByVal ObjType As Integer, ByVal AppName As String, ByVal DbName As String, ByVal ObjName As String, ByVal Create As Integer, ByVal Path As String, ByVal szPath As Integer) As Long

Sub ESB_GetLocalPath ()
    Dim sts As Long
    Dim AppName As String
    Dim DbName As String
    Dim ObjName As String
    Dim ObjType As Integer
    Dim Create As Integer
    Const szPath = 128
    Dim Path As String * szPath
    AppName = "Sample"
    DbName = "Basic"
    ObjName = ESB_OBJTYPE_TEXT
    Create = ESB_YES '**************
    ' Get local path
    '**************
    sts = EsbGetLocalPath (hCtx, ObjType, AppName, DbName, ObjName, Create, Path, szPath)
End Sub

See Also
- EsbCreateLocalContext
- EsbListObjects
**EsbGetLocationAliasList**

Returns a list of all currently-defined location aliases, together with lists of the host names, application names, database names and user names to which the location aliases are mapped.

**Syntax**

```vbnet
EsbGetLocationAliasList (hCtx, ListCount, Aliases, Hosts, _
    AppNames, DbNames, UserNames)
```

- **hCtx** as Long
- **ListCount** as Integer
- **Aliases** as Variant
- **Hosts** as Variant
- **AppNames** as Variant
- **DbNames** as Variant
- **UserNames** as Variant

**Parameter Description**

- **hCtx** API context handle
- **ListCount** Number of location aliases returned
- **Aliases** List of location aliases returned
- **Hosts** List of hosts returned
- **AppNames** List of applications returned
- **DbNames** List of databases returned
- **UserNames** List of user logins returned

**Notes**

- **hCtx** is the only input parameter.
- **ListCount, Aliases, Hosts, AppNames, DbNames** and **UserNames** are output parameters; that is, values returned by reference.

**Example**

```vbnet
Public Sub LocationAliasTest()

    Dim status As Long
    Dim ListCount As Integer
    Dim Aliases As Variant
    Dim HostNames As Variant
    Dim AppNames As Variant
    Dim DbNames As Variant
    Dim UserNames As Variant

    status = EsbCreateLocationAlias(hCtx, "bla1", "LocalHost", "Demo", "Basic", _
        "admin", "password")

    If (status <> 0) Then
        MsgBox "Create routine Failed"
        Exit Sub
    End If

End Sub
```

1310 Visual Basic Main API Functions
status = EsbCreateLocationAlias(hCtx, "blah2", "LocalHost", "Demo", "Basic", _
        "admin", "password")
If (status <> 0) Then
    MsgBox "Create routine Failed"
    Exit Sub
End If

status = EsbGetLocationAliasList(hCtx, ListCount, Aliases, HostNames, _
        AppNames, DbNames, UserNames)
If (status <> 0) Then
    MsgBox "Get routine Failed"
    Exit Sub
End If

If (ListCount > 0) Then
    ' Retrieve the elements as Aliases(0) to Aliases(ListCount -1)
End If

status = EsbDeleteLocationAlias(hCtx, "blah1")
If (status <> 0) Then
    MsgBox "Delete routine Failed"
    Exit Sub
End If

status = EsbGetLocationAliasList(hCtx, ListCount, Aliases, HostNames, _
        AppNames, DbNames, UserNames)
If (status <> 0) Then
    MsgBox "Get routine Failed"
    Exit Sub
End If

End Sub

See Also
- EsbCreateLocationAlias
- EsbDeleteLocationAlias

EsbGetLogFile
Copies all or part of an application log file from the server to the client.

Syntax

EsbGetLogFile (hCtx, AppName, TimeStamp, LocalName)
ByVal hCtx As Long
ByVal AppName As String
ByVal TimeStamp As String
ByVal LocalName As String,

Parameter   Description
hCtx         VB API context handle
AppName      Application name. If AppName = "," the Essbase.log is returned.
### Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TimeStamp</td>
<td>Time stamp, indicating date &amp; time of earliest log file entry required</td>
</tr>
<tr>
<td>LocalName</td>
<td>Full path name of local destination file on client</td>
</tr>
</tbody>
</table>

### Notes

The time represented by `TimeStamp` is the number of seconds elapsed since midnight (00:00:00) Greenwich Mean Time on January 1, 1970. Only log file entries which occurred after the date & time specified by `TimeStamp` will be copied to the client. If `TimeStamp` is set to 0 (zero), the entire log file will be copied.

### Return Value

If successful, the object is copied to the local file specified by `ByVal`.

### Access

This function requires the caller to have Application Design privilege (ESB_PRIV_APPDESIGN), or Database Design privilege (ESB_PRIV_DBDESIGN) for the specified application or any of its databases.

### Example

```visual-basic
Declare Function EsbGetLogFile Lib "ESBAPIN" (ByVal hCtx As Long, ByVal AppName As String, ByVal TimeStamp As Long, ByVal LocalName As String) As Long
Sub ESB_GetLogFile ()
    Dim sts As Long
    Dim AppName As String
    Dim TimeStamp As Long
    Dim LocalName As String
    AppName = "Sample" '***************
    ' Get everything
    '***************
    TimeStamp = 0
    LocalName = "c:\essbase\client\test.log"  '***************
    ' Get Log File
    '***************
    sts = EsbGetLogFile (hCtx, AppName, TimeStamp, LocalName)
End Sub
```

### See Also

- **EsbDeleteLogFile**

### EsbGetMemberCalc

Gets the calc equation for a specific member in the active database outline.

### Syntax

```visual-basic
EsbGetMemberCalc (hCtx, MbrName, MbrCalc, szMbrCalc, MbrLastCalc, szMbrLastCalc)
ByVal hCtx    As Long
ByVal MbrName As String
ByVal MbrCalc As String
ByVal szMbrCalc As Integer
```
ByVal MbrLastCalc As String
ByVal szMbrLastCalc As Integer

Parameter     Description

hCtx            VB API context handle.
MbrName        Member name.
MbrCalc        Buffer to receive a member calc string.
szMbrCalc      Size of the buffer to receive a member calc string.
MbrLastCalc    Buffer to receive a member last calc string.
szMbrLastCalc  Size of the buffer to receive a member last calc string.

Notes

- The last calc string is the formula used to calculate the member the last time the database was calculated. It might be left from LastCalcStr if a calc script was used to calculate the database.
- If Calc/LastCalc string length is greater than the size of the buffer, the string will be truncated.

Return Value

If successful, this function returns the calc string and last calc string in CalcCtr and LastCalcStr.

Access

This function requires the caller to have at least read access (ESB_PRIV_READ) to the database, and to have selected it as their active database using EsbSetActive().

Example

Declare Function EsbGetMemberCalc Lib "ESBAPIN" (ByVal hCtx As Long, ByVal MbrName As String, ByVal Calc As String, ByVal szCalc As Integer, ByVal LastCalc As String, ByVal szLastCalc As Integer) As Long

Sub ESB_GetMemberCalc ()
    Dim sts As Long
    Dim MbrName As String
    Const szCalc = 256
    Dim Calc As String * szCalc
    Const szLastCalc = 256
    Dim LastCalc As String * szLastCalc
    MbrName = "Year"    '****************
    ' Get Member Calc
    ' ****************
    sts = EsbGetMemberCalc (hCtx, MbrName, Calc, szCalc, LastCalc, szLastCalc)
End Sub

See Also

- EsbGetMemberInfo
**EsbSetActive**

**EsbGetMemberInfo**

Gets a structure containing information about a specific member in the active database outline.

**Syntax**

```vbnet
EsbGetMemberInfo (hCtx, MbrName, MbrInfo)
ByVal hCtx As Long
ByVal MbrName As String
MbrInfo As ESB_MEMBERINFO_T
```

**Parameter Description**

- **hCtx** VB API context handle.
- **MbrName** Member name.
- **MbrInfo** Buffer to receive a member information structure.

**Notes**

Attributes:

- The ESB_MBRSTS_ATTRIBUTE constant for the Status field of the "ESB_MEMBERINFO_T" on page 1182 structure indicates that the dimension or member is an attribute dimension or attribute member.
- Two fields of the ESB_MEMBERINFO_T structure are for attributes only:
  - Attribute
  - IsAttributed

**Return Value**

If successful, this function returns a member information structure in `pMbrInfo`. If the member has no parent, this function returns an empty string in the ParentMbrName field of the ESB_MEMBERINFO_T structure.

**Access**

This function requires the caller to have access to the database, and to have selected it as their active database using `EsbSetActive()`.

**Example**

```vbnet
Declare Function EsbGetMemberInfo Lib "ESBAPIN" (ByVal hCtx As Long,
ByVal MbrName As String,
MbrInfo As ESB_MEMBERINFO_T) As Long

Sub ESB_GetMemberInfo ()
    Dim sts As Long
    Dim MbrName As String
    Dim MbrInfo As ESB_MEMBERINFO_T
    MbrName = "Year"
```

1314 Visual Basic Main API Functions
'**************************
'Get Member Info structure
'**************************

nts = EsbGetMemberInfo (hCtx, MbrName, MbrInfo)
End Sub

See Also

- EsbCheckMemberName
- EsbGetMemberCalc
- EsbQueryDatabaseMembers
- EsbSetActive

EsbGetMessage

Retrieves the top message from the message stack filled during VB API function execution if
ClientError in ESB_INIT_T structure has been set to ESB_TRUE during EsbInit().

Syntax

EsbGetMessage (hInst, ErrLevel, ErrNum, ErrMessage, szErrMessage)
ByVal hInst As Long
ErrLevel As Integer
ErrNum As Long
ByVal ErrMessage As String
ByVal szErrMessage As Integer

Parameter Description

hInst VB API instance handle.
ErrLevel Pointer to variable to receive message level.
ErrNum Pointer to variable to receive message number in the message database.
ErrMessage Buffer to receive a message string.
szErrMessage Size of the buffer to receive a message string.

Notes

- Message stack is being initialized every time a VB API function is called. All messages from
the previous call are lost.
- All messages including information, warning, and error messages go into the message stack.
- If the number of messages generated during one VB API function call exceeds the setting in
ErrorStack from ESB_INIT_T (or default) new messages overwrite old messages.
- If no more messages in the stack Message is reset to empty string, pNumber and pLevel are
reset to zero.
- If the message string length is greater that the size of the buffer, the message is truncated.
Return Value
If successful, returns a pointer to a message level, a pointer to a message number, and a message string. Also decrements internal message stack pointer.

Access
This function requires no special privileges.

Example
Declare Function EsbGetMessage Lib "ESBAPIN" (ByVal hInst As Long, ErrLevel As Integer, ErrNum As Long, ByVal Message As String, ByVal szMessage As Integer) As Long

Sub ESB_GetMessage ()
    Dim Items As Integer
    Dim AppName As String
    Dim DbName As String
    Dim FilterName As String
    Const szMessage = 256
    Dim Message As String * szMessage
    Dim Number As Long
    Dim Level As Integer
    Dim sts As Long
    AppName = "Demo"
    DbName = "Basic"
    FilterName = "Filter" '****************
    ' Get Filter List
    '******************************
    sts = EsbGetFilterList (hCtx, AppName, DbName, FilterName, Items) '******************************
    ' Process all messages if error occurred till the bottom of the message stack enItemsered
    '****************
    If sts > 0 Then
        sts = EsbGetMessage (hInst, Level, Number, Message, szMessage)
        Do While Mid$(Message, 1, 1) <> Chr$(0)
            Print Level
            Print Number
            Print Message
            sts = EsbGetMessage (hInst, Level, Number, Message, szMessage)
        Loop
    End If
End Sub

See Also
- EsbAutoLogin
- EsbInit

EsbGetNextItem
Retrieves next item from an array or list generated by calling other VB API functions.
Syntax

EsbGetNextItem (hCtx, dType, pItem)
ByVal hCtx As Long
ByVal dType As Integer
ByRef pItem As Any

Parameter Description

hCtx VB API context handle.
dType Type of the array/list to retrieve item from: ESB_xxx...xx_TYPE, where xxx...xx is the name of a global constant, such as “ESB_USERINFO_T, ESB_GROUPINFO_T” on page 1193.
pItem Buffer to receive the next item.

Notes

- This function retrieves items from the list/array and must be called in a FOR loop based on the number of items returned by the VB API function that generates the array/list.
- pItem is a buffer of a necessary data type.

Return Value

If successful, returns an item in pItem. Returns -1 on failure to indicate a wrong Type or 1 to indicate that there are no more items in the list.

Access

This function requires no special privileges.

Example

 Declare Function EsbGetNextItem Lib "ESBAPIN" (ByVal hCtx As Long, ByVal dType As Integer, pItem As Any)

Sub ESB_GetNextItem ()
    Dim Items As Integer
    Dim AppName As String
    Dim DbName As String
    Dim FilterName As String
    Dim User As String * ESB_USERNAMELEN
    Dim sts As Long   AppName = "Demo"
                    DbName = "Basic"
                    FilterName = "Filter"   '***************
                        'Get Filter List
                '*******************
                sts = EsbGetFilterList (hCtx, AppName, DbName, FilterName, Items)   '*******************
                        'Print out all user names
                'from the list
                '*******************
                For n = 1 To Items   '**************************
                        'Get next User Name String
                'from the list
                        '**************************
                    sts = EsbGetNextItem (hCtx, ESB_USERINFO_TYPE, Userinfo)
                        Print User
Global Constants for EsbGetNextItem

<table>
<thead>
<tr>
<th>Constant</th>
<th>Context</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESB_USERINFO_TYPE = 1</td>
<td>ESB_USERINFO_T (EsbListUsers)</td>
</tr>
<tr>
<td>ESB_GROUPINFO_TYPE = 2</td>
<td>ESB_USERINFO_T (EsbListGroups)</td>
</tr>
<tr>
<td>ESB_USERAPP_TYPE = 3</td>
<td>ESB_USERAPP_T (EsbGetApplicationAccess)</td>
</tr>
<tr>
<td>ESB_USERDB_TYPE = 4</td>
<td>ESB_USERDB_T (EsbGetDatabaseAccess)</td>
</tr>
<tr>
<td>ESB_LOCKINFO_TYPE = 5</td>
<td>ESB_LOCKINFO_T (EsbListLocks)</td>
</tr>
<tr>
<td>ESB_OBJINFO_TYPE = 6</td>
<td>ESB_OBJINFO_T (EsbListObjects)</td>
</tr>
<tr>
<td>ESB_APPDB_TYPE = 7</td>
<td>ESB_APPDB_T (EsbListDatabases)</td>
</tr>
<tr>
<td>ESB_CAPPDB_TYPE = 8</td>
<td>ESB_APPDB_T (EsbListCurrencyDatabase)</td>
</tr>
<tr>
<td>ESB_APPNAME_TYPE = 9</td>
<td>ByVal var As String * ESB_APPNAMELEN (EsbListApplications)</td>
</tr>
<tr>
<td>ESB_DBNAME_TYPE = 10</td>
<td>ByVal var As String * ESB_DBNAMELEN (EsbGetApplicationInfo)</td>
</tr>
<tr>
<td>ESB_GROUPNAME_TYPE = 11</td>
<td>ByVal var As String * ESB_USERNAMELEN (EsbGetGroupList)</td>
</tr>
<tr>
<td>ESB_FTRNAME_TYPE = 12</td>
<td>ByVal var As String * ESB_FTRNAMELEN (EsbListFilters)</td>
</tr>
<tr>
<td>ESB_USERNAME_TYPE = 13</td>
<td>ByVal var As String * ESB_USERNAMELEN (EsbGetFilterList)</td>
</tr>
<tr>
<td>ESB_OBJNAME_TYPE = 14</td>
<td>ByVal var As String * ESB_OBJNAMELEN (EsbGetCalcList)</td>
</tr>
<tr>
<td>ESB_DIMSTATS_TYPE = 15</td>
<td>ESB_DIMSTATS_T (EsbGetDatabaseStats)</td>
</tr>
<tr>
<td>ESB_USERINFO_TYPE = 16</td>
<td>ESB_USERINFO_T (EsbListConnections)</td>
</tr>
<tr>
<td>ESB_LAPPDB_TYPE = 17</td>
<td>ESB_APPDB_T (EsbLogin)</td>
</tr>
<tr>
<td>ESB_ALIASNAME_TYPE = 18</td>
<td>ESB_ALIASNAME_T (EsbListAliases)</td>
</tr>
<tr>
<td>ESB_MBRALT_TYPE = 19</td>
<td>ESB_MBRALT_T (EsbDisplayAlias)</td>
</tr>
<tr>
<td>ESB_RATEINFO_TYPE = 20</td>
<td>ESB_RATEINFO_T (EsbGetCurrencyRateInfo)</td>
</tr>
<tr>
<td>ESB_OUTLINEINFO_TYPE = 21</td>
<td>ByVal var As String * ESB_ALIASNAMELEN (EsbOtlGetOutlineInfo)</td>
</tr>
<tr>
<td>ESB_OUTERROR_TYPE = 22</td>
<td>ESB_OUTERROR_T (EsbOtlVerifyOutline)</td>
</tr>
<tr>
<td>ESB_OTLUSERATTR_TYPE = 23</td>
<td>ByVal var As String * ESB_MBRNAMELEN (EsbOtlGetUserAttributes)</td>
</tr>
<tr>
<td>ESB_APPINFOEX_TYPE = 24</td>
<td>ESB_APPINFOEX_T (EsbGetApplicationInfoEx)</td>
</tr>
<tr>
<td>ESB_DBREQINFO_TYPE = 25</td>
<td>ESB_DBREQINFO_T (EsbGetDatabaseInfo)</td>
</tr>
<tr>
<td>ESB_DIMINFO_TYPE = 26</td>
<td>ESB_DIMENSIONINFO_T (EsbGetDimensionInfo)</td>
</tr>
<tr>
<td>Constant</td>
<td>Context</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>----------------------------------------------</td>
</tr>
<tr>
<td>ESB_HMEMBER_TYPE = 27</td>
<td>ESB_HMEMBER_T (EsbOtlQueryMembers)</td>
</tr>
<tr>
<td>ESB_GENLEVELNAME_TYPE = 28</td>
<td>ESB_GENLEVELNAME_T (EsbOtlGetGenNames)</td>
</tr>
<tr>
<td>ESB_VARIABLE_TYPE = 29</td>
<td>ESB_VARIABLE_T (EsbListVariables)</td>
</tr>
<tr>
<td>ESB_LRO_TYPE = 30</td>
<td>ESB_LRODESC_T</td>
</tr>
<tr>
<td>ESB_PART_INFO_TYPE = 31</td>
<td>ESB_PART_INFO_T</td>
</tr>
<tr>
<td>ESB_DTS_TYPE = 32</td>
<td>ESB_DTSMBRINFO_T (EsbGetEnabledDTSMembers)</td>
</tr>
<tr>
<td>ESB_MBRNAME_TYPE = 33</td>
<td>ESB_MBRNAME_T (EsbOtlGetDimensionUserAttributes)</td>
</tr>
</tbody>
</table>

See Also

- EsbListUsers
- EsbListGroups
- EsbGetApplicationAccess
- EsbGetDatabaseAccess
- EsbListLocks
- EsbListObjects
- EsbListDatabases
- EsbListCurrencyDatabases
- EsbListApplications
- EsbGetApplicationInfo
- EsbGetApplicationInfoEx
- EsbGetGroupList
- EsbListFilters
- EsbGetFilterList
- EsbGetCalcList
- EsbGetDatabaseState
- EsbGetDatabaseStats
- EsbListConnections
- EsbLogin
- EsbListAliases
- EsbDisplayAlias
- EsbGetCurrencyRateInfo
- EsbLROGetMemberCombo
- EsbOtlQueryMembersByName

**EsbGetObject**

Copies an object from the server or client object system to a local file, and optionally locks it.

**Syntax**

```plaintext```
EsbGetObject (hCtx, ObjType, AppName, DbName, ObjName, LocalName, isLock)
```

ByVal hCtx As Long
Parameter | Description
--- | ---
hCtx | VB API context handle. Can be local context handle returned by `EsbCreateLocalContext()`.
ObjType | Object type (must be single type). Refer to "Bitmask Types."
AppName | Application name.
DbName | Database name. If an empty string, uses the Application sub-directory.
ObjName | Name of an object to get.
LocalName | Full path name of a local destination file on client.
isLock | Flag to control object locking. If TRUE, the server object is locked to prevent updates by other users.

**Notes**

To lock an object, it must already exist on the server and not be locked by another user. Locking is not supported on the client.

**Return Value**

If successful, the object is copied to the local file specified by `LocalName`.

**Access**

This function requires the caller to have the appropriate level of access to the specified application and/or database containing the object (depending on the object type). To lock the object (lock flag is TRUE), the caller must have Application or Database Manager privilege (`ESB_PRIV_APPDESIGN` or `ESB_PRIV_DBDESIGN`) for the specified application or database containing the object.

**Example**

```vba
Declare Function EsbGetObject Lib "ESBAPIN" (ByVal hCtx As Long, ByVal ObjType As Integer, ByVal AppName As String, ByVal DbName As String, ByVal ObjName As String, ByVal LocalName As String, ByVal Lock As Integer) As Long

Sub ESB_GetObject ()
    Dim sts As Long
    Dim AppName As String
    Dim DbName As String
    Dim ObjName As String
    Dim ObjType As Long
    Dim LocalName As String
    Dim Lock As Integer
    AppName = "Sample"
    DbName = "Basic"
    ObjName = "Basic"
    ObjType = ESB_OBJTYPE_OUTLINE
End Sub
```
LocalName = "C:\ESSBASE\CLIENT\BASIC.OTL"
Lock = ESB_YES '***********
' Get Object
'***********
sts = EsbGetObject (hCtx, ObjType, AppName,
   DbName, ObjName, LocalName, Lock)
End Sub

See Also
- EsbGetObjectInfo
- EsbListObjects
- EsbLockObject
- EsbPutObject
- EsbUnlockObject

EsbGetObjectInfo

Gets information about a specific object on the server or locally on the client.

Syntax

EsbGetObjectInfo (hCtx, ObjType, AppName, DbName, ObjName, ObjInfo)

ByVal hCtx As Long
ByVal ObjType As Long
ByVal AppName As String
ByVal DbName As String
ByVal objName As String
   ObjInfo As ESB_OBJINFO_T

Parameter Description

hCtx VB API context handle. Can be local context handle returned by EsbCreateLocalContext().

ObjType Object type (must be single type). Refer to "Bitmask Types" for a list of possible values.

AppName Application name.

DbName Database name. If an empty string, uses the Application sub-directory.

ObjName Object name.

ObjInfo Buffer to receive object info structure.

Return Value

If successful, returns an object structure containing information about the appropriate object in pObject.

Access

This function requires the caller to have the appropriate level of access to the specified application and/or database containing the object (depending on the object type).
Example

Declare Function EsbGetObjectInfo Lib "ESBAPIN" (ByVal hCtx As Long, ByVal ObjType As Integer, ByVal AppName As String, ByVal DbName As String, ByVal ObjName As String, ObjInfo As ESB_OBJINFO_T) As Long

Sub ESB_GetObjectInfo ()
    Dim sts As Long
    Dim AppName As String
    Dim DbName As String
    Dim ObjName As String
    Dim ObjType As Integer
    Dim Object As ESB_OBJINFO_T
    AppName = "Sample"
    DbName = "Basic"
    ObjName = "Basic"
    ObjType = ESB_OBJTYPE_OUTLINE

    ' Get Object info structure
    sts = EsbGetObjectInfo (hCtx, ObjType, AppName, DbName, ObjName, Object)
End Sub

See Also

- EsbGetObject
- EsbListObjects
- EsbCreateLocalContext

EsbGetProcessState

Gets the current state of an asynchronous process, such as a calculate or a data import.

Syntax

EsbGetProcessState (hCtx, ProcState)
ByVal hCtx As Long
ProcState As ESB_PROCSTATE_T

Parameter Description

hCtx    VB API context handle.
ProcState   Pointer to process state structure

Notes

- Your program should call this function at regular intervals (between 5 & 10 seconds) until it returns ESB_STATE_DONE in ProcState.
- Calling this function except after initiating a successful asynchronous database operation, for example, a calculation, generates an error.

Return Value

If successful, returns the current process state in the state structure ProcState. Values for ProcState:
• ESB_STATE_DONE: 0 = Done
• ESB_STATE_INPROGRESS: 1 = In progress
• ESB_STATE_FINALSTAGE: 5 = In final stage; cannot be canceled

Access
This function requires no special privilege.

Example
Declare Function EsbGetProcessState Lib "ESBAPIN" (ByVal hCtx As Long, ProcState As ESB_PROCSTATE_T) As Long

See the examples for EsbBeginCalc, EsbCalc, and EsbImport.

See Also
• EsbBeginCalc
• EsbCalc
• EsbCancelProcess
• EsbImport

EsbGetString
Gets a string of data from the active database.

Syntax
EsbGetString (hCtx, getString, szString)
ByVal hCtx As Long
ByVal getString As String
ByVal szString As Integer

Parameter Description
hCtx VB API context handle.
getString Buffer to receive a returned data string. Should be at least as large as the largest string generated by the server. Maximum string size: 256 KB.
szString Size of the buffer to receive a returned data string.

Notes
• This function should be called after EsbReport(), EsbEndReport(), or EsbQueryDatabaseMember() if data is being returned.
• This function must be called repeatedly until it returns an empty string, which indicates that there is no more data to be returned.
• Always include the carriage return/line feed with this command or errors will result.
• Calling this function other than after successfully executing a report generates an error.
• The returned string will be less than 64 KB long.
• If getString length is greater than buffer size, the string is truncated.
If the returned string is less than 256 bytes the remaining empty bytes are filled with NULL characters.

**Return Value**

A data string is returned in `getString`. The function returns an empty string buffer if there is no more data to be returned.

**Access**

This function requires no special privileges.

**Example**

Declare Function EsbGetString Lib "ESBAPIN" (ByVal hCtx As Long, ByVal(getString As String, ByVal szString As Integer) As Long

See the examples for [EsbReport](#) and [EsbQueryDatabaseMembers](#).

**See Also**

- [EsbGetStringBuf](#)
- [EsbReport](#)
- [EsbEndReport](#)
- [EsbQueryDatabaseMembers](#)

---

**EsbGetStringBuf**

Gets data from the active database until it returns all available data or until the caller’s buffer is full.

**Syntax**

```
EsbGetStringBuf (hCtx,(getString, szString)
ByVal hCtx As Long
ByVal getString As String
ByVal szString As Integer
```

**Parameter Description**

- **hCtx** VB API context handle.
- **getString** Buffer to receive returned data strings. Maximum buffer size: 64 K.
- **szString** Size of buffer to receive returned data string.

**Notes**

- Call this function after [EsbReport()](#), [EsbEndReport()](#), or [EsbQueryDatabaseMember()](#) if data is returned.
- Calling this function other than after successfully executing a report generates an error.
- This function differs from [EsbGetString()](#) in that [EsbGetString()](#) returns data one line at a time. For larger data sets use [EsbGetString()](#).
**Return Value**

This function returns one or more data strings in `getString`. If no more data is left, it returns an empty string buffer.

This function returns all the data the buffer can accommodate, even if this means taking part of a record and including this partial record at the end of the buffer. The next call to `EsbGetStringBuf()` returns the rest of the partial record at the start of the buffer.

**Access**

This function requires no special privileges.

**Example**

```vbnet
Declare Function EsbGetStringBuf Lib "ESBAPIN" (  
    ByVal hCtx As Long,  
    ByVal(getString As String,  
    ByVal szString As Integer)  
    As Long

See the examples for `EsbReport` and `EsbQueryDatabaseMembers`.

See Also
- `EsbGetString`
- `EsbReport`
- `EsbEndReport`
- `EsbQueryDatabaseMembers`
```

**EsbGetUser**

Gets a user information structure, which contains security information for the user.

**Syntax**

```vbnet
EsbGetUser (hCtx, userName, pUserInfo)
ByVal hCtx As Long
ByVal userName As String
     pUserInfo As ESB_PUSERINFO_T
```

**Parameter Description**

- `hCtx` VB API context handle.
- `userName` User name.
- `pUserInfo` Buffer to receive a user info structure.

**Return Value**

If successful, returns the user information structure in `pUserInfo`. 
Access

This function requires the caller to have Create/Delete User privilege (ESB_PRIV_USERCREATE) for the logged in server, unless they are getting their own user information.

Example

Declare Function EsbGetUser Lib "ESBAPIN" (ByVal hCtx As Long, ByVal User As String, UserInfo As ESB_USERINFO_T) As Long

Sub ESB_GetUser()
    Dim sts As long
    Dim User As String
    Dim UserInfo As ESB_USERINFO_T
    User = "Joseph"    '************************
    ' Get User Info structure
    '************************
    sts = EsbGetUser (hCtx, User, UserInfo)
End Sub

See Also

- EsbGetApplicationAccess
- EsbListUsers
- EsbSetUser

**EsbGetUserEx**

Gets the user information structure containing security information for the user.

Syntax

Declare Function EsbGetUserEx Lib "esbapi" (ByVal hCtx As Long, ByVal userName As String, pUserInfo As ESB_USERINFOEX_T) As Long

Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>API context handle.</td>
</tr>
<tr>
<td>UserName</td>
<td>User name.</td>
</tr>
<tr>
<td>pUserInfoEx</td>
<td>Address of pointer to receive info structure of externally authenticated user.</td>
</tr>
</tbody>
</table>

Notes

This function operates similarly to **EsbGetUser**. The difference is that this function returns the extended user information structure ESB_USERINFOEX_T.

Return Value

If successful, returns the user information structure in pUserInfo.
Access

This function requires callers to have Create/Delete User privilege (ESS_PRIV_USERCREATE) for the logged in server, unless they are getting their own user information.

**EsbGetVariable**

Retrieves the value of a substitution variable.

**Syntax**

```plaintext
EsbGetVariable (hCtx, pVariable)
ByVal hCtx    As Long
    pVariable As ESB_PVARIABLE_T
```

**Parameter Description**

- **hCtx**  
  Context handle to the API.

- **pVariable**  
  The pointer to the structure containing the description of the specified substitution variable.

**Return Value**

If successful, **EsbGetVariable()** returns the value of the substitution variable in the *VarValue* field of structure *ESB_VARIABLE_T*.

**Example**

```visualbasic
Declare Function EsbGetVariable Lib "esbapin" (ByVal hCtx As Long, pVariable As ESB_VARIABLE_T) As Long
Sub Esb_GetVariable ()
    Dim sts As Long
    Dim oVariable As ESB_VARIABLE_T

    ' Get value of "QuarterName" Substitution Variable at the Sample application level
    oVariable.Server = "Localhost"
    oVariable.AppName = "Sample"
    ' ** Note that DbName has been left empty
    oVariable.VarName = "QuarterName"

    sts = EsbGetVariable(hCtx, oVariable)

    MsgBox oVariable.VarValue
End Sub
```

**See Also**

- “*ESB_VARIABLE_T*” on page 1196
- **EsbCreateVariable**
- **EsbDeleteVariable**
- **EsbListVariables**
EsbGetVersion

Gets the full version number of the connected Essbase Server, in the form *Release.Version.Revision*, for example, 3.0.0.

Syntax

```
EsbGetVersion (hCtx, Release, Version, Revision)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>VB API context handle.</td>
</tr>
<tr>
<td>Release</td>
<td>Address of variable to receive release number.</td>
</tr>
<tr>
<td>Version</td>
<td>Address of variable to receive version number.</td>
</tr>
<tr>
<td>Revision</td>
<td>Address of variable to receive revision number.</td>
</tr>
</tbody>
</table>

Notes

You can call this function after connecting to a server, to ensure that the Essbase Server version supports all the features used by your program.

Return Value

If successful, returns the full incremental Essbase Server version number in `pRelease`, `pVersion` and `pRevision`.

Access

This function requires no special privileges.

Example

```
Declare Function EsbGetVersion Lib "ESBAPIN" (ByVal hCtx As Long, Release As Integer, Version As Integer, Revision As Integer) As Long

Sub ESB_GetVersion ()
    Dim sts As Long
    Dim Release As Integer
    Dim Version As Integer
    Dim Revision As Integer   '************
    ' Get Version
    '************
    sts = EsbGetVersion (hCtx, Release, Version, Revision)
End Sub
```

EsbImport

Allows importing data from different sources to the Essbase Server.
Syntax

**EsbImport** (hCtx, pRules, pData, User, ErrName, isAbortOnError)

**Parameter**  **Description**

hCtx    VB API context handle.
pRules   Pointer to the rules file object definition structure.
pData    Pointer to the data file object definition structure.
User     Pointer to the SQL user structure (if data source is a SQL database). A NULL SQL user structure indicates a non SQL data source.
ErrName   Name of the error output file to be created locally.
isAbortOnError  If TRUE import stops on the first error otherwise continues.

Notes

- For a non SQL source, if **AppName** and **DbName** fields in ESB_OBJDEF_T structures for the pRules and pData are empty strings, hCtx must be a local context handle, and the ESB_OBJDEF_T FileName field must contain the fully qualified path to the file.
- If a local object is used, EsbCreateLocalContext() must be called first.

Return Value

None.

Access

This function requires the caller to have database design privilege for the specified database (ESB_PRIV_DBDESIGN).

Example

```
Declare Function EsbImport Lib "ESBAPIN" (By Val hCtx As Long, Rules As ESB_OBJDEF_T, Data As ESB_OBJDEF_T, User As ESB_MBRUSER_T, By Val ErrName As String, By Val AbortOnError As Integer) As Long

Sub ESB_Import ()
    Dim st s As Long
    Dim Rules As ESB_OBJDEF_T
    Dim Data As ESB_OBJDEF_T
    Dim User As ESB_MBRUSER_T
    Dim ErrorName As String
    Dim AbortOnError As Integer
    ' Rules file resides at the server
    '*****************
    Rules.hCtx = hCtx
```
Rules.Type     = ESB_OBJTYPE_RULES
Rules.AppName  = "Demo"
Rules.DbName   = "Basic"
Rules.FileName = "Test"

' **************************************************************
' Data file resides at the server
' **************************************************************
Data.hCtx      = hCtx
Data.Type      = ESB_OBJTYPE_TEXT
Data.AppName   = "Demo"
Data.DbName    = "Basic"
Data.FileName  = "Data"               ' **************************************************************
' Specify file to redirect errors
' to if any
' **************************************************************
ErrorName      = "IMPORT.ERR"   ' **************************************************************
' Abort on the first error
' **************************************************************
AbortOnError   = ESB_YES   '******
' Import
'******
sts            = EsbImport (hCtx, Rules, Data, User, ErrorName, AbortOnError)
' **************************************************************
' When a SQL data source is defined in the rules file, define
' the variables in the ESB_OBJDEF_T Data structure as follows:
'    Data.hCtx     = hCtx
'    Data.AppName  = ""
'    Data.DbName   = ""
'    Data.ObjType  = ESB_OBJTYPE_NONE
'    Data.FileName = ""
'
' Also, provide strings for the variables in the ESB_MBRUSER_T
' User structure; for example:
'    User.User     = "Dbusernm"
'    User.Password = "Dbpasswd"
'
' Use a blank string for User and Password, if the SQL source
' does not require user and password information; for example:
'
'    User.User     = ""
'    User.Password = ""
'
' Call the function as follows:
'    sts            = EsbImport (hCtx, Rules, Data, User, AbortOnError)
'
' ************************************************************** End Sub

See Also
- EsbExport
- EsbBuildDimension
**EsbInit**

Initializes the VB API and message database. The ESB_INIT_T structure passed to this function includes a number of initialization parameters, including the name of the message database, the flag indicating whether to use a customized error handler, the maximum message stack size to be used by this error handler, the name and location of your help file, and version number.

**Syntax**

```vbnet
EsbInit (pInit, phInst)
```

- **pInit** As ESB_INIT_T
- **phInst** As Long

**Parameter Description**

- **pInit** Pointer to VB API initialization structure.
- **phInst** Pointer to VB API instance handle.

**Notes**

- You must call this function before any other VB API functions.
- If any field in the initialization structure is an empty string or zero (as appropriate), the API uses a default value for those parameters.
- ESB_TRUE and ESB_FALSE are global variables. Assign them integer values as demonstrated in the example.

**Return Value**

The ESB_INIT_T structure passed to this function includes a number of initialization parameters, including the name of the message database, error handler, the name and location of your help file, and version number.

**Example**

```vbnet
Declare Function EsbInit Lib "ESBAPIN.DLL" (Init As ESB_INIT_T, hInst As Long) As Long

Sub ESB_Init()
    Dim hInst As Long
    Dim Init As ESB_INIT_T
    Dim sts As Long  ESB_FALSE = 0
    ESB_TRUE = 1  '***************

    ' Define init structure
    '***********************
    Init.Version = ESB_API_VERSION
    Init.MaxHandles = 10
    Init.LocalPath = "C:\ESSBASE"
```
' Use default message file
Init.MessageFile ="
' Use EsbGetMessage to retrieve
' messages
Init.ClientError = ESB_TRUE
Init.ErrorStack = 100               '*******************
' Initialize the API
'*******************
sts = EsbInit (Init, hInst)
End Sub

See Also
- EsbLogin
- EsbAutoLogin
- EsbTerm
- EsbGetMessage

**EsbKillRequest**

Terminates specific user sessions or requests.

**Syntax**

```vbnet
EsbKillRequest (hCtx, ReqInfo)
Val hCtx     As Long
Val pReqInfo  As ESB_REQUESTINFO_T
```

**Parameter Description**

- **hCtx**  Context handle
- **pReqInfo**  Pointer to the Request Information structure.

**Notes**

- *EsbKillRequest()* uses the information in ESB_REQUESTINFO_T regarding current
  sessions and requests to terminate a specific user session. This function can also be used to
  terminate (without logging out the user) any active requests being made to an application,
  a database, or the system during a user session.
- A session is the time in seconds between a user’s login and logout.
- A request is a query sent to Essbase Server by a user or by another process; for example,
  starting an application or restructuring a database outline. Each session can process only
  one request at a time; therefore, sessions and requests have a one-to-one relationship.
- This function terminates the sessions and requests specified by the UserName, AppName,
  and DbName in the ESB_REQUESTINFO_T structure. If those fields are null, this function
  terminates all sessions and requests initiated by this process (user). The application program
  is responsible for allocating and freeing the memory used by ESB_REQUESTINFO_T.
**Return Value**

If successful, returns a count of the number of users in Items, and generates a list of users with access to the specified application and database that is accessible using `EsbGetNextItem()`.

**Access**

This function requires no special privileges.

**Example**

Declare Function `EsbKillRequest` Lib "ESAPINW" (ByVal hCtx As Long, pReqInfo As ESB_REQUESTINFO_T) As Long

Sub ESB_KillRequest()
    Dim Items As Integer
    Dim ReqInfo As ESB_REQUESTINFO_T
    Dim sts As Long
    Dim pAccess As Integer

    '************
    ' List Requests
    '************
    'sts = EsbSetActive(hCtx, AppName, DbName, pAccess)
    'Debug.Print "EsbSetActive = " & sts
    'sts = EsbDefaultCalc(hCtx)
    'Debug.Print "EsbDefaultCalc = " & sts
    sts = EsbListRequests(hCtx, UserName, AppName, DbName, Items)
    Debug.Print "EsbListRequests = " & sts & " " & Items
    For n = 1 To Items
        '********************
        ' Get next Request Info
        ' from the list
        '********************
        sts = EsbGetNextItem(hCtx, ESB_REQUESTINFO_TYPE, ReqInfo)
        Debug.Print "EsbGetNextItem = " & sts & " " & ReqInfo.LoginId & " " & ReqInfo.DbRequestCode
        sts = EsbKillRequest(hCtx, ReqInfo)
        Debug.Print "EsbKillRequest = " & sts
    Next
End Sub

**See Also**

- `EsbListRequests`

**EsbListAliases**

Lists all the alias table names in the active database.

**Syntax**

`EsbListAliases (hCtx, pItems)`

ByVal `hCtx` As Long

`pItems` As Integer
### Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>VB API context handle.</td>
</tr>
<tr>
<td>pItems</td>
<td>Address of variable to receive Items of alias tables.</td>
</tr>
</tbody>
</table>

**Return Value**

If successful, this function returns a Items of alias tables in `pItems`, and generates an array of alias table names accessible via `EsbGetNextItem()`.

**Access**

This function requires callers to have access to the database, and to have selected it as their active database using `EsbSetActive()`.

**Example**

```vbnet
Declare Function EsbListAliases Lib "ESBAPIN" (ByVal hCtx As Long, Items As Integer) As Long

Sub ESB_ListAliases ()
    Dim Items As Integer
    Dim AliasName As String * ESB_ALIASNAMELEN
    Dim sts As Long
    ' *************
    ' List Aliases
    ' *************
    sts = EsbListAliases (hCtx, Items)   For n = 1 To Items
    ' ****************
    ' Get next Alias Name
    ' from the list
    ' ****************
    sts = EsbGetNextItem (hCtx, ESB_ALIASNAME_TYPE, ByVal AliasName)
    Next
End Sub
```

**See Also**

- `EsbDisplayAlias`
- `EsbGetNextItem`

---

### EsbListApplications

Lists all applications which are accessible to the caller.

**Syntax**

```
EsbListApplications (hCtx, pItems)
```

**Parameter Description**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>VB API context handle.</td>
</tr>
</tbody>
</table>
**Parameter Description**

pItems Address of variable to receive Items of returned applications.

**Return Value**

If successful, this function returns a Items of the number of accessible applications in pItems, and generates a list of application name strings accessible via EsbGetNextItem(). There are 'Items' number of items in the list.

**Access**

This function requires no special privileges; note however that server applications will only be listed if the caller has access to them.

**Example**

Declare Function EsbListApplications Lib "ESBAPIN" (ByVal hCtx As Long, Items As Integer) As Long

Sub ESB_ListApplications ()
    Dim sts As Long
    Dim Items As Integer
    Dim AppName As String * ESB_APPNAMELEN   '****************************
    ' Get List of Application names
    '******************************
    sts = EsbListApplications (hCtx, Items)   For n = 1 To Items
    '*********************************
    ' Get next Application name string
    '*********************************
    sts = EsbGetNextItem (hCtx,
        ESB_APPNAME_TYPE, ByVal AppName)
    Next
End Sub

**See Also**

- EsbListDatabases
- EsbListObjects
- EsbGetNextItem

### EsbListCalcFunctions

Lists all calculator functions available in the active application. The list of available functions includes all native functions and all custom-defined functions (CDFs) and custom-defined macros (CDMs).

**Syntax**

Declare Function EsbListCalcFunctions Lib "esbapin" ( ByVal hCtx As Long, ByVal CalcString As String, ByVal szString As Integer) As Long
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>API context handle.</td>
</tr>
<tr>
<td>CalcString</td>
<td>The string containing the available calculator functions. The string is in the form of XML.</td>
</tr>
<tr>
<td>szString</td>
<td>The size of the string containing the available calculator functions.</td>
</tr>
</tbody>
</table>

**Notes**

**EsbListCalcFunctions()** requires administrator privilege. The user must also have database access to receive this list. To avoid an error, the user must have both administrator privilege and access to the database to run a program with EsbListCalcFunctions().

The contents of the string returned by EsbGetCalcList is formatted as XML and must be either rendered in an XML utility or parsed to display only the actual text. All XML tags are enclosed in angle brackets (for example, `<xml_tag>`).

Here is a pared-down example of a typical XML output file:

```
ESSBASE API v.62000
1051034: Logging in user admin
1051035: Last login on Tuesday, May 22, 2001 10:31:19 AM
</list>
<group name="Boolean">
<function>
<name>&lt;![[CDATA[@ISACCTYPE]]]&gt;</name>
<syntax>&lt;![[CDATA[@ISACCTYPE(tag)]]]&gt;</syntax>
</function>
</group>
</list>
```

**Return Value**

Returns zero if successful; error code if unsuccessful.

**EsbListConnections**

Lists all users who are connected to the currently logged in server or application.
Syntax

EsbListConnections (hCtx, pItems)
ByVal hCtx As Long
    pItems As Integer

Parameter Description

hCtx    VB API context handle.
pItems  Variable to receive Items of users.

Notes

- If hCtx is an Administrator, pItems contains the number of users logged in to the server. If hCtx is an Application Manager, pItems contains the number users connected to any application for which hCtx is an Application Manager.
- Call EsbGetNextItem() once for each user (returned in the pItems variable). Each call to EsbGetNextItem() returns the user information in a ESB_USERINFO_T structure.

Return Value

Returns 0 if successful.

Access

This function requires the caller to have Administrator or Application Manager privilege.

Example

Declare Function EsbListConnections Lib "ESBAPIN" (ByVal hCtx As Long, Items As Integer) As Long
Sub ESB_ListConnections()
    Dim Items As Integer
    Dim UserInfo As ESB_USERINFO_T
    Dim sts As Long
    '*****************
    ' List Connections
    '*****************
    sts = EsbListConnections(hCtx, Items)
    For n = 1 To Items
        '****************************
        ' Get next User Info structure
        ' from the list
        '****************************
        sts = EsbGetNextItem(hCtx,
            ESB_USERINFO_TYPE, UserInfo)
    Next
End Sub

See Also

- EsbListLocks
- EsbListUsers
- EsbGetNextItem
EsbListCurrencyDatabases

Lists all currency databases within a specific application which are accessible to the caller.

Syntax

\[
\text{EsbListCurrencyDatabases} \ (hCtx, \ AppName, \ pItems)
\]

ByVal hCtx As Long

ByVal AppName As String

    pItems As Integer

Parameter Description

hCtx VB API context handle.

AppName Application name.

pItems Address of variable to receive Items of currency databases.

Notes

This function can only be used to list currency databases within an application on the server, not the client.

Return Value

If successful, this function returns a Items of the number of accessible currency databases in pItems, and generates a list of applications/currency database names that is accessible via EsbGetNextItem().

Access

This function requires no special privileges; note however that server currency databases will only be listed if the caller has access to them.

Example

Declare Function EsbListCurrencyDatabases Lib "ESBAPIN" (ByVal hCtx As Long, ByVal AppName As String, Items As Integer) As Long

Sub ESB_ListCurrencyDatabases ()
    Dim Items As Integer
    Dim AppName As String
    Dim AppDb As ESB_APPDB_T
    Dim sts As Long   AppName = "Sample"   '************************
    ' List Currency Databases
    ***************
    sts = EsbListCurrencyDatabases (hCtx, AppName,
        Items)   For n = 1 to Items   '************************
        ' Get next Application/Database
        ' item from the list
    '***********************
    sts = EsbGetNextItem (hCtx,
        ESB_CAPPDB_TYPE, AppDb)
    Next
End Sub
See Also

- EsbGetDatabaseInfo
- EsbGetDatabaseState
- EsbListApplications
- EsbListDatabases
- EsbListObjects
- EsbGetNextItem

### EsbListDatabases

Lists all databases which are accessible to the caller, either within a specific application, or on an entire server.

**Syntax**

```vba
EsbListDatabases (hCtx, AppName, pItems)
```

**Parameters**

- **hCtx** As Long: VB API context handle.
- **AppName** As String: Application name.
- **pItems** As Integer: Address of variable to receive count of applications and databases.

**Notes**

If the AppName argument is an empty string, this function lists all the accessible applications and databases on the server.

**Return Value**

If successful, this function returns a count of the number of accessible databases in `pCount`, and generates a list of the application and database names that is accessible via `EsbGetNextItem()`.

**Access**

This function requires no special privileges; note however that server databases will only be listed if the caller has access to them.

**Example**

```vba
Declare Function EsbListDatabases Lib "ESBAPIN" (ByVal hCtx As Long, ByVal AppName As String, Count As Integer) As Long

Sub ESB_ListDatabases ()
    Dim pItems As Integer
    Dim AppName As String
    Dim AppDb As ESB_APPDB_T
    Dim sts As Long   AppName = "Sample"   '**************
    ' List Databases
```
sts = EsbListDatabases (hCtx, AppName, pItems)   For n = 1 To pItems
' Get next Application/Database
' item from the list
sts = EsbGetNextItem (hCtx, ESB_APPDB_TYPE, AppDb)
Next
End Sub

See Also
- EsbGetDatabaseInfo
- EsbGetDatabaseState
- EsbListApplications
- EsbListCurrencyDatabases
- EsbListObjects
- EsbGetNextItem

**EsbListDbFiles**

Retrieves information on specified index and data files.

**Syntax**

```vba
EsbListDbFiles (hCtx, AppName, DbName, FileType, Items)
```

ByVal hCtx      As Long
ByVal AppName   As String
ByVal DbName    As String
ByVal FileType  As Integer
                   Items    As Long

**Parameter Description**

- **hCtx**  Context handle
- **AppName**  Application name
- **DbName**  Database name
- **FileType**  One of the following file types to be returned:
  - ESB_FILETYPE_INDEX
  - ESB_FILETYPE_DATA
  - ESB_FILETYPE_INDEX | ESB_FILETYPE_DATA
- **Items**  Number of index and data files returned

**Notes**

After you call **EsbListDbFiles()**, call **EsbGetNextItem()**, using ESB_DBFILEINFO_TYPE, to retrieve the database file information structure(s) that you want.
Return Value

- If successful, `EsbListDbFiles()` returns 0
- `Items` contains the number of index or data files returned
- A list of `ESB_DBFILEINFO_T` structures is created. Each structure has information on one of the index or data files returned.

Example

```vbnet
Dim OutDbInfo As ESB_DBFILEINFO_T
Dim FileType As Integer
Dim Count As Long

FileType = ESB_FILETYPE_INDEX + ESB_FILETYPE_DATA
sts = EsbListDbFiles(hCtx, "sample", "basic", FileType, Count)
MsgBox (sts)
If Not sts Then
    For Index = 1 To Count
        sts = EsbGetNextItem(hCtx, ESB_DBFILEINFO_TYPE, OutDbInfo)
        Next
    End If
```

See Also

- “ESB_DBFILEINFO_T” on page 1170

EsbListDrillThruURLs

Lists the drill-through URLs within the active database outline.

“Drill-through URL Limits” on page 1729.

Syntax

```vbnet
Declare Function EsbListDrillThruURLs Lib "esbapin" (ByVal hCtx As Long, ByRef URLNames As Variant) As Long
```

Parameter  Description

- hCtx  Visual Basic API context handle
- URLNames  List of URL names

Return Value

- If successful, lists names of drill-through URLs in the active database outline.
- If unsuccessful, returns an error code.

Access

- Caller must have database Read privilege (ESB_PRIV_READ) for the specified database.
- Caller must have selected the specified database as their active database using `EsbSetActive()`.
**Example**

Sub ESB_ListGLDrillThru()
    Dim intX As Integer
    Dim URLNames As Variant
    sts = EsbListDrillThruURLs(hCtx, URLNames)
    If sts = 0 Then
        Debug.Print "EsbListDrillThruURLs sts: " & sts
        For intX = LBound(URLNames) To UBound(URLNames)
            Debug.Print "URL Name: " & URLNames(intX)
        Next
    End If
End Sub

See also an extended example in "Drill-through Visual Basic API Example" on page 1132.

**EsbListFilters**

Lists all filters for a database.

**Syntax**

```
EsbListFilters (hCtx, AppName, DbName, pItems)
```

ByVal hCtx As Long
ByVal AppName As String
ByVal DbName As String
    pItems As Integer

**Parameter Description**

- **hCtx**: VB API context handle.
- **AppName**: Application name.
- **DbName**: Database name.
- **pItems**: Address of variable to receive Items of filter names.

**Return Value**

If successful, returns the Items of filters in the database in `pItems`, and generates an array of filter name strings accessible via `EsbGetNextItem()`.

**Access**

This function requires the caller to have Database Manager privilege (ESB_PRIV_DBDESIGN) for the specified database.

**Example**

```
Declare Function EsbListFilters Lib "ESBAPIN" (ByVal hCtx As Long, ByVal AppName As String, ByVal DbName As String, Items As Integer) As Long
```
Sub ESB_ListFilters ()
    Dim Items As Integer
    Dim AppName As String
    Dim DbName As String
    Dim FilterName As String * ESB_FTRNAMELEN
    Dim sts As Long   AppName = "Sample"
    DbName = "Basic"   ' *************
    ' List Filters
    ' *************
    sts = EsbListFilters (hCtx, AppName, DbName, Items)   For n = 1 To Items
    ' Get next Filter Name String
    ' from the list
    ' *************
    sts = EsbGetNextItem (hCtx,
    ESB_FTRNAME_TYPE, ByVal FilterName)
    Next
End Sub

See Also
- EsbGetFilter
- EsbSetFilter
- EsbGetNextItem

**EsbListGroups**

Lists all groups who have access to a particular Essbase Server, application or database.

**Syntax**

EsbListGroups (hCtx, AppName, DbName, pItems)

ByVal hCtx As Long
ByVal AppName As String
ByVal DbName As String
    pItems As Integer

**Parameter Description**

hCtx VB API context handle.

AppName Application name. If an empty string, lists all groups.

DbName Database name. If an empty string, lists groups for all databases within application.

pItems Address of variable to receive Items of groups.

**Notes**

If both *AppName* and *DbName* are not an empty string, only groups with access to the specified application and database will be listed. If *DbName* is an empty string, only groups with access to the specified application will be listed. If *AppName* is an empty string, all groups on the logged in server will be listed.
Return Value

If successful, returns a list of the number of groups in `pItems`, and generates a list of groups with access to the specified application and database accessible via `EsbGetNextItem()`.

Access

This function requires no special privileges.

Example

Declare Function EsbListGroups Lib "ESBAPIN" (ByVal hCtx As Long, ByVal AppName As String, ByVal DbName As String, Items As Integer) As Long

Sub ESB_ListGroups ()
    Dim Items As Integer
    Dim AppName As String
    Dim DbName As String
    Dim GroupInfo As ESB_USERINFO_T
    Dim sts As Long
    AppName = "Sample"
    DbName = "Basic"
    ' List Groups
    '************
    sts = EsbListGroups (hCtx, AppName, DbName,
                       Items) For n = 1 To Items '********************
    ' Get next Group structure
    ' from the list
    '************
    sts = EsbGetNextItem (hCtx,
                           ESB_GROUPINFO_TYPE, GroupInfo)
    Next
End Sub

See Also

- `EsbGetGroup`
- `EsbListUsers`
- `EsbGetNextItem`

**EsbListLocks**

Lists all users who are connected to a specific application and database, together with a list of data blocks which they currently have locked.

Syntax

```vbnet
EsbListLocks (hCtx, AppName, DbName, pItems)
```

Parameter Description

- `hCtx` VB API context handle.
Parameter | Description
--- | ---
AppName | Application name.
DbName | Database name.
pItems | Address of variable to receive Items of users.

Notes
This function is a "snapshot", in that only those users who are connected to the server when this function is called will be listed.

Return Value
If successful, returns a Items of the number of connected users in pItems, and generates a list of user lock structures accessible via EsbGetNextItem().

Access
This function requires the caller to have Database Design privilege (ESB_PRIV_DBDESIGN) for the specified database.

Example
Declare Function EsbListLocks Lib "ESBAPIN" (ByVal hCtx As Long, ByVal AppName As String, ByVal DbName As String, Items As Integer) As Long

Sub ESB_ListLocks ()
    Dim Items As Integer
    Dim AppName As String
    Dim DbName As String
    Dim LockInfo As ESB_LOCKINFO_T
    Dim sts As Long   AppName = "Sample"
    DbName = "Basic"   '***********
    ' List Locks
    '***********
    sts = EsbListLocks (hCtx, AppName, DbName,
        Items)   For n = 1 To Items   '***************************
        ' Get next user lock structure
        ' from the list
        '***************************
        sts = EsbGetNextItem (hCtx,
            ESB_LOCKINFO_TYPE, LockInfo)
    Next
End Sub

See Also
- EsbListConnections
- EsbListUsers
- EsbRemoveLocks
- EsbGetNextItem
**EsbListLogins**

Returns the list of login instances in the current session.

**Syntax**

```vba
Declare Function EsbListLogins Lib "esbapin" ((ByVal hCtx As Long, ByVal pItems As Integer) As Long
```

**Parameter Description**

- **hCtx**: API context handle.
- **pItems**: The number of logins in the login list returned from the server.

**Notes**

You can call `EsbListLogins()` more than once for the same user name and server. The API returns a unique context handle for each login to the specified server.

**Return Value**

If successful, returns login information and a count of current logins.

**Access**

Before calling this function, you must first initialize the API and obtain a valid instance handle by calling `EsbInit()`.

---

**EsbListObjects**

Lists all objects of the specified types on the server or locally on the client.

**Syntax**

```vba
EsbListObjects (hCtx, ObjType,AppName, DbName, pItems)
```

**Parameter Description**

- **hCtx**: VB API context handle. Can be local context handle returned from `EsbCreateLocalContext()`.
- **ObjType**: Object type (may be multiple types). Refer to "Bitmask Types" for a list of possible values.
- **AppName**: Application name.
- **DbName**: Database name. If an empty string, lists objects in the application sub-directory.
- **pItems**: Address of variable to receive the Items of objects of the appropriate type(s).
Return Value

If successful, returns a `Items` of the number of objects of the appropriate type(s) in `pItems`, and generates an array of matching object structures accessible via `EsbGetNextItem()`.

Access

This function requires no special privileges; note however that server objects will only be listed if the caller has the appropriate level of access to the application and/or database (depending on the object type).

Example

Declare Function EsbListObjects Lib "ESBAPIN" (ByVal hCtx As Long, ByVal ObjType As Integer, ByVal AppName As String, ByVal DbName As String, Items As Integer) As Long

Sub ESB_ListObjects ()
    Dim Items As Integer
    Dim AppName As String
    Dim DbName As String
    Dim ObjType As Integer
    Dim ObjInfo As ESB_OBJINFO_T
    Dim sts As Long   Appname = "Sample"
    DbName = "Basic"
    ObjType = ESB_OBJTYPE_OUTLINE   '*********************
    ' List Outline Objects
    '*********************
    sts = EsbListObjects (hCtx, ObjType,
                           AppName, DbName, Items)   For n = 1 To Items   '****************************
    ' Get next Object Structure
    ' from the list
    '****************************
    sts = EsbGetNextItem (hCtx,
                           ESB_OBJINFO_TYPE, ObjInfo)
    Next
End Sub

See Also

- `EsbGetObject`
- `EsbGetObjectInfo`
- `EsbGetNextItem`

**EsbListRequests**

Returns information about active sessions and requests.

Syntax

```
EsbListRequests (hCtx, UserName, AppName, DbName, Items)
ByVal hCtx As Long
ByVal UserName As String
ByVal AppName As String
ByVal DbName As String
ByVal Items As Long
```
### Parameter Description

- **hCtx**: Context handle  
- **AppName**: Application name  
- **DbName**: Database name  
- **UserName**: User name  
- **Items**: Number of index and data files returned

### Notes

- A session is the time in seconds between a user’s login and logout.
- A request is a query sent to Essbase by a user or by another process; for example, starting an application or restructuring a database outline. Each session can process only one request at a time; therefore, sessions and requests have a one-to-one relationship.
- Some of the listed requests may have been recently terminated, but are still listed as active due to network delay.
- This function returns information on requests and sessions initiated by the process specified by the UserName, AppName, and DbName. If these parameters are null or empty, then all the processes in the system are listed. This function returns the number of current requests and one ESB_REQUESTINFO_T structure for each request.
- After you call `EsbListRequests()`, call `EsbGetNextItem()`, using ESB_REQUESTINFO_TYPE, to retrieve the request information structures that you want.

### Return Value

If successful, returns a count of the number of users in Items, and generates a list of users with access to the specified application and database that is accessible using `EsbGetNextItem()`.

### Access

This function requires no special privileges.

### Example

Declare Function EsbListRequests Lib "ESBAPIN" (ByVal hCtx As Long, ByVal UserName As String, ByVal AppName As String, ByVal DbName As String, pItems As Integer) As Long

Sub ESB_ListRequests()
    Dim Items As Integer
    Dim ReqInfo As ESB_REQUESTINFO_T
    Dim sts As Long
    Dim pAccess As Integer

    'sts = EsbSetActive(hCtx, AppName, DbName, pAccess)
    'sts = EsbDefaultCalc(hCtx)
    '*************
    ' List Requests
    '*************
    sts = EsbListRequests(hCtx, UserName, AppName, DbName, Items)
    Debug.Print "EsbListRequests = " & sts & " & Items"
For n = 1 To Items
    '********************
    ' Get next Request Info
    ' from the list
    '********************
    sts = EsbGetNextItem(hCtx, ESB_REQUESTINFO_TYPE, ReqInfo)
    Debug.Print "EsbGetNextItem = " & sts & " " & ReqInfo.LoginId & " " & ReqInfo.DbRequestCode
    Next
End Sub

See Also
- EsbKillRequest

## EsbListUsers

Lists all users who have access to a particular Essbase Server, application or database.

### Syntax

```vbnet
EsbListUsers (hCtx,AppName,DbName,pItems)
```

### Parameter Description

- **hCtx**  
  VB API context handle

- **AppName**  
  Application name. If an empty string, lists all users

- **DbName**  
  Database name.

- **pItems**  
  Address of variable to receive count of users

### Notes

- If both `AppName` and `DbName` are not empty strings, only users with access to the specified application and database will be listed. If `DbName` is an empty string, only users with access to the specified application will be listed. If `AppName` is an empty string, all users that exist on the server will be listed.

- The list of users with access to the specified application and database--which is accessible via `EsbGetNextItem()`--is returned as a list of `ESB_USERINFO_T` structures. The `AppName` and `DbName` fields of these returned user info structures contain NULL values.

### Return Value

If successful, returns a count of the number of users in `pCount`, and generates a list of users with access to the specified application and database that is accessible via `EsbGetNextItem()`.
Access

This function requires no special privileges.

Example

Declare Function EsbListUsers Lib "ESBAPIN" (ByVal hCtx As Long, ByVal AppName As String, ByVal DbName As String, Count As Integer) As Long

Sub ESB_ListUsers ()
    Dim Count As Integer
    Dim AppName As String
    Dim DbName As String
    Dim UserInfo As ESB_USERINFO_T
    Dim sts As Long
    AppName = "Sample"
    DbName = "Basic"
    ' List Users
    sts = EsbListUsers (hCtx, AppName, DbName, Count)    For n = 1 To Count
        ' Get next User Info structure
        ' from the list
        sts = EsbGetNextItem (hCtx, ESB_USERINFO_TYPE, UserInfo)
    Next
End Sub

See Also

- EsbGetUser
- EsbListConnections
- EsbListGroups
- EsbListLocks
- EsbGetNextItem

### EsbListUsersEx

Lists all users who have access to a particular Essbase Server, application or database. This function operates similar to **EsbListUsers**, with the addition of the security protocol parameter.

**Syntax**

Declare Function EsbListUsersEx Lib "esbapin" (ByVal hCtx As Long, ByVal AppName As String, ByVal DbName As String, ByVal Protocol As String, pItems As Integer) As Long

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>The API context handle.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>AppName</td>
<td>The Application name. If NULL, lists all users.</td>
</tr>
<tr>
<td>DbName</td>
<td>The database name. If NULL, lists users for all databases within application.</td>
</tr>
<tr>
<td>Protocol</td>
<td>The name of the external authentication security protocol mechanism.</td>
</tr>
<tr>
<td>pItems</td>
<td>The count of users.</td>
</tr>
</tbody>
</table>

**Notes**

- If both *AppName* and *DbName* are not NULL, only users with access to the specified application and database are listed. If *DbName* is NULL, only users with access to the specified application are listed. If *AppName* is NULL, all users that exist on the server are listed.
- The list of users with access to the specified application and database is returned as a list of “ESB_USERINFOEX_T” on page 1194 structures. The *AppName* and *DbName* fields of these returned user info structures contain NULL values. The list of structures is accessed by calling `EsbGetNextItem`.

**Return Value**

If successful, returns a count of the number of users in `pCount`, and list of users with access to the specified application and database in `ppUserList`.

**Access**

This function requires no special privileges.

**EsbListVariables**

Lists all substitution variables that conform to the input criteria.

**Syntax**

```vbnet
EsbListVariables (hCtx, pVariable, pItems)
```

- `hCtx` As Long
- `pVariable` As ESB_PVARIABLE_T
- `pItems` As Integer

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>Context handle to the API.</td>
</tr>
</tbody>
</table>
Parameter | Description
--- | ---
pVariable | The pointer to the structure containing the description of the substitution variables being listed.
  - The members VarName and VarValue are ignored.
  - If the Server member is given but AppName and DbName are empty, then the function will list substitution variables at the server level only.
  - If the Server and AppName members are given but DbName is empty, it lists all variables at both the given server and application levels.
  - If all three of Server, AppName, and DbName members are given, it lists variables from all three specified levels.
  - If a field is empty, then that field is treated as a "don't care."
pItems | The pointer to an unsigned long value indicating the number of variables being returned in the ppVarList parameter.

Return Value
If successful returns zero (0).

Example
Declare Function EsbListVariables Lib "esbapin" (ByVal hCtx As Long, pVariable As ESB_VARIABLE_T, pItems As Integer) As Long
Public Sub ESB_ListVariables ()
  Dim i As Integer
  Dim nCount As Integer
  Dim sts As Long
  Dim oVariable As ESB_VARIABLE_T
  oVariable.AppName = "Sample"
  sts = EsbListVariables(hCtx, oVariable, nCount)
  If sts = 0 Then
    If nCount <> 0 Then
      For i = 1 To nCount
        sts = EsbGetNextItem(hCtx, ESB_VARIABLE_TYPE, oVariable)
        Debug.Print "Variable Name: " & oVariable.VarName
        Debug.Print "Value: " & oVariable.VarValue
        Debug.Print
      Next
    Else
      MsgBox "No substitution variables found."
    End If
  Else
    MsgBox "Error listing substitution variables."
  End If
End Sub

See Also
- "ESB_VARIABLE_T" on page 1196
- EsbCreateVariable
- EsbDeleteVariable
- EsbGetVariable
**EsbLoadAlias**

Creates and permanently loads an alias table for the active database from a structured text file.

**Syntax**

```vbnet
EsbLoadAlias (hCtx, AltName, FileName)
ByVal hCtx As Long
ByVal AltName As String
ByVal FileName As String
```

**Parameter Description**

- **hCtx**: VB API context handle.
- **AltName**: Name of an alias table to load.
- **FileName**: Full path name of a structured alias names file on the server.

**Notes**

- This function will not complete successfully if `AliasName` already exists. To load an alias table of the same name as an existing one, the existing alias table must first be deleted.
- The alias table file format is described in the *Oracle Essbase Database Administrator’s Guide*.

**Return Value**

None.

**Access**

This function requires the caller to have access to the database, and to have selected it as their active database using `EsbSetActive()`.

**Example**

```vbnet
Declare Function EsbLoadAlias Lib "ESBAPIN" (ByVal hCtx As Long, ByVal AliasName As String, ByVal FileName As String) As Long
Sub ESB_LoadAlias ()
    Dim sts As Long
    Dim AliasName As String
    Dim FileName As String
    AliasName = "TestAlias"
    FileName = "c:\essbase\test.alt"  '***********
    ' Load Alias
    '***********
    sts = EsbLoadAlias (hCtx, AliasName, FileName)
End Sub
```

**See Also**

- `EsbListAliases`
- `EsbRemoveAlias`
- `EsbSetActive`
**EsbLoadApplication**

Starts an application on the server.

**Syntax**

```
EsbLoadApplication (hCtx, AppName)
```

*Parameters*  
- **hCtx**: VB API context handle.  
- **AppName**: Name of an application to load.

**Notes**

To load an application, the connected user must have load access to the application.

**Return Value**

None.

**Access**

This function requires the caller to have Application Load/Unload privilege (ESB_PRIV_APPLOAD) for the specified application.

**Example**

```vbnet
Declare Function EsbLoadApplication Lib "ESBAPIN" (ByVal hCtx As Long, ByVal AppName As String) As Long

Sub ESB_LoadApplication ()  
    Dim sts As Long  
    Dim AppName As String  
    AppName = "Sample"  ; *****************  
    ' Load Application  ; *****************  
    sts = EsbLoadApplication (hCtx, AppName)  
End Sub
```

**See Also**

- [EsbLoadDatabase](#)
- [EsbUnloadApplication](#)

**EsbLoadDatabase**

Starts a database within an application on the server.

**Syntax**

```
EsbLoadDatabase (hCtx, AppName, DbName)
```

*Parameters*  
- **hCtx**: VB API context handle.  
- **AppName**: Name of an application.  
- **DbName**: Name of a database.
### EsbLoadDatabase

**Parameter**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>VB API context handle.</td>
</tr>
<tr>
<td>AppName</td>
<td>Application name.</td>
</tr>
<tr>
<td>DbName</td>
<td>Name of a database to load.</td>
</tr>
</tbody>
</table>

**Return Value**

None.

**Access**

This function requires the caller to have Database Load/Unload privilege (ESB_PRIV_APPLOAD).

**Example**

```vbnet
Declare Function EsbLoadDatabase Lib "ESBAPIN" (ByVal hCtx As Long, ByVal AppName As String, ByVal DbName As String) As Long

Sub ESB_LoadDatabase ()
    Dim sts As Long
    Dim AppName As String
    Dim DbName As String
    AppName = "Sample"
    DbName = "Basic"  '**************
    ' Load Database
    '**************
    sts = EsbLoadDatabase (hCtx, AppName, DbName)
End Sub
```

**See Also**

- EsbLoadApplication
- EsbUnloadDatabase

### EsbLockObject

Locks an object on the server or client object system to prevent other users from updating it.

**Syntax**

```vbnet
EsbLockObject (hCtx, ObjType, AppName, DbName, ObjName)
```

**Parameter**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>VB API context handle. May be local context handle returned by EsbCreateLocalContext().</td>
</tr>
<tr>
<td>ObjType</td>
<td>Object type (must be single type. Refer to Table 15 for a list of possible values.</td>
</tr>
</tbody>
</table>
**Parameter Description**

AppName  Application name.

DbName   Database name. If an empty string, uses the application sub-directory.

ObjName   Name of an object to lock.

**Notes**

- To lock an object, the object must already exist and not be locked by another user.
- This function does not retrieve the object. To retrieve the object, use EsbGetObject().

**Return Value**

None.

**Access**

This function requires the caller to have Application or Database Design privilege (ESB_PRIV_APPDESIGN or ESB_PRIV_DBDESIGN) for the specified application or database containing the object.

**Example**

Declare Function EsbLockObject Lib "ESBAPIN" (ByVal hCtx As Long, ByVal ObjType As Integer, ByVal AppName As String, ByVal DbName As String, ByVal ObjName As String) As Long

Sub ESB_LockObject ()
    Dim sts As Long
    Dim AppName As String
    Dim DbName As String
    Dim ObjName As String
    Dim ObjType As Integer   AppName = "Sample"
    DbName = "Basic"
    ObjName = "Basic"
    ObjType = ESB_OBJTYPE_OUTLINE      '***************
    ' Lock Rules Object
    '***************
    sts = EsbLockObject (hCtx, ObjType, AppName,
                         DbName, ObjName)
End Sub

**See Also**

- EsbGetObject
- EsbGetObjectInfo
- EsbListObjects
- EsbPutObject
- EsbUnlockObject
**EsbLogin**

Logs in a user to an Essbase Server. This function should normally be called after executing a successful call to EsbInit, and prior to making any other VB API calls which require a context handle argument.

**Syntax**

```vbnet
EsbLogin (hInst, Server, User, Password, pItems, hCtx)
```

**ByVal**

- `hInst` `As Long` VB API instance handle.
- `Server` `As String` Network server name string. Required field.
- `User` `As String` User name string. Required field.
- `Password` `As String` Password string. Required field.
- `pItems` `As Integer` Address of variable to receive Items of accessible applications/databases.
- `hCtx` `As Long` Pointer to an Essbase Server context handle.

**Parameter Description**

- **hInst**: VB API instance handle.
- **Server**: Network server name string. Required field. The server name can be expressed as `hostname, hostname:port`, or as a URL representing the APS servlet endpoint with the Essbase failover cluster name; for example:
  ```
  http://myhost:13080/aps/Essbase?clustername=Essbase-Cluster1
  ```
  For secure mode (SSL), the URL syntax is
  ```
  http[s]://host:port/aps/Essbase?
  ClusterName=logicalName&SecureMODE=yesORno
  ```
  For example,
  ```
  ```
- **User**: User name string. Required field.
- **Password**: Password string. Required field.
- **pItems**: Address of variable to receive Items of accessible applications/databases.
- **hCtx**: Pointer to an Essbase Server context handle.

**Notes**

- If you are programming in Microsoft Windows, you should consider using the EsbAutoLogin function instead of EsbLogin.
- You can call EsbLogin more than once for the same user name and server. The API returns a unique context handle for each login to the specified server.

**Return Value**

If successful, returns an Essbase Server context handle in `phCtx`, which can be used as an argument in subsequent calls to other API functions. Also returns a Items of databases accessible to the specified user in `pItems`, and generates a list of accessible applications and databases that can be read by calling EsbGetNextItem.
Access

Before calling this function, you must first initialize the API and obtain a valid instance handle by calling the EsbInit function.

Example

Declare Function EsbLogin Lib "ESBAPIN" (ByVal hInst As Long, ByVal Server As String, ByVal User As String, ByVal Password As String, Items As Integer, hCtx As Long) As Long

Sub ESB_Login ()
    Dim hInst As Long
    Dim Server As String * ESB_SVRNAMELEN
    Dim User As String * ESB_USERNAMELEN
    Dim Password As String * ESB_PASSWORDLEN
    Dim Items As Integer
    Dim AppDb As ESB_APPDB_T
    Dim hCtx As Long   '*****************
    ' Login to Essbase Server
    '*****************
    sts = EsbLogin (hInst, Server, User, Password, Items, hCtx)   For n = 1 To Items
    '******************************
    ' Get next Application/Database
    ' name combination from the list
    '******************************
    sts = EsbGetNextItem (hCtx, ESB_LAPPDB_TYPE,AppDb)
    Next
End Sub

See Also

- EsbAutoLogin
- EsbInit
- EsbListDatabases
- EsbLogout
- EsbSetActive
- EsbGetNextItem

EsbLoginSetPassword

Logs in a user, and changes the password. Use this function if the password expires, or must be changed at the next login.

Syntax

EsbLoginSetPassword(hInstance, Server, UserName, Password, _
    NewPassword, Items, hCtx)

ByVal hInstance As Long
ByVal Server As Long
ByVal UserName As String
ByVal Password As String
ByVal NewPassword As String
    Items As Integer
hCtx As Long
### Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>API instance handle.</td>
</tr>
<tr>
<td>Server</td>
<td>Network server name string. The server name can be expressed as <code>hostname, hostname:port</code>, or as a URL representing the APS servlet endpoint with the Essbase failover cluster name; for example: <a href="http://myhost:13080/aps/Essbase?clustername=Essbase-Cluster1">http://myhost:13080/aps/Essbase?clustername=Essbase-Cluster1</a> For secure mode (SSL), the URL syntax is http[s]://host:port/aps/Essbase?ClusterName=logicalName&amp;SecureMODE=yesORno For example, <a href="https://myhost:13080/aps/Essbase?clustername=Essbase-Cluster1&amp;SecureMODE=Yes">https://myhost:13080/aps/Essbase?clustername=Essbase-Cluster1&amp;SecureMODE=Yes</a></td>
</tr>
<tr>
<td>UserName</td>
<td>User name.</td>
</tr>
<tr>
<td>Password</td>
<td>Old password.</td>
</tr>
<tr>
<td>NewPassword</td>
<td>New password.</td>
</tr>
<tr>
<td>Items</td>
<td>Number of accessible databases.</td>
</tr>
<tr>
<td>hCtx</td>
<td>Essbase Server context handle.</td>
</tr>
</tbody>
</table>

### Notes
- Call `EsbLoginSetPassword` after you call `EsbLogin`, and after you receive status code 1051090 (Password has expired), or 1051093 (Change password now).
- In Microsoft Windows, consider using `EsbAutoLogin`, instead of `EsbLoginSetPassword`.
- Free memory allocated for `Items` using `EsbFree`.

### Return Value
If successful, `EsbLoginSetPassword`:
- Returns in `hCtx` the Essbase Server context handle.
- Returns in `Items` the number of databases accessible to the user.
- Generates a list of accessible databases, which can be read by calling `EsbGetNextItem`.

### Access
Before you call `EsbLoginSetPassword`, call `EsbInit` to initialize the API, and obtain a valid instance handle.

### Example
Declare Function `EsbLoginSetPassword` Lib "ESBAPIN" (ByVal hInst As Long, ByVal Server As String, ByVal User As String, ByVal Password As String, ByVal NewPassword As String, Items As Integer, hCtx As Long) As Long
Sub ESB_LoginSetPassword()
Dim hInst As Long
Dim Server As String * ESB_SVRNAMELEN
Dim User As String * ESB_USERNAMELEN
Dim Password As String * ESB_PASSWORDLEN
Dim NewPassword As String * ESB_PASSWORDLEN
Dim Items As Integer
Dim AppDb As ESB_APPDB_T
Dim hCtx As Long
sts = EsbLoginSetPassword(hInst, Server, User, Password, NewPassword, Items, hCtx)
For n = 1 To Items
' Get next Application/Database
' name combination from the list
'******************************************************************************
sts = EsbGetNextItem(hCtx, ESB_LAPPDB_TYPE, AppDb)
Next
End Sub

See Also
- EsbAutoLogin
- EsbInit
- EsbListDatabases
- EsbLogout
- EsbSetActive

EsbLogout
Logs a user out from an Essbase Server.

Syntax
EsbLogout (hCtx)
ByVal hCtx As Long

Parameter Description
hCtx VB API context handle to logout.

Notes
- This function logs out only the login represented by the specified context handle. No other logins or contexts are affected, even if using the same user name.
- This function should only be used for login contexts. For local contexts, use the EsbDeleteLocalContext() function.

Return Value
None.

Access
To call this function, the caller must have previously logged in successfully using either the EsbLogin() or EsbAutoLogin() functions.
Example

Declare Function EsbLogout Lib "ESBAPIN" (ByVal hCtx As Long) As Long

Sub ESB_Logout ()
    Dim sts As Long   '******
    ' Logout
    '******
    sts = EsbLogout (hCtx)
End Sub

See Also

- EsbAutoLogin
- EsbDeleteLocalContext
- EsbGetActive
- EsbLogin
- EsbLogoutUser

EsbLogoutUser

Allows an Administrator or an Application Manager to disconnect another user from an Essbase Server.

Syntax

EsbLogoutUser (hCtx, LoginId)
ByVal hCtx As Long
ByVal LoginId As Long

Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>VB API context handle of user forcing the log out.</td>
</tr>
<tr>
<td>LoginId</td>
<td>Login ID of user to be logged out.</td>
</tr>
</tbody>
</table>

Notes

- LoginId can be obtained from the user information structure returned by the EsbListConnections() function.
- This function logs out only the login represented by the specified LoginId. No other logins or contexts are affected.
- An Administrator can log out anyone logged in to the server to which hCtx is logged in. An Application Manager can log out only those users connected to an application for which hCtx is an Application Manager.
- You cannot log yourself out.

Return Value

None.
**Access**

To call this function, you must have Administrator or Application Manager privilege.

**Example**

```vba
Declare Function EsbLogoutUser Lib "ESBAPIN" (ByVal hCtx As Long, ByVal LoginId As Long) As Long

Sub ESB_LogoutUser()
    Dim Items As Integer
    Dim UserInfo As ESB_USERINFO_T
    Dim sts As Long
    '*****************
    ' List Connections
    '*****************
    sts = EsbListConnections(hCtx, Items)
    '*****************
    ' Log out all users
    '*****************
    For n = 1 To Items
        '*****************************
        ' Get next User Info structure
        ' from the list
        '*****************************
        sts = EsbGetNextItem(hCtx, ESB_USERINFO_TYPE, UserInfo)
        sts = EsbLogoutUser(hCtx, UserInfo.LoginId)
    Next
End Sub
```

**See Also**

- `EsbListConnections`
- `EsbLogout`

---

**EsbLogSize**

Returns the size of the Essbase Server log file (`essbase.log`), or of the application log file (`appname.log`).

**Syntax**

```vba
Declare Function EsbLogSize Lib "esbapin" ( ByVal hCtx As Long, ByVal isAgentLog As Integer, ByVal AppName As String, pulLogSize As Long) As Long
```

**Parameter Description**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>API context handle.</td>
</tr>
<tr>
<td>isAgentLog</td>
<td>If TRUE, the size of the Essbase Server log file (<code>essbase.log</code>) is returned. If FALSE, the size of the application log file (<code>appname.log</code>) is returned.</td>
</tr>
<tr>
<td>AppName</td>
<td>Application name.</td>
</tr>
</tbody>
</table>
### pulLogSize

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pulLogSize</td>
<td>Size of log file returned.</td>
</tr>
</tbody>
</table>

### Notes

- Use EsbGetLogFile() to view message logs.
- For the locations of `essbase.log` and `appname.log`, see the *Oracle Essbase Database Administrator's Guide*.

### Return Value

Returns a zero if successful.

### Access

This function does not require the caller to have access privileges.

## EsbLROAddObject

Links reporting objects to a data cell in an Essbase database.

### Syntax

```vbnet
EsbLROAddObject (hCtx, memCount, MemComb, usOption, pLRODesc)
ByVal hCtx As Long
ByVal memCount As Long
ByVal MemComb As String
ByVal usOption As Integer
    pLRODesc As ESB_LRODESC_API_T
```

### Parameter Description

- **hCtx**: API context handle.
- **memCount**: The number of members specified in `MemComb`.
- **MemComb**: String of member names (EOL, CR delimited) that define the data cell to be linked.
- **usOption**: Option specifying where to store the object. Use one of the following:
  - ESB_STORE_OBJECT_API stores an object on the server
  - ESB_NOSTORE_OBJECT_API to not store on server
- **pLRODesc**: Pointer to object's description structure, “ESB_LRODESC_API_T” on page 1156.

### Notes

- The linked object can be any of the following types:
  - A flat file, such as a Word document, Excel spreadsheet, or bitmap image.
  - A cell note containing up to 599 characters of text.
  - A link to a URL.
  - A link to another database (a linked partitions feature).
If you elect not to store the object on the server (*usOption*), only the link information is stored on the server, and your application is responsible for all file management tasks for the object.

- The *usOption* parameter is ignored for cell notes, which are always stored on the server.
- The *usOption* parameter for a URL linked object should always be `ESB_NOSTORE_OBJECT_API`.
- `EsbLROAddObject` uses the currently logged in user name as the "created by" user name for the object and ignores any user name specified in the `pLRODesc` object description structure.

**Return Value**

If successful, returns `ESB_STS_NOERR`. Otherwise, returns an error code.

**Access**

A call to this function requires write privileges (ESB_PRIV_WRITE) to the data cell or the active database.

**Example**

```vbnet
Declare Function EsbLROAddObject Lib "esbapin" _
(ByVal hCtx As Long, ByVal memCount As Long, _
ByVal memComb As String, ByVal usOption As Integer, _
pLRODesc As ESB_LRODESC_API_T) As Long

Public Sub ESB_LROAddObject()   Dim Desc As ESB_LRODESC_API_T
    Dim memCount As Long
    Dim memComb As String
    Dim opt As Integer
    Dim i As Integer

    Desc.userName = "Admin"
    Desc.ObjType = ESB_LROTYPE_CELLNOTE_API
    Desc.note = "Note from DFS"
    memCount = 5
    memComb = "Jan" & vbLf & "Sales" & vbLf & _
        "Cola" & vbLf & "Utah" & vbLf & _
        "Actual"   opt = ESB_NOSTORE_OBJECT_API

    sts = EsbLROAddObject(hCtx, memCount, memComb, _
        opt, Desc)
End Sub
```

**See Also**

- "`ESB_LROINFO_API_T` on page 1157"
- `EsbLROUpdateObject`
- `EsbLRODeleteObject`

**EsbLRODeleteCellObjects**

Deletes all objects linked to a given data cell in a database. To delete a specific object linked to a cell, use `EsbLRODeleteObject`. 
Syntax

EsbLRODeleteCellObjects (hCtx, memCount, memComb, PulCount)

Parameter Description

hCtx API context handle.

memCount Number of members specified in MemComb.

MemComb String of member names (EOL, CR delimited).

PulCount Number of LRO catalog entries deleted.

Notes

- This function deletes all objects linked to the specified cell along with their catalog entries.
- If the object is not stored on the server, only the cell link is destroyed.
- EsbLRODeleteCellObjects() generates a list of the objects it deletes. After calling this function, use EsbGetNextItem to retrieve information about each deleted object. The example code demonstrates how to do this.

Return Value

If successful, returns ESB_STS_NOERR. Otherwise, returns an error code.

Access

A call to this function requires write privileges (ESB_PRIV_WRITE) to the data cell or the active database.

Example

Declare Function EsbLRODeleteCellObjects Lib "esbapin" _
(ByVal hCtx As Long, ByVal memCount As Long, _
ByVal memComb As String, PulCount As Integer) As Long

Public Sub ESB_LRODeleteCellObjects()   Dim Desc As ESB_LRODESC_API_T
    Dim Items As Integer
    Dim memCount As Long
    Dim memComb As String
    Dim i As Integer

    memCount = 5
    memComb = "Jan" & vbCrLf & "Sales" & _
        "Cola" & vbCrLf & "Utah" & _
        "Actual"

    sts = EsbLRODeleteCellObjects(hCtx, memCount, _
        memComb, Items)

    If sts = 0 Then
        For i = 1 To Items
EsbLRODeleteObject

Deletes a specific object linked to a data cell in a database. To delete all objects linked to a cell, use EsbLRODeleteCellObjects.

Syntax

EsbLRODeleteObject (hCtx, pLinkId)
ByVal hCtx    As Long
    pLinkId As ESB_LROHANDLE_API_T

Parameter Description

hCtx    API context handle.

pLinkId    Pointer to object identification structure. The structure is returned by EsbLROAddObject through the “ESB_LRODESC_API_T” on page 1156 structure.

Notes

● The specified object is deleted and also removed from the Catalog list.
● If the object is not stored on the server, only the cell link is destroyed.

Return Value

If successful, returns ESB_STS_NOERR. Otherwise, returns an error code.

Access

A call to this function requires write privileges (ESB_PRIV_WRITE) to the data cell or the active database.

Example

Declare Function EsbLRODeleteObject Lib "esbapin" __
    (ByVal hCtx As Long, pLinkID As ESB_LROHANDLE_API_T) __
    As Long

Public Sub ESB_LRODeleteObject()
    Dim LinkID As ESB_LROHANDLE_API_T

    LinkID.hObject = 1
LinkID.cellKey.cellOffset = 0
LinkID.cellKey.blkOffset = 198
LinkID.cellKey.segment = 0

sts = EsbLRODeleteObject(hCtx, LinkID)End Sub

See Also

- “Constant and Structure Definitions for Linked Objects” on page 1155
- EsbLROAddObject
- EsbLRODeleteCellObjects
- EsbLROPurgeObjects

**EsbLROGetCatalog**

Retrieves a list of linked object catalog entries for a given data cell in a database.

**Syntax**

```plaintext
EsbLROGetCatalog (hCtx, memCount, memComb, PulCount)
ByVal hCtx As Long
ByVal memCount As Long
ByVal memComb As String
      PulCount As Integer
```

**Parameter**  **Description**

- **hCtx**  API context handle.
- **memCount**  Number of members specified in `memComb`.
- **memComb**  Array of member names.
- **PulCount**  Number of LRO catalog entries returned to caller.

**Notes**

To retrieve the catalog information, call `EsbGetNextItem` after calling this function. The example code demonstrates how to do this.

**Return Value**

If successful, returns ESB_STS_NOERR. Otherwise, returns an error code.

**Access**

A call to this function requires read privileges (ESB_PRIV_READ) for the data cell or the active database.

**Example**

```plaintext
Declare Function EsbLROGetCatalog Lib "esbapin" _
      (ByVal hCtx As Long, ByVal memCount As Long, _
ByVal memComb As String, PulCount As Integer) As Long
Public Sub ESB_LROGetCatalog()
```
Dim Desc As ESB_LRODESC_API_T
Dim Items As Integer
Dim memCount As Long
Dim memComb As String
Dim i As Integer

memCount = 5
memComb = "Jan" & vbCrLf & "Sales" & _
   "Cola" & vbCrLf & "Utah" & _
   "Actual"

sts = EsbLROGetCatalog(hCtx, memCount, _
   memComb, Items)

If sts = 0 Then
   For i = 1 To Items
      '*******************************
      '* Get the next LRO description
      '* item from the list
      '*******************************
      sts = EsbGetNextItem(hCtx,
         ESB_LRO_TYPE, Desc)    Next i
   End If
End Sub

See Also

- “Constant and Structure Definitions for Linked Objects” on page 1155
- EsbGetNextItem
- EsbLROAddObject
- EsbLROUpdateObject
- EsbLROGetObject
- EsbLRODeleteObject

EsbLROGetMemberCombo

Retrieves the nth member from the member combination list of the current LRO.

Syntax

EsbLROGetMemberCombo (hCtx, memberIndex, memberName)

ByVal hCtx As ESB_HCTX_T
ByRef memberIndex As ESB_ULONG_T
ByVal memberName As ESB_MBRNAME_T

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>VB API context handle.</td>
</tr>
<tr>
<td>memberIndex</td>
<td>Position in the member list of the member to be returned.</td>
</tr>
<tr>
<td>memberName</td>
<td>Returned member's name.</td>
</tr>
</tbody>
</table>
Notes

- EsbLROGetMemberCombo() returns the member combination (that identifies the data cell associated with the linked object) which memComb in “ESB_LRODESC_API_T” on page 1156 fails to return.
- To ensure that the list in memory associated with hCtx is a list of LROs, call EsbLROListObjects() or EsbLROGetObject() before you call EsbLROGetMemberCombo().
- After you call EsbLROListObjects(), you must call EsbGetNextItem(). EsbLROGetMemberCombo() then operates on the current LRO fetched by EsbGetNextItem().
- You can still use memCount in ESB_LRODESC_API_T to find the number of member names in the member combination identifying the data cell. See Example.

Return Value

If successful, returns a member in memberName. If fails, returns -1 to indicate the current object is not an LRO type, or 1 to indicate out of bounds. Out of bounds means either there is no member at the memberIndex position, or there is no LRO at the current item. See Notes.

Access

This function requires no special privileges.

Example

Declare Function EsbLROGetMemberCombo Lib "ESBAPIN" (ByVal hCtx As Long, ByVal MemberIndex As Long, ByVal MemberName As String * ESB_MBRNAMELEN) As Long

Sub ESB_LROGetMemberCombo()
    Dim userName As String * ESB_USERNAMELEN
    Dim listDate As Long
    Dim Count As Integer
    Dim Desc As ESB_LRODESC_API_T
    Dim i As Integer
    Dim j As Integer
    Dim CutOffDate As Date
    Dim MemberName As String * ESB_MBRNAMELEN

    Const ESB_REFERENCE_DATE = #1/1/70#
    userName = "admin"
    CutOffDate = #8/1/97#
    listDate = DateDiff("s", CutOffDate, ESB_REFERENCE_DATE)

    sts = EsbLROListObjects(hCtx, userName, listDate, Count)
    If sts = 0 Then
        For i = 1 To Count
            '**************************************
            ' * Get the next LRO item from the list
            '**************************************
            sts = EsbGetNextItem(hCtx, ESB_LRO_TYPE, Desc)
            If sts = 0 Then
                For j = 1 To Desc.memCount

EsbLROGetMemberCombo

Get the jth member from the member list of the current LRO

```vbnet
sts = EsbLROGetMemberCombo(hCtx, j, MemberName)
Next j
Next i
End If
End Sub
```

See Also

- EsbGetNextItem
- EsbLROGetObject
- EsbLROListObjects

### EsbLROGetObject

Retrieves an object linked to a data cell in a database.

**Syntax**

```vbnet
eSbLROGetObject (hCtx, pLinkId, targetFile, usOption, pLRODesc)
```

**Parameter Description**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>API context handle.</td>
</tr>
<tr>
<td>pLinkId</td>
<td>Pointer to object identification structure. The link ID is returned by EsbLROAddObject through the “ESB_LRODESC_API_T” on page 1156 structure.</td>
</tr>
<tr>
<td>targetFile</td>
<td>The name of the target file into which the object is retrieved.</td>
</tr>
<tr>
<td>usOption</td>
<td>Option specifying whether to retrieve the object, its catalog entry, or both. Use one of the following:</td>
</tr>
<tr>
<td></td>
<td>- ESB_LRO_OBJ_API retrieves only the object.</td>
</tr>
<tr>
<td></td>
<td>- ESB_LRO_CATALOG_API retrieves only the catalog entry.</td>
</tr>
<tr>
<td></td>
<td>- ESB_LRO_BOTH_API retrieves object and catalog entry.</td>
</tr>
<tr>
<td>pLRODesc</td>
<td>Object's description structure, “ESB_LRODESC_API_T” on page 1156.</td>
</tr>
</tbody>
</table>

**Notes**

Cell notes are part of the catalog entry for an object. To determine if the object is a cell note, check the ObjectType field in “ESB_LRODESC_API_T” on page 1156. To retrieve a cell note, use ESB_LRO_CATALOG_API for the usOption parameter. The note contents are contained in “ESB_LRODESC_API_T” on page 1156.

**Return Value**

If successful, returns ESB_STS_NOERR. Otherwise, returns an error code.
Access

A call to this function requires read privileges (ESB_PRIV_READ) for the data cell or the active database.

Example

Declare Function EsbLROGetObject Lib "esbapin" _
(ByVal hCtx As Long, pLinkID As ESB_LROHANDLE_API_T, _
ByVal targetFile As String, ByVal usOption As Integer, _
pLRODesc As ESB_LRODESC_API_T) As Long

Public Sub ESB_LROGetObject()   Dim Desc As ESB_LRODESC_API_T
   Dim LinkID As ESB_LROHANDLE_API_T
   Dim TargetFile As String
   Dim opt As Integer
   Dim InputMsg As String
   LinkID.hObject = 1
   LinkID.cellKey.cellOffset = 0
   LinkID.cellKey.blkOffset = 198
   LinkID.cellKey.segment = 0
   TargetFile = "c:\docs\myfile.doc"
   InputMsg = "Danger, Will Robinson"
   opt = InputBox(InputMsg, , ESB_LRO_BOTH_API)
   sts = EsbLROGetObject(hCtx, LinkID, TargetFile, _
      opt, Desc)
End Sub

See Also

● “Constant and Structure Definitions for Linked Objects” on page 1155
● EsbLROAddObject
● EsbLROGetMemberCombo
● EsbLROUpdateObject
● EsbLRODeleteObject

EsbLROListObjects

Retrieves a list of all objects linked to cells in the active database for a given user name and/or modification date.

Syntax

EsbLROListObjects (hCtx, userName, listDate, PulCount)
ByVal hCtx    As Long
ByVal userName As String
ByVal listDate As Long
ByVal PulCount As Integer
Parameter Description

hCtx  API context handle.

userName  A user name. If specified, returns a list of all objects last modified by the given user.

listDate  A modification date. If specified, returns a list of all objects modified on or before the given date. The time is a Long representing the number of seconds since January 1, 1970.

PulCount  Number of LRO catalog entries returned.

Notes

- If you specify both the userName and listDate parameters, objects meeting both criteria are listed.
- To retrieve the list of objects, call EsbGetNextItem after calling this function. The example code demonstrates how to do this.
- In order to replicate the functionality of EssLROListObjects using EsbLROLListObjects(), you must call EsbLROGetMemberCombo after calling EsbGetNextItem.

Return Value

If successful, returns ESB_STS_NOERR. Otherwise, returns an error code.

Access

A call to this function requires read privileges (ESB_PRIV_READ) to the date cell or the active database.

Example

Declare Function EsbLROLListObjects Lib "esbapin" _
(ByVal hCtx As Long, ByVal userName As String, _
ByVal listDate As Long, PulCount As Integer) As Long

Public Sub ESB_LROLListObjects()

    Dim userName As String * ESB_USERNAMELEN
    Dim listDate As Long
    Dim Items As Integer
    Dim Desc As ESB_LRODESC_API_T
    Dim i As Integer
    Dim CutOffDate As Date

    Const ESB_REFERENCE_DATE = #1/1/70#
    userName = "admin"
    CutOffDate = #8/1/97#
    listDate = DateDiff("s", CutOffDate, _
        ESB_REFERENCE_DATE)

    sts = EsbLROLListObjects(hCtx, userName, _
        listDate, Items)

    If sts = 0 Then
        For i = 1 To Items
            '*******************************
            1372 Visual Basic Main API Functions
' Get the next LRO description
' item from the list
'******************************************************************************
sts = EsbGetNextItem(hCtx, ESB_LRO_TYPE, Desc)
Next i
End If
End Sub

See Also

- “Constant and Structure Definitions for Linked Objects” on page 1155
- EsbGetNextItem
- EsbLROGetCatalog
- EsbLROGetMemberCombo
- EsbLROPurgeObjects

EsbLROPurgeObjects

Deletes all objects linked to cells in the active database for a given user name and/or modification date.

Syntax

EsbLROPurgeObjects (hCtx, userName, purgeDate, PulCount)
ByVal hCtx As Long
ByVal userName As String
ByVal purgeDate As Long
ByVal PulCount As Integer

Parameter Description

hCtx API context handle.

userName Pointer to a user name. If specified, deletes all objects last modified by the given user.

purgeDate A modification date. If specified, deletes all objects modified on or before the given date. The date is a Long representing the number of seconds since January 1, 1970.

PulCount Number of LRO catalog entries purged.

Notes

- If you specify both the userName and purgeDate parameters, objects meeting both criteria are deleted.
- EsbLROPurgeObjects() generates a list of the objects it deletes. After calling this function, use EsbGetNextItem to retrieve information about each deleted object. The example code demonstrates how to do this.

Return Value

If successful, returns ESB_STS_NOERR. Otherwise, returns an error code.
Access

A call to this function requires design privileges (ESB_PRIV_DBDESIGN) for the data cell or the active database.

Example

Declare Function EsbLROPurgeObjects Lib "esbapin" _
(ByVal hCtx As Long, ByVal userName As String, _
ByVal purgeDate As Long, PulCount As Integer) As Long

Public Sub ESB_LROPurgeObjects()   Dim userName As String * ESB_USERNAMELEN
   Dim purgeDate As Long
   Dim Items As Integer
   Dim Desc As ESB_LRODESC_API_T
   Dim CutOffDate As Date
   Dim i As Integer   Const ESB_REFERENCE_DATE = #1/1/70#
   userName = "admin"

   CutOffDate = #8/1/97#
   purgeDate = DateDiff("s", ESB_REFERENCE_DATE, _
                  CutOffDate)

   sts = EsbLROPurgeObjects(hCtx, userName, _
                          purgeDate, Items)

   If sts = 0 Then
      For i = 1 To Items
         '*******************************
         ' * Get the next LRO description
         ' * item from the list
         '*******************************
         sts = EsbGetNextItem(hCtx,
                          ESB_LRO_TYPE, Desc)      Next i
   End If
End Sub

See Also

- “Constant and Structure Definitions for Linked Objects” on page 1155
- EsbGetNextItem
- EsbLROGetCatalog
- EsbLRODeleteObject
- EsbLRODeleteCellObjects

EsbLROUpdateObject

Stores an updated version of a linked object on the server.

Syntax

EsbLROUpdateObject (hCtx, pLinkId, usOption, pLRODesc)
ByVal hCtx As Long
    pLinkId As ESB_LROHANDLE_API_T
ByVal usOption As Integer
    pLRODesc As ESB_LRODESC_API_T
### Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>API context handle.</td>
</tr>
<tr>
<td>pLinkId</td>
<td>Object identification structure.</td>
</tr>
<tr>
<td>usOption</td>
<td>Option specifying which part of the object to update. Use one of the following:</td>
</tr>
<tr>
<td></td>
<td>- ESB_LRO_BOTH_API to update both the object’s file and catalog entry</td>
</tr>
<tr>
<td></td>
<td>- ESB_LRO_OBJ_API to update only the object’s file</td>
</tr>
<tr>
<td></td>
<td>- ESB_LRO_CATALOG_API to update only the object’s catalog entry</td>
</tr>
<tr>
<td>pLRODesc</td>
<td>Object’s description structure, “ESB_LRODESC_API_T” on page 1156.</td>
</tr>
</tbody>
</table>

### Notes

- Essbase assigns the link ID when you create an object and stores it in the object’s catalog entry. Use `EsbLROGetCatalog` to retrieve the catalog entry contained in the description structure, “ESB_LRODESC_API_T” on page 1156. You can then modify the description structure and call `EsbLROUpdateObject()` to save your changes on the server.

- Specify the `usOption` parameter as follows:
  - If the object is a cell note, use ESB_LRO_CATALOG_API. Cell notes are stored in the catalog entry.
  - If the object is a file, use ESB_LRO_BOTH_API to update both the file contents and the catalog.
  - If you only want to update the catalog information (such as the object description or user name), use ESB_LRO_CATALOG_API. In this case, the file contents are not updated.
  - If you only want to update the file contents and not the catalog, use ESB_LRO_OBJ_API. In this case, only the file contents and modification date are updated.

- The linked object can be any of the following types:
  - A flat file, such as a Word document, Excel spreadsheet, or bitmap image.
  - A cell note containing up to 599 characters of text.
  - A link to another Essbase database (linked partitions feature).

### Return Value

If successful, returns ESB_STS_NOERR. Otherwise, returns an error code.

### Access

A call to this function requires write privileges (ESB_PRIV_WRITE) to the data cell or the active database.

### Example

```visualbasic
Declare Function EsbLROUpdateObject Lib "esbapin" _
(ByVal hCtx As Long, ByVal pLinkID As ESB_LROHANDLE_API_T, _
ByVal usOption As Integer, _
pLRODesc As ESB_LRODESC_API_T) As Long
```
Public Sub ESB_LROUpdateObject()
    Dim LinkID As ESB_LROHANDLE_API_T
    Dim Desc As ESB_LRODESC_API_T
    Dim opt As Integer

    LinkID.hObject = 1
    LinkID.cellKey.cellOffset = 0
    LinkID.cellKey.blkOffset = 198
    LinkID.cellKey.segment = 0

    Desc.userName = "admin"
    Desc.ObjType = ESB_LROTYPE_CELLNOTE_API
    Desc.note = "New Note from DFS"
    opt = ESB_STORE_OBJECT_API

    sts = EsbLROUpdateObject(hCtx, LinkID, _
                           opt, Desc)
End Sub

See Also

- “ESB_LROINFO_API_T” on page 1157
- EsbLROGetObject
- EsbLROAddObject
- EsbLRODeleteObject

**EsbPartitionApplyOtlChangeFile**

Requests server to apply a list of outline change files.

**Syntax**

```vbnet
EsbPartitionApplyOtlChangeFile (hCtx, usFileNum, fileList)
```

**Parameter** | **Description**
--- | ---
```
hCtx
```
Handle to API context.
```
usFileNum
```
The number of outline change files.
```
fileList
```
A string of filenames delimited by CR/LF. The size of the array is defined by `usFileNum`.

**Notes**

Obtain `filename` by calling **EsbPartitionGetOtlChanges**.

**Return Value**

Returns zero if successful, error code if unsuccessful.

**Access**

Database Manager privilege is required.
Example

Public Sub ESB_PartitionApplyOtlChangeFile()
    Dim FileItems As Integer
    Dim Filelist As String
    Dim ProcState As ESB_PROCSTATE_T

    FileItems = 1
    Filelist = "C:\ESSBASE\APP\SAMPPART\COMPANY\ESS00001.CHG"
    sts = EsbPartitionApplyOtlChangeFile(hCtx, FileItems, Filelist)
        If sts = 0 Then
            sts = EsbGetProcessState(hCtx, ProcState)
            Do Until ProcState.State = ESB_STATE_DONE
                sts = EsbGetProcessState(hCtx, ProcState)
            Loop
        End If
End Sub

See Also

- “Constant and Structure Definitions for Partitions ” on page 1157
- EsbPartitionGetAreaCellCount
- EsbPartitionGetList
- EsbPartitionGetOtlChanges
- EsbPartitionGetReplCells
- EsbPartitionPurgeOtlChangeFile
- EsbPartitionPutReplCells
- EsbPartitionResetOtlChangeTime

EsbPartitionApplyOtlChangeRecs

Applies outline changes to a target outline. This function is designed to be used interactively with EsbPartitionReadOtlChangeFile after a call to EsbPartitionGetOtlChanges.

Syntax

EsbPartitionApplyOtlChangeRecs (hCtx, pszChgFileName, MetaChangeReadHandle, SourceTime)

ByVal hCtx As Long
ByVal pszChgFileName As String
ByVal MetaChangeReadHandle As Long
ByVal SourceTime As Long

Parameter | Description
---|---
hCtx | Handle to API context.
pszChgFileName | The name of the change file.
MetaChangeReadHandle | Handle to the metadata change file.
SourceTime | The time of the latest change to the metadata file.

Notes

- There may be dependencies among change records.
Rejecting a record may cause a failure when applying another record. For example, you have two records "add A" and "add AA as a child of A". Rejecting the first record and accepting the second causes an apply failure.

**Return Value**

Returns zero if successful; error code if unsuccessful.

**Access**

A call to this function requires Database Manager access privileges.

**See Also**

- `EsbPartitionApplyOtlChangeFile`
- `EsbPartitionGetAreaCellCount`
- `EsbPartitionGetList`
- `EsbPartitionGetOtlChanges`
- `EsbPartitionGetReplCells`
- `EsbPartitionPurgeOtlChangeFile`
- `EsbPartitionPutReplCells`
- `EsbPartitionReadOtlChangeFile`
- `EsbPartitionResetOtlChangeTime`

---

### EsbPartitionGetAreaCellCount

Returns the number of cells in the specified slice string.

**Syntax**

```vbnet
EsbPartitionGetAreaCellCount (hCtx, pszSlice, pdCount)
```

**Parameter Description**

- **hCtx**  
  API context handle.
- **pszSlice**  
  Input slice definition to be checked.
- **pdCount**  
  Returns number of cells here.

**Return Value**

Returns zero if successful; error code if unsuccessful.

**Example**

```vbnet
Public Sub ESB_PartitionGetAreaCellItems()
    Dim Slice As String
    Dim CellItems As Double

    Slice = "@Idescendants(Market)"
```
sts = EsbPartitionGetAreaCellCount(hCtx, Slice, CellItems)

    If sts = 0 Then MsgBox Items
End Sub

See Also

- “Constant and Structure Definitions for Partitions” on page 1157
- EsbPartitionApplyOtlChangeFile
- EsbPartitionGetList
- EsbPartitionGetOtlChanges
- EsbPartitionGetReplCells
- EsbPartitionPurgeOtlChangeFile
- EsbPartitionPutReplCells
- EsbPartitionResetOtlChangeTime

EsbPartitionGetList

Returns a list of the partition definitions in which the currently selected database participates.

Syntax

EsbPartitionGetList (hCtx, SelectRegion, pusCount)

ByVal hCtx As Long
SelectRegion As ESB_PARTSLCT_T
pusCount As Integer

Parameter Description

hCtx API context handle.
SelectRegion Criteria to select partitions.
pusCount Items of partitions returned.

Return Value

Returns zero if successful; error code if unsuccessful.

Example

Public Sub ESB_PartitionGetList()
    Dim SelectPartition As ESB_PARTSLCT_T
    Dim Partition As ESB_PART_INFO_T
    Dim Items As Integer
    Dim i As Integer

    SelectPartition.OperationTypes = ESB_PARTITION_OP_ALL
    SelectPartition.DirectionTypes = ESB_PARTITION_DATA_BOTH
    SelectPartition.MetaDirectionTypes = ESB_PARTITION_META_BOTH
    sts = EsbPartitionGetList(hCtx, SelectPartition, Items)

    If sts = 0 And Items > 0 Then
        For i = 1 To Items
            sts = EsbGetNextItem(hCtx, ESB_PART_INFO_TYPE, Partition)
            '*****************************************
            ' Get information in ESB_PART_INFO_T here
        Next i
    End If
End Sub
End Sub

See Also

- “Constant and Structure Definitions for Partitions” on page 1157
- EsbPartitionApplyOtlChangeFile
- EsbPartitionGetAreaCellCount
- EsbPartitionGetOtlChanges
- EsbPartitionGetReplCells
- EsbPartitionPurgeOtlChangeFile
- EsbPartitionPutReplCells
- EsbPartitionResetOtlChangeTime

EsbPartitionGetOtlChanges

Pulls outline changes from a given source and stores them in a file.

Syntax

`EsbPartitionGetOtlChanges(hCtx, MetaQuery, ChangeFile, szChangeFile)`

- **hCtx**: API context handle.
- **MetaQuery**: Change query criteria.
- **ChangeFile**: Caller allocated change file and informational structure.
- **szChangeFile**: The size of the change file.

Parameter | Description
--- | ---
**hCtx** | API context handle.
**MetaQuery** | Change query criteria.
**ChangeFile** | Caller allocated change file and informational structure.
**szChangeFile** | The size of the change file.

Notes

Multiple files must be passed as a CR/LF delimited file list. You must use the path name on the server (as referenced by `EsbGetOtlChanges()`).

Return Value

Returns zero if successful, error code if unsuccessful.

Access

A call to this function requires Database Manager access privileges.

Example

```vbnet
Public Sub ESB_PartitionGetOtlChanges()   Dim PartQuery As ESB_PARTOTL_QUERY_T
Const SizeofChangeFile = 150
    Dim ChangeFile As String * SizeofChangeFile
```
PartQuery.OperationType = ESB_PARTITION_OP_REPLICATED
PartQuery.HostDatabase.HostName = "Dscharton2" PartQuery.HostDatabase.AppName = "Samplest"
PartQuery.HostDatabase.DbName = "East"
PartQuery.MetaFilter.TimeStamp = DateDiff("s", #1/1/70#, #6/18/97#) PartQuery.MetaFilter.DimFilter = ESB_PARTITION_OTLDIM_ALL
PartQuery.MetaFilter.MbrFilter = ESB_PARTITION_OTLMBR_ALL
PartQuery.MetaFilter.MbrAttrFilter = ESB_PARTITION_OTLMBRATTR_ALL
sts = EsbPartitionGetOtlChanges(hCtx, PartQuery, ChangeFile, SizeofChangeFile) If sts = 0 Then MsgBox ChangeFile

End Sub

See Also

- “Constant and Structure Definitions for Partitions” on page 1157
- EsbPartitionApplyOtlChangeFile
- EsbPartitionGetAreaCellCount
- EsbPartitionGetList
- EsbPartitionGetReplCells
- EsbPartitionPurgeOtlChangeFile
- EsbPartitionPutReplCells
- EsbPartitionResetOtlChangeTime

**EsbPartitionGetReplCells**

Replicates all data cells that are identified in the replication partition from the source database to the selected target database.

**Syntax**

```vbnet
EsbPartitionGetReplCells (hCtx, ReplicatedRegion, HostAppDbList)
```

**Parameter** | **Description**
--- | ---
hCtx | API context handle.
ReplicatedRegion | Partition information.
HostAppDbList | A string of server, application, and database sets delimited by CR/LF.

**Return Value**

Returns zero if successful; error code if unsuccessful.
Access

A call to this function requires Database Manager access privileges.

Example

Public Sub ESB_PartitionGetReplCells()   Dim ReplPartition As ESB_PART_REPL_T   Dim HostAppDbList As String   Dim ProcState     As ESB_PROCSTATE_T   Dim ind, i        As Long

    ReplPartition.PartitionCount   = -1  'All areas    ReplPartition.UpdatedOnly      = 0 'Updated only cells    HostAppDbList                  = "localhost" & vbCrLf & _        "Sampeast" & vbCrLf & _        "East"

    sts = EsbPartitionGetReplCells(hCtx, ReplPartition, HostAppDbList)

    If sts = 0 Then
        sts = EsbGetProcessState(hCtx, ProcState)
        Do Until ProcState.State = ESB_STATE_DONE
            sts = EsbGetProcessState(hCtx, ProcState)
        Loop
    End If
End Sub

See Also

- “Constant and Structure Definitions for Partitions” on page 1157
- EsbPartitionApplyOtlChangeFile
- EsbPartitionGetAreaCellCount
- EsbPartitionGetList
- EsbPartitionGetOtlChanges
- EsbPartitionPutReplCells
- EsbPartitionResetOtlChangeTime

EsbPartitionPurgeOtlChangeFile

Purges changes made previous to the time specified with the TimeStamp parameter.

Syntax

EsbPartitionPurgeOtlChangeFile (hCtx, pRegion, TimeStamp)
ByVal hCtx       As Long
    pRegion     As ESB_PART_DEFINED_T
ByVal TimeStamp  As Long

Parameter          Description

hCtx              API context handle.

pRegion           Partition specification.
Parameter | Description
--- | ---
TimeStamp | Purge all change records before this time.

**Return Value**

Returns zero if successful; error code if unsuccessful.

**Example**

Public Sub Esb_PartitionPurgeOtlChangeFile()

    Dim PartitionInfo As ESB_PART_DEFINED_T
    Dim TimeStamp As Variant

    PartitionInfo.usType = ESB_PARTITION_OP_REPLICATED
    PartitionInfo.Direction = ESB_PARTITION_DATA_SOURCE

    PartitionInfo.HostDatabase.HostName = "Jsnider"
    PartitionInfo.HostDatabase.AppName = "Samppart"
    PartitionInfo.HostDatabase.DbName = "Company"

    TimeStamp = DateDiff("s", #1/1/70#, #7/7/97#)
    sts = EsbPartitionPurgeOtlChangeFile(hCtx, PartitionInfo, TimeStamp)

End Sub

**See Also**

- "Constant and Structure Definitions for Partitions " on page 1157
- EsbPartitionApplyOtlChangeFile
- EsbPartitionGetAreaCellCount
- EsbPartitionGetList
- EsbPartitionGetOtlChanges
- EsbPartitionGetReplCells
- EsbPartitionPutReplCells
- EsbPartitionResetOtlChangeTime

**EsbPartitionPutReplCells**

Replicates all data cells that are identified in the replication partition from the selected source database to the target database.

**Syntax**

```Visual Basic
EsbPartitionPutReplCells (hCtx, ReplicatedRegion, HostAppDbList)
ByVal hCtx As Long
    ReplicatedRegion As ESB_PART_REPL_T
ByVal HostAppDbList As String
```

**Parameter | Description**
--- | ---
hCtx | API context handle.
### Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReplicatedPartition</td>
<td>Partition information.</td>
</tr>
<tr>
<td>HostAppDbList</td>
<td>List of database/applications on the host server.</td>
</tr>
</tbody>
</table>

### Notes

This routine removes the file if it is empty after purging.

### Return Value

Returns zero if successful; error code if unsuccessful.

### Access

A call to this function requires Database Manager privilege.

### Example

```vbnet
Public Sub ESB_PartitionPutReplCells()

    Dim ReplPartition As ESB_PART_REPL_T
    Dim HostAppDbList As String
    Dim ProcState     As ESB_PROCSTATE_T
    Dim ind, i        As Long

    ReplPartition.PartitionCount   = -1  'All areas
    ReplPartition.UpdatedOnly      = 0 'Updated only cells
    HostAppDbList                  = "localhost" & vbCrLf & _
                                    "Sampeast" & vbCrLf & _
                                    "East"

    sts = EsbPartitionPutReplCells(hCtx, ReplPartition, HostAppDbList)

    If sts = 0 Then
        sts = EsbGetProcessState(hCtx, ProcState)
        Do Until ProcState.State = ESB_STATE_DONE
            sts = EsbGetProcessState(hCtx, ProcState)
            Loop
        End If

End Sub
```

### See Also

- “Constant and Structure Definitions for Partitions ” on page 1157
- `EsbPartitionApplyOtlChangeFile`
- `EsbPartitionGetAreaCellCount`
- `EsbPartitionGetList`
- `EsbPartitionGetOtlChanges`
- `EsbPartitionGetReplCells`
- `EsbPartitionPurgeOtlChangeFile`
- `EsbPartitionResetOtlChangeTime`
**EsbPartitionReadOtlChangeFile**

Reads changes from a change file (*.CHG) on the target database into memory. This function is designed to be used interactively with `EsbPartitionApplyOtlChangeRecs()` after a call to `EsbPartitionGetOtlChanges()`. This function can be used with filters.

**Syntax**

```plaintext
EsbPartitionReadOtlChangeFile (hCtx, pszChgFileName, QueryFilter, MetaChangeReadHandle, SourceTime)
ByVal hCtx As Long
ByVal pszChgFileName As String
ByRef QueryFilter As ESB_PARTOTL_QRY_FILTER_T
ByRef MetaChangeReadHandle As Long
ByRef SourceTime As Long
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>API context handle.</td>
</tr>
<tr>
<td>pszChgFileName</td>
<td>The name of the metadata change file.</td>
</tr>
<tr>
<td>QueryFilter</td>
<td>The query filter expression.</td>
</tr>
<tr>
<td>MetaChangeReadHandle</td>
<td>Handle for the metadata change file.</td>
</tr>
<tr>
<td>SourceTime</td>
<td>The time of the latest change to the metadata file.</td>
</tr>
</tbody>
</table>

**Notes**

This routine returns a time in `pMetaChangeRead`. This is the same time stamp you should pass to `EsbPartitionApplyOtlChangeRecs` to update the timestamp at the target database.

**Return Value**

Returns zero if successful; error code if unsuccessful.

**Access**

A call to this function requires Database Manager access privileges.

**See Also**

- `EsbPartitionApplyOtlChangeFile`
- `EsbPartitionApplyOtlChangeRecs`
- `EsbPartitionGetAreaCellCount`
- `EsbPartitionGetList`
- `EsbPartitionGetOtlChanges`
- `EsbPartitionGetRep1Cells`
- `EsbPartitionPurgeOtlChangeFile`
- `EsbPartitionPutRep1Cells`
- `EsbPartitionResetOtlChangeTime`
**EsbPartitionResetOtlChangeTime**

Takes the last change time from the source partition and assigns it as a last metadata change time of a destination partition.

**Syntax**

```plaintext
EsbPartitionResetOtlChangeTime (hCtx, pSourceRegion, pDestRegion)
```

```plaintext
ByVal hCtx As Long
pSourceRegion As ESB_PART_DEFINED_T
pDestRegion As ESB_PART_DEFINED_T
```

**Parameter Description**

- `hCtx`: API context handle.
- `pSourceRegion`: Partition for the new time.
- `pDestRegion`: Partition where the time is reset.

**Notes**

- The source partition refers to a partition that provides a time stamp, and a target partition refers to a partition which receives the time stamp.
- A source partition does not have to be either a data source partition or an outline source partition.

**Return Value**

Returns zero if successful; error code if unsuccessful.

**Access**

A call to this function requires Database Manager access privileges.

**Example**

```visualbasic
Public Sub ESB_PartitionResetOtlChangeTime()
    Dim SourcePartition As ESB_PART_DEFINED_T
    Dim DestPartition As ESB_PART_DEFINED_T

    SourcePartition.usType = ESB_PARTITION_OP_REPLICATED
    DestPartition.usType = ESB_PARTITION_OP_REPLICATED

    SourcePartition.Direction = ESB_PARTITION_DATA_SOURCE
    DestPartition.Direction = ESB_PARTITION_DATA_TARGET

    SourcePartition.HostDatabase.HostName = "Dscharton2"
    DestPartition.HostDatabase.HostName = "Dscharton2"

    SourcePartition.HostDatabase.AppName = "Sampeast"
    DestPartition.HostDatabase.AppName = "East"

    SourcePartition.HostDatabase.DbName = "Samppart"
    DestPartition.HostDatabase.DbName = "Company"
```
sts = EsbPartitionResetOtlChangeTime(hCtx, _
    SourcePartition, DestPartition)

End Sub

See Also

- “Constant and Structure Definitions for Partitions” on page 1157
- EsbPartitionApplyOtlChangeFile
- EsbPartitionGetAreaCellCount
- EsbPartitionGetList
- EsbPartitionGetOtlChanges
- EsbPartitionGetReplCells
- EsbPartitionPurgeOtlChangeFile
- EsbPartitionPutReplCells

**EsbPutObject**

Copies an object from a local file to the server or client object system, and optionally unlocks it.

**Syntax**

```vbnet
EsbPutObject (hCtx, ObjType,AppName, DbName, ObjName, LocalName, isUnlock)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>VB API context handle. Can be local context handle returned by EsbCreateLocalContext().</td>
</tr>
<tr>
<td>ObjType</td>
<td>Object type (must be single type). Refer to Table 15 for a list of possible values.</td>
</tr>
<tr>
<td>AppName</td>
<td>Application name.</td>
</tr>
<tr>
<td>DbName</td>
<td>Database name. If an empty string, uses the application sub-directory.</td>
</tr>
<tr>
<td>ObjName</td>
<td>Name of an object to put.</td>
</tr>
<tr>
<td>LocalName</td>
<td>Full path name of local source file on client.</td>
</tr>
<tr>
<td>isUnlock</td>
<td>Flag to control object unlocking. If TRUE, the server object is unlocked to allow updates by other users.</td>
</tr>
</tbody>
</table>

**Notes**

In order to put an object which already exists on the server, it must have previously been locked by the caller. If the object does not already exist on the server, it will be created.

**Return Value**

If successful, the object is copied to the server from the local file specified by LocalName.
Access

This function requires the caller to have the appropriate level of access to the specified application and/or database to contain the object (depending on the object type). To unlock the object (unlock flag is TRUE), the caller must have application or Database Design privilege (ESB_PRIV_APPDESIGN or ESB_PRIV_DBDESIGN) for the specified application or database containing the object.

Example

Declare Function EsbPutObject Lib "ESBAPIN" (ByVal hCtx As Long, ByVal ObjType As Integer, ByValAppName As String, ByVal DbName As String, ByVal ObjName As String, ByVal LocalName As String, ByVal Unlock As Integer) As Long

Sub ESB_PutObject ()
    Dim sts As Long
    DimAppName As String
    DimDbName As String
    DimObjName As String
    DimObjType As Integer
    DimLocalName As String
    DimUnlock As Integer
    AppName = "Sample"
    DbName = "Basic"
    ObjName = "Basic"
    LocalName = "C:\ESSBASE\CLIENT\BASIC.TXT"
    Unlock = ESB_YES
    ' Put Object
    '***********
    sts = EsbPutObject (hCtx, ObjType, AppName,
    DbName, ObjName, LocalName, Unlock)
End Sub

See Also

- EsbGetObject
- EsbLockObject
- EsbUnlockObject
- EsbListObjects

EsbQueryDatabaseMembers

Performs a report-style query to list a selection of database member information.

Syntax

EsbQueryDatabaseMembers (hCtx, mbrQuery)
ByVal hCtx As Long
ByVal mbrQuery As String

Parameter Description

hCtx VB API context handle.
Parameter Description

mbrQuery  Member query string. A query string is a command similar to a report specification. For descriptions of report specifications refer to the *Oracle Essbase Technical Reference*. Valid query strings are listed in Notes, below. String must be less than 64 KB in length.

Notes

- The member information returned by this query must be read by calling `EsbGetString()` until an empty string is returned.
- This function supports an attribute member long name.
- The Member query string consists of a selection string and an optional sorting command followed by an optional output command. The form is:

  \[mbrQuery ==: <selectionstring> \[<sortcommand> \[<outputcommand>\]\] \]

- The valid values for member `<selectionstring>` are:

  - `<CHILDRENOF` -- returns `ICHILDRENOF`
  - `<ALLINSAMEDIM`
  - `<DIMTOP`
  - `<OFSAMEGENERATION`
  - `<ONSAMELEVELAS`
  - `<ANCESTORSOF` -- returns `IANCESTORSOF`
  - `<PARENTOF`
  - `<DESCENDANTSOF` -- returns `IDESCENDANTSOF`
  - `<ALLSIBLINGSOF`
  - `<LSIBLINGOF`

- Valid values for `<sortcommand>` are:

  - `<SORTASCENDING`
  - `<SORTDESCENDING`
  - `<SORTNONE`
  - `<SORTMBRNAMES`
  - `<SORTALTNAMES`
  - `<SORTMBRNUMBERS`
  - `<SORTDIMNUMBERS`
  - `<SORTLEVELNUMBERS`
  - `<SORTGENERATION`

- The form for `<outputcommand>` is:

  \[<outputcommand> ==: Item \[separator\] | FORMAT \{<item> \[separator\]\} \]

  To obtain a one-item list of information on a member, use the following output commands:

  \[<outputcommand> ==: \{<MBRNAMES | <ALTNAMES | <MBRNUMBERS | <DIMNUMBERS | <LEVELNUMBERS | <GENERATIONS | <CALCSTRINGS | <UCALCS | <TABSEPARATED | <SPACESEPARATED | <COMMASEPARATED \}\]
To obtain a list of two or more items of information on a member, use a format specification clause. Specify the items you want listed, their order, and what character to use to separate them. The syntax for a format specification clause is:

```format <item> [separator] <item> [separator]```

The valid values for `<item>` are:

- `MBRNAMES`
- `ALTNAMES`
- `MBRNUMBERS`
- `DIMNUMBERS`
- `LEVELNUMBERS`
- `GENERATIONS`
- `CALCSTRINGS`
- `UCALCS`
- `ATTRIBUTES`

`ATTRIBUTES` are listed as the number of attributes followed by a tab-separated list of attribute names.

The valid values for `separator` are:

- `TABSEPARATED`
- `SPACESEPARATED`
- `COMMASEPARATED`
- `NEWLINESEPARATED`

If you do not specify a separator, the default is `TABSEPARATED`.

Here is a sample script:

```plaintext
login "local" "user1" "password" "" ""
select "attr" "attr"
GetMembers "<newlineSeparated
<format {
MBRNAMES SPACESEPARATED ALTNAMES TABSEPARATED
MBRNUMBERS SPACESEPARATED DIMNUMBERS TABSEPARATED
LEVELNUMBERS SPACESEPARATED GENERATIONS TABSEPARATED
CALCSTRINGS SPACESEPARATED UCALCS TABSEPARATED
DIMTYPES SPACESEPARATED STATUSES TABSEPARATED
ATTRIBUTES
}
<descendants Product 

Return Value
None.

Access
This function requires the caller to have access to the database, and to have selected it as their active database using `EsbSetActive()`.

Example
Declare Function EsbQueryDatabaseMembers Lib "ESBAPIN" (ByVal hCtx As Long, ByVal Query As String) As Long
Sub ESB_QueryDatabaseMembers ()
    Dim sts As Long
    Dim Query As String
    Const szMString = 256
    Dim MString As String * szMString
    Query = "<ALLINSAMEDIM"    '********************
        'Query Database members
        '***********************
    sts = EsbQueryDatabaseMembers (hCtx, Query)    '********************
        'Print out all strings
        '***********************
    If sts = 0 Then
        sts = EsbGetString (hCtx, MString, szMString)
        Do While Mid$(MString, 1, 1) <> Chr$(0)
            Print MString
            sts = EsbGetString (hCtx, MString, szMString)
        Loop
    End If
End Sub

See Also

- EsbCheckMemberName
- EsbGetMemberInfo
- EsbGetString
- EsbSetActive

EsbRemoveAlias

Permanently removes an alias table from the active database.

Syntax

EsbRemoveAlias (hCtx, AltName)
ByVal hCtx As Long
ByVal AltName As String

Parameter Description

hCtx VB API context handle.
AltName Name of an alias table to remove.

Notes

- "Default" or currently active alias table can not be removed.
- Make sure that no one else is using the same database as the one you try to remove an alias table from by calling EsbListConnections().

Return Value

None.
Access

This function requires the caller to have access to the database, and to have selected it as their active database using `EsbSetActive()`.

Example

Declare Function EsbRemoveAlias Lib "ESBAPIN" (ByVal hCtx As Long, ByVal Name As String) As Long

Sub ESB_RemoveAlias ()
    Dim sts As Long
    Dim Name As String   Name = "TestAlias"   '*************
    ' Remove Alias
    '*************
    sts = EsbRemoveAlias (hCtx, Name)
End Sub

See Also

- `EsbClearAliases`
- `EsbListAliases`
- `EsbLoadAlias`
- `EsbListConnections`
- `EsbSetActive`

**EsbRemoveLocks**

Removes all data block locks on a Database which are currently held by a user.

Syntax

```vbnet
EsbRemoveLocks (hCtx,AppName,DbName,LoginId)
```

**Parameter Description**

- **hCtx**  VB API context handle.
- **AppName**  Application name.
- **DbName**  Database name.
- **LoginId**  id of user login whose locks are to be removed.

Notes

- The required LoginId can be obtained from the user lock info structure returned by the `EsbListLocks` function.
- `EsbRemoveLocks()` terminates the connection of the user specified by `LoginId` if that user is currently logged in.
**Return Value**

None.

**Access**

This function requires the caller to have Database Design privilege (ESB_PRIV_DBDESIGN) for the specified database.

**Example**

Declare Function EsbRemoveLocks Lib "ESBAPIN" (ByVal hCtx As Long, ByVal AppName As String, ByVal DbName As String, ByVal LoginId As Long) As Long

Sub ESB_RemoveLocks ()
    Dim Items As Integer
    Dim AppName As String
    Dim DbName As String
    Dim LockInfo As ESB_LOCKINFO_T
    Dim sts As Long
    AppName = "Sample"
    DbName = "Basic"
    '*************
    ' List Locks
    '*************
    sts = EsbListLocks (hCtx, AppName, DbName, Items)
    '********************
    ' Remove all locks
    '********************
    For n = 1 To Items
        '*******************************
        ' Get next user lock structure
        ' from the list and remove locks
        '*******************************
        sts = EsbGetNextItem (hCtx,
                           ESB_LOCKINFO_TYPE, LockInfo)
        sts = EsbRemoveLocks (hCtx, AppName,
                           DbName, LockInfo.LoginId)
    Next
End Sub

**See Also**

- EsbListLocks

---

**EsbRenameApplication**

Renames an existing Application, either on the client or the server. If the Application is running on the server, it is first stopped.

**Syntax**

\[
\text{EsbRenameApplication} \ (hCtx, \ AppName, \ nAppName)
\]

ByVal hCtx As Long
ByVal AppName As String
ByVal nAppName As String
### EsbRenameApplication

Renames an existing application to a new name. Returns None.

#### Parameters
- **hCtx**: VB API context handle.
- **AppName**: Name of an existing application to rename.
- **nAppName**: New name of the application. See “Application Name Limits” on page 1727.

#### Example

```vba
Declare Function EsbRenameApplication Lib "ESBAPIN" (ByVal hCtx As Long, ByVal OldName As String, ByVal NewName As String) As Long

Sub ESB_RenameApplication ()
    Dim sts As Long
    Dim OldName As String
    Dim NewName As String
    OldName = "Sample"
    NewName = "NewSamp"  '********************
    ' Rename Application
    '********************
    sts = EsbRenameApplication (hCtx, OldName, NewName)
End Sub
```

#### See Also
- EsbRenameDatabase
- EsbRenameObject

### EsbRenameDatabase

Renames an existing database within an application, either on the client or the server. If the database is running on the server, it is first stopped.

#### Syntax

```
EsbRenameDatabase (hCtx,AppName,DbName,nDbName)
```

- **hCtx**: VB API context handle.
- **AppName**: Name of an existing application.
- **DbName**: Name of the database to rename.
- **nDbName**: New name of the database.
Parameter Description

hCtx VB API context handle. Can be local context handle returned from EsbCreateLocalContext().

AppName Application name.

DbName Name of an existing database to rename.

nDbName New name of the database. See “Database Name Limits” on page 1728.

Notes
Renaming a client database renames the local database directory.

Return Value
None.

Access
For a server database, the caller must have Database Create/Delete/Edit privilege (ESB_PRIV_DBCREATE).

Example
Declare Function EsbRenameDatabase Lib "ESBAPIN" (ByVal hCtx As Long, ByVal appName As String, ByVal oldValue As String, ByVal newValue As String) As Long

Sub ESB_RenameDatabase ()
    Dim sts As Long
    Dim AppName As String
    Dim OldName As String
    Dim NewName As String    AppName = "Sample"
    OldName = "Basic"
    NewName = "NewBasic"    ' Rename database
    '****************
    ' Rename database
    '****************
    sts = EsbRenameDatabase (hCtx, AppName, OldName, NewName)
End Sub

See Also
- EsbRenameApplication
- EsbRenameObject

EsbRenameFilter

Renames an existing filter.

Syntax
EsbRenameFilter (hCtx, appName, dbName, fltName, nFltName)
ByVal hCtx As Long
ByVal appName As String
ByVal dbName As String
ByVal FltName As String
ByVal nFltName As String

Parameter Description

hCtx VB API context handle.

AppName Application name.

DbName Database name.

FltName Old name of an existing filter to be renamed.

nFltName New name of the renamed filter. See “Filter Name Limits” on page 1728.

Notes

The old filter name must already exist, and the destination filter name must not exist.

Return Value

None.

Access

This function requires the caller to have Database Design privilege (ESB_PRIV_DBDESIGN) for the specified database.

Example

Declare Function EsbRenameFilter Lib "ESBAPIN" (ByVal hCtx As Long, ByVal AppName As String, ByVal DbName As String, ByVal OldName As String, ByVal NewName As String) As Long

Sub ESB_RenameFilter ()
    Dim sts As Long
    Dim AppName As String
    Dim DbName As String
    Dim OldName As String
    Dim NewName As String
    AppName = "Sample"
    DbName = "Basic"
    OldName = "Filter"
    NewName = "NewFilter"
    ' Rename Filter
    '***************
    sts = EsbRenameFilter (hCtx, AppName, DbName, OldName, NewName)
End Sub

See Also

- EsbCopyFilter
- EsbDeleteFilter
- EsbListFilters
- EsbSetFilter
**EsbRenameGroup**

Renames an existing group.

**Syntax**

\[
\text{EsbRenameGroup} \ (hCtx, \ GrpName, \ nGrpName)
\]

`ByVal \ hCtx` As Long

`ByVal \ GrpName` As String

`ByVal \ nGrpName` As String

**Parameter**

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
</tr>
<tr>
<td>GrpName</td>
</tr>
<tr>
<td>nGrpName</td>
</tr>
</tbody>
</table>

**Notes**

The specified new group name must not already exist.

**Return Value**

None.

**Access**

This function requires the caller to have Create/Delete User privilege (ESB_PRIV_USERCREATE) for the logged in server.

**Example**

```Visual Basic
Declare Function EsbRenameGroup Lib "ESBAPIN" (ByVal hCtx As Long, ByVal GroupName As String, ByVal nGrpName As String) As Long

Sub ESB_RenameGroup ()
    Dim sts As Long
    Dim OldName As String
    Dim NewName As String
    OldName = "PowerUsers"
    NewName = "NewUsers"  ;*************
    ' Rename Group  ;*************
    sts = EsbRenameGroup (hCtx, OldName, NewName)
End Sub
```

**See Also**

- [EsbCreateGroup](#)
- [EsbDeleteGroup](#)
- [EsbListGroups](#)

---

**EsbRenameObject**

Renames an existing object on the server or client object system.
Syntax

EsbRenameObject (hCtx, ObjType,AppName, DbName, ObjName, nObjName);
  ByVal hCtx As Long
  ByVal ObjType As Long
  ByVal AppName As String
  ByVal DbName As String
  ByVal ObjName As String
  ByVal nObjName As String

Parameter    Description

  hCtx       VB API context handle. Can be local context handle returned by EsbCreateLocalContext().
  ObjType    Object type (must be single type). Refer to Table 15 for a list of possible values.
  AppName   Application name.
  DbName     Database name. If an empty string, uses the application sub-directory.
  OldName   Old name of an object to rename.
  NewName   New name of the renamed object. See “Object Name Limits” on page 1728.

Notes

- To rename an object, the object must not be locked, and the new object must not already exist.
- Outline objects and LRO objects cannot be renamed.
- Use the EsbRenameDatabase() function to rename a database, including its associated outline.
- Objects cannot be renamed across different applications or databases. Use the EsbCopyObject() function to copy an object to another application or database.

Return Value

None.

Access

This function requires the caller to have application or database Design privilege (ESB_PRIV_APPDESIGN or ESB_PRIV_DBDESIGN) for the specified application or database containing the object.

Example

Declare Function EsbRenameObject Lib "ESBAPIN" (ByVal hCtx As Long, ByVal ObjType As Integer, ByVal AppName As String, ByVal DbName As String, ByVal OldName As String, ByVal NewName As String) As Long

Sub ESB_RenameObject ()
  Dim sts As Long
  Dim AppName As String
  Dim DbName As String
  Dim OldName As String
  Dim NewName As String

  '...
Dim ObjType As Integer   AppName = "Sample"
DbName = "Basic"
OldName = "Basic"
NewName = "NewBasic"
ObjType = ESB_OBJTYPE_OUTLINE   ' *******************
' Rename Rules Object
 ' *******************
    sts = EsbRenameObject (hCtx, ObjType, AppName,
    DbName, OldName, NewName)
End Sub

See Also

- EsbCopyObject
- EsbCreateObject
- EsbDeleteObject
- EsbListObjects
- EsbUnlockObject

EsbRenameUser

Renames an existing user.

Syntax

EsbRenameUser (hCtx, UserName, nUserName)
ByVal hCtx As Long
ByVal UserName As String
ByVal nUserName As String

Parameter Description

hCtx VB API context handle.
UserName Old name of an existing user to rename.
nUserName New name for the renamed user. See “User Name Limits” on page 1728.

Notes

The specified new user name must not already exist.

Return Value

None.

Access

This function requires the caller to have Create/Delete User privilege (ESB_PRIV_USERCREATE) for the logged in server.

Example

Declare Function EsbRenameUser Lib "ESBAPIN" (ByVal hCtx As Long, ByVal OldName As String, ByVal NewName As String) As Long
Sub ESB_RenameUser ()
    Dim sts As Long
    Dim OldName As String
    Dim NewName As String
    OldName = "Joseph"
    NewName = "Joe"  '************
    ' Rename user
    '************
    sts = EsbRenameUser (hCtx, OldName, NewName)
End Sub

See Also
- EsbCreateUser
- EsbDeleteUser
- EsbListUsers

**EsbReport**

Sends a report specification to the active database as a single string.

**Syntax**

```vbnet
EsbReport (hCtx, isOutput, isLock, rptQuery)
ByVal hCtx As Long
ByVal isOutput As Integer
ByVal isLock As Integer
ByVal rptQuery As String
```

**Parameter Description**

- **hCtx**: VB API context handle.
- **isOutput**: Controls output of data. If TRUE, data is output from the server, according to the specified report. If FALSE, no data is output.
- **isLock**: Controls block locking. If TRUE, all blocks which are accessed by the report specification are locked for update. If FALSE, no blocks are locked.
- **rptQuery**: The report specification, as a single string (must be less than 64 KB).

**Notes**

- This function is equivalent to making a call to `EsbBeginReport()`, followed by calls to `EsbSendString()` and finally `EsbEndReport()`. The report data can either be output, or the report specification can just be verified and any errors returned. Also, the corresponding data blocks in the Database can optionally be locked by this call (lock for update).
- The report specification string must be less than 64 KB long.
- If this function causes data to be output (Output flag is TRUE), the returned data can be read by calling `EsbGetString()` until an empty string is returned.
- If this function causes blocks to be locked (Lock flag is TRUE), the caller is responsible for unlocking the locked blocks (e.g. by calling `EsbUpdate()` with the Unlock flag set to TRUE).
If both the `Output` and `Lock` flags are set to FALSE, the Database merely performs a syntax check of the report specification.

**Return Value**
None.

**Access**
This function requires the caller to have read privilege (ESB_PRIV_READ) to one or more members in the active database. Any members that the caller does not have access to will be returned as missing.

**Example**

```vbnet
Declare Function EsbReport Lib "ESBAPIN" (ByVal hCtx As Long, ByVal Output As Integer, ByVal Lock As Integer, ByVal Query As String) As Long

Sub ESB_Report()
    Dim sts As Long
    Dim pOutput As Integer
    Dim pLock As Integer
    Dim Query As String
    Const szRString = 256
    Dim RString As String * szRString
    Query = "<Desc Year !"
    pOutput = ESB_YES
    pLock = ESB_NO
    ' Run Report
    ' *****************
    sts = EsbReport (hCtx, pOutput, pLock, Query) '********************
    ' Print out all strings
    ' ******************
    If sts = 0 Then
        sts = EsbGetString (hCtx, RString, szRString)
        Do While Mid$(RString, 1, 1) <> Chr$(0)
            Print RString
            sts = EsbGetString (hCtx, RString, szRString)
        Loop
    End If
End Sub
```

**See Also**

- `EsbBeginReport`
- `EsbEndReport`
- `EsbGetString`
- `EsbReportFile`
- `EsbUpdate`
- `EsbSendString`
**EsbReportFile**

Sends a report specification to the active database from a file. The report data can either be output, or the report specification can just be verified and any errors returned. Also, the corresponding data blocks in the Database can optionally be locked by this call (lock for update).

**Syntax**

```vbc
EsbReportFile (hDestCtx, hSrcCtx, AppName, DbName, FileName, isOutput, isLock)
```

- **hDestCtx** As Long
- **hSrcCtx** As Long
- **AppName** As String
- **DbName** As String
- **FileName** As String
- **isOutput** As Integer
- **isLock** As Integer

**Parameter Description**

- **hDestCtx**: VB API context handle of a target database on the server.
- **hSrcCtx**: VB API context handle for the report file location. The report file can reside on the client or on the same server as the target Database. If the report file is on the client (local), the local context must be created with `EsbCreateLocalContext`. 
- **AppName**: Application name for the report file location.
- **DbName**: Database name for the report file location.
- **FileName**: Name of the report specification file. It is not necessary to specify the file extension; the extension is understood to be `.rep`.
- **isOutput**: Controls output of data. If TRUE, data is output from the server, according to the specified report. If FALSE, no data is output.
- **isLock**: Controls block locking. If TRUE, all blocks which are accessed by the report specification are locked for update. If FALSE, no blocks are locked.

**Notes**

- If this function causes data to be output (**Output** flag is TRUE), the returned data can be read by calling `EsbGetString()`.
- If this function causes blocks to be locked (**Lock** flag is TRUE), the caller is responsible for unlocking the locked blocks (e.g. by calling `EsbUpdate()` with the **Unlock** flag set to TRUE).
- If both the **Output** and **Lock** flags are set to FALSE, the database merely performs a syntax check of the report specification.

**Return Value**

None.

**Access**

This function requires the caller to have read privilege (ESB_PRIV_READ) to one or more members in the active database.
Example

Declare Function EsbReportFile Lib "ESBAPIN" (ByVal hDestCtx As Long, ByVal hSrcCtx As Long, ByVal AppName As String, ByVal DbName As String, ByVal FileName As String, ByVal Output As Integer, ByVal Lock As Integer) As Long

Sub ESB_ReportFile ()
    Dim sts As Long
    Dim AppName As String
    Dim DbName As String
    Dim FileName As String
    Dim pOutput As Integer
    Dim pLock As Integer
    Dim hSrcCtx As Long
    Const szRString = 256
    Dim RString As String * szRString
    AppName = "Sample"
    DbName = "Basic"
    hSrcCtx = hCtx
    FileName = "test"
    pOutput = ESB_YES
    pLock = ESB_NO
    sts = EsbReportFile(hCtx, hSrcCtx, AppName, DbName, FileName, pOutput, pLock)
    If sts = 0 Then
        sts = EsbGetSTring(hCtx, RString, szRString)
        Do While Mid$(RString, 1, 1) <> Chr$(0)
            Print RString
            sts = EsbGetSTring(hCtx, RString, szRString)
        Loop
    End If
End Sub

See Also
- EsbBeginReport
- EsbGetString
- EsbReport
- EsbUpdateFile

EsbResetUser

Resets the user’s security structure to its initial state.

Syntax

    EsbResetUser (hCtx, UserName)

ByVal hCtx As Long
ByVal UserName As String

Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>API context handle</td>
</tr>
<tr>
<td>UserName</td>
<td>User name</td>
</tr>
</tbody>
</table>

Notes

The following user security parameters are reset to their initial state:
- LockedOut
- PwdChgNow
- Failcount
- LastLogin
- LastPwdChg
- Expiration

**Return Value**

Returns zero (0) if successful.

**Access**

This function requires the caller to have Create/Delete User privilege (ESB_PRIV_USERCREATE) for the logged in server.

**Example**

```visual-basic
Declare Function EsbResetUser Lib "esbapin" (ByVal hCtx As Long, _
ByVal UserName As String) As Long

Sub ESB_ResetUser()
    Dim UserName As String
    Dim sts As Long
    UserName = "William"
    sts = EsbResetUser(hCtx, UserName)
End Sub
```

**See Also**

- EsbInit
- EsbLogin
- EsbLogout

**EsbRestore**

No longer in use.

This function is retained for compatibility with earlier releases of Essbase only. For current Essbase archiving, see `EsbArchiveBegin()` and `EsbArchiveEnd()`. This function now returns the error message ESB_STS_OBSOLETE.

**See Also**

- EsbArchive
- EsbGetProcessState
- EsbSetActive
- EsbArchiveBegin
- EsbArchiveEnd
EsbSendString

Snds a string of data to the active database. The string must be less than 32 KB long. This function should be called after EsbBeginReport(), EsbBeginUpdate(), or EsbBeginCalc().

Syntax

EsbSendString (hCtx, sndString)
ByVal hCtx As Long
ByVal sndString As String

Parameter Description

hCtx VB API context handle.

sndString Data string (must be less than 32 KB in length).

Notes

- Calling this function other than after successfully executing a begin report, update or calculate function will generate an error.
- The string to be sent must be less than 32 KB long.
- When you are using this function with EsbBeginUpdate(), you must end the update string with a carriage return or line feed character.

Return Value

None.

Access

This function requires no special privileges.

Example

Declare Function EsbSendString Lib "ESBAPIN" (ByVal hCtx As Long, ByVal sndString As String) As Long

See the examples for EsbBeginReport and EsbBeginUpdate.

See Also

- EsbBeginCalc
- EsbBeginReport
- EsbBeginUpdate
- EsbGetString

EsbSetActive

Sets the caller’s active application and database.

Syntax

EsbSetActive (hCtx, AppName, DbName, pAccess)
ByVal hCtx As Long
ByVal AppName As String
ByVal DbName As String
ByVal pAccess As Integer

Parameter Description

hCtx VB API context handle.

AppName Application name.

DbName Database name.

pAccess Address of variable to receive the user's access level to the selected Database. See Table 15 for a list of possible values for this field.

Notes

- If the application and database have not been loaded, this function will load them.
- The EsbAutoLogin() function can also be used to allow a user to login and set the active application and database.

Return Value

If successful, returns the user's access level to the selected application and database in pAccess.

Access

This function requires no special privileges.

Example

Declare Function EsbSetActive Lib "ESBAPIN" (ByVal hCtx As Long, ByVal AppName As String, ByVal DbName As String, Access As Integer) As Long

Sub ESBSetActive()
    Dim AppName As String
    Dim DbName As String
    Dim pAccess As Integer
    Dim sts As Long
    AppName = "Demo"
    DbName = "Basic"
    '*****************************************************
    ' Set active Application & Database
    '*****************************************************
    sts = EsbSetActive (hCtx, AppName, DbName, pAccess)
End Sub

See Also

- EsbClearActive
- EsbGetActive
- EsbListApplications
- EsbListDatabases
- EsbLogin
- EsbSetActive
**EsbSetAlias**

Sets the active alias table in the active database for a user.

**Syntax**

```vbnet
EsbSetAlias (hCtx, AltName)
ByVal hCtx As Long
ByVal AltName As String
```

**Parameter Description**

- **hCtx**: VB API context handle.
- **AltName**: Name of alias table to set active.

**Return Value**

None.

**Example**

Declare Function EsbSetAlias Lib "ESBAPIN" (ByVal hCtx As Long, ByVal Name As String) As Long

Sub ESB_SetAlias()
    Dim sts As Long
    Dim pName As String
    pName = "TestAlias"
    '**********
    ' Set Alias
    '**********
    sts = EsbSetAlias (hCtx, pName)
End Sub

**See Also**

- [EsbGetAlias](#)
- [EsbListAliases](#)

**EsbSetApplicationAccess**

Sets a user application access structure, which contains information about user access to an application.

**Syntax**

```vbnet
EsbSetApplicationAccess (hCtx, Items, pUserApp)
ByVal hCtx As Long
ByVal Items As Long
ByVal pUserApp As ESB_USERAPP_T
```

**Parameter Description**

- **hCtx**: VB API context handle.
- **Items**: Reserved for future use.
**Parameter**  
**Description**

pUserApp  Pointer to a user application structure.

**Notes**

The *Access* field of the user application structure is used to set the user’s granted access to the application. For this call the *MaxAccess* field is ignored.

**Return Value**

None.

**Access**

This function requires the caller to have Application Design privilege (ESB_PRIV_APPDESIGN) for the specified application.

**Example**

```vbnet
Declare Function EsbSetApplicationAccess Lib "esbapin" (ByVal hCtx As Long, ByVal Items As Integer, UserApp As ESB_USERAPP_T) As Long

Sub Esb_SetApplicationAccess ()
    Dim sts As Long
    Dim Items As Integer
    Dim UserApp As ESB_USERAPP_T
    '****************************
    ' Initialize UserApp structure
    '****************************
    UserApp.UserName = "Joseph"
    UserApp.AppName = "Sample"
    UserApp.Access = ESB_ACCESS_SUPER
    UserApp.MaxAccess = ESB_ACCESS_SUPER
    '***************************
    ' Set Administrator access level
    '***************************
    sts = EsbSetApplicationAccess (hCtx, Items,
        UserApp)
End Sub
```

**See Also**

- EsbGetApplicationAccess
- EsbListUsers
- EsbSetDatabaseAccess
- EsbSetUser

**EsbSetApplicationState**

Sets user-configurable parameters for the application using the application’s state structure.

**Syntax**

```vbnet
EsbSetApplicationState (hCtx, AppName, pAppState)
ByVal hCtx As Long
```
ByVal **AppName**  As String  
ByVal **pAppState**  As ESB_PAPPSTATE_T

**Parameter**  

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>VB API context handle.</td>
</tr>
<tr>
<td>AppName</td>
<td>Application name.</td>
</tr>
<tr>
<td>pAppState</td>
<td>Pointer to application state structure.</td>
</tr>
</tbody>
</table>

**Notes**

- When changing parameter values, it is advisable to call `EsbGetApplicationState()` first to get the correct values of any parameters you do not wish to change. For example, here is a way to disable connects:

  ```vbnet
  Function ESB_DisableConnects(AppName As String) As Long
    Dim sts As Long
    Dim AppState As ESB_APPSTATE_T
    sts = EsbGetApplicationState(phCtx, AppName, AppState)
    If sts = 0 Then
      AppState.Connects = ESB_FALSE
      sts = EsbSetApplicationState(phCtx, AppName, AppState)
    End If
    ESB_SetApplicationState = sts
  End Function
  ```

- The following parameters do not apply to aggregate storage databases: *LockTimeout* and *lroSizeLimit*.

**Return Value**

None.

**Access**

This function requires the caller to have Application Manager privilege (ESB_PRIV_APPDESIGN) for the specified application.

**Example**

```vbnet
Declare Function EsbSetApplicationState Lib "ESBAPIN" (ByVal hCtx As Long, ByVal AppName As String, AppState As ESB_APPSTATE_T) As Long

Sub ESB_SetApplicationState()
    Dim sts As Long
    DimAppName As String
    Dim AppState As ESB_APPSTATE_T
    AppName = "Sample"
    AppState.Description = "This is a test application"
    AppState.Loadable = ESB_TRUE
    AppState.Autoload = ESB_TRUE
    AppState.Access = ESB_PRIV_APPCREATE
    AppState.Connects = ESB_TRUE
    AppState.Commands = ESB_TRUE
    AppState.Updates = ESB_TRUE
End Sub
```
AppState.Security = ESB_TRUE
AppState.LockTimeout = 1000    '******************************
' Set Application State structure
'******************************
sts = EsbSetApplicationState (hCtx, AppName, AppState)
End Sub

See Also

- EsbGetApplicationState
- EsbSetDatabaseState

EsbSetCalcList

Sets the list of calc scripts objects which are accessible to a user.

Syntax

EsbSetCalcList (hCtx, User, AppName, DbName, isAllCalcs, CalcList, Items)

ByVal hCtx As Long
ByVal User As String
ByVal AppName As String
ByVal DbName As String
ByVal isAllCalcs As Integer
ByVal CalcList As String
ByVal Items As Integer

Parameter Description

hCtx    VB API context handle.
User    User name.
AppName Application name.
DbName Database name. If an empty string, uses application sub-directory
isAllCalcs Allow all calcs flag. If TRUE, the user can access all calc scripts, otherwise, they can only access those specified in the CalcList argument.
CalcList String of calc script object names (CR, EOL delimited), the string must be less than 64 KB in length
Items Items of the number of accessible calc script objects in the CalcList string.

Notes

- If the AllCalcs flag is set to TRUE, the Items and pCalcList arguments will be ignored.
- In order to access any calc script objects, the user must have at least calculate access to the appropriate database.

Return Value

None.
**Access**

This function requires the caller to have Database Design privilege (ESB_PRIV_DBDESIGN) for the specified database.

**Example**

```vbscript
Declare Function EsbSetCalcList Lib "ESBAPIN" (ByVal hCtx As Long, ByVal User As String, ByVal AppName As String, ByVal DbName As String, ByVal isAllCalcs As Integer, ByVal CalcList As String, ByVal Items As Integer) As Long

Sub EsbSetCalcList ()
    Dim sts As Long
    Dim Items As Integer
    Dim User As String
    Dim AppName As String
    Dim DbName As String
    Dim AllCalcs As Integer
    Dim CalcList As String   User = "Joseph"
    AppName = "Sample"
    DbName = "Basic"
    AllCalcs = ESB_NO   '****************************
    ' Initialize CalcList and Items
    '******************************
.
   '**************
   ' Set Calc list
   '**************
    sts = EsbSetCalcList (hCtx, User, AppName, DbName, AllCalcs, CalcList, Items)
End Sub
```

**See Also**

- EsbGetCalcList
- EsbListObjects
- EsbListUsers

**EsbSetDatabaseAccess**

Sets a user database access structure, which contains information about user access to a database.

**Syntax**

```vbscript
EsbSetDatabaseAccess (hCtx, Items, pUserDb)
ByVal hCtx As Long
ByVal Items As Integer
ByVal pUserDb As ESB_USERDB_T
```

**Parameter Description**

- **hCtx**  
  VB API context handle.
- **Items**  
  Reserved for future use.
- **pUserDb**  
  Pointer to a user database structure.
Notes

The *Access* field of the user database structure is used to set the user's granted access to the database. For this call the *MaxAccess* and *FilterName* fields are ignored.

Return Value

None.

Access

This function requires the caller to have Database Manager privilege (ESB_PRIV_DBDESIGN) for the specified database.

Example

Declare Function EsbSetDatabaseAccess Lib "ESBAPIN" (ByVal hCtx As Long, ByVal Items As Integer, UserDb As ESB_USERDB_T) As Long

Sub EsbSetDatabaseAccess ()
    Dim sts As Long
    Dim hCtx As Long
    Dim Items As Integer
    Dim UserDb As ESB_USERDB_T '*****************************
    ' Initialize UserDb structure
    '*****************************
    UserDb.UserName = "Joseph"
    UserDb.AppName = "Sample"
    UserDb.DbName = "Basic"
    UserDb.Access = ESB_ACCESS_SUPER
    UserDb.MaxAccess = ESB_ACCESS_SUPER
    UserDb.FilterName = ""   '***************************
    ' Set Administrator access level
    '***************************
    sts = EsbSetDatabaseAccess (hCtx, Items, UserDb)
End Sub

See Also

- EsbGetDatabaseAccess
- EsbListUsers
- EsbSetApplicationAccess
- EsbSetUser

EsbSetDatabaseNote

Sets a database's note-of-the-day message. This message may be used to display useful information about the database (whether data has been loaded, when it was last calculated, etc.) to users before they connect to the database.

Syntax

EsbSetDatabaseNote (hCtx, AppName, DbName, DbNote)
ByVal hCtx As Long
ByVal AppName As String
Parameter Description

hCtx VB API context handle.
AppName Application name.
DbName Database name.
DbNote Pointer to a database note string.

Notes
The database note string must be less than 64 KB in length.

Return Value
None.

Access
This function requires the caller to have Database Design privilege (ESB_PRIV_DBDESIGN) for the specified database.

Example
Declare Function EsbSetDatabaseNote Lib "ESBAPIN" (ByVal hCtx As Long, ByVal AppName As String, ByVal DbName As String, ByVal DbNote As String) As Long

Sub ESB_SetDatabaseNote ()
    Dim sts As Long
    Dim AppName As String
    Dim DbName As String
    Dim DbNote As String
    AppName = "Sample"
    DbName = "Basic"
    DbNote = "This is a test"
    '******************
    ' Set Database note
    '******************
    sts = EsbSetDatabaseNote (hCtx, AppName, DbName, DbNote)
End Sub

See Also
- EsbGetDatabaseNote

**EsbSetDatabaseState**
Sets user-configurable parameters for the database using the database's state structure.

Syntax

```
EsbSetDatabaseState (hCtx, AppName, DbName, pDbState)
ByVal hCtx As Long
```
ByVal AppName As String
ByVal DbName As String
ByVal pDbState As ESB_PDBSTATE_T

**Parameter Description**

hCtx  VB API context handle.

AppName  Application name.

DbName  Database name.

pDbState  Pointer to a database state structure.

**Notes**

- EsbGetDatabaseState() should be called to initialize the ESB_DBSTATE_T structure before EsbSetDatabaseState() is called.
- This function can only set user-configurable parameters for server databases.

**Return Value**

None.

**Access**

This function requires the caller to have Database Design privilege (ESB_PRIV_DBDESIGN) for the specified database.

**Example**

Declare Function EsbSetDatabaseState Lib "ESBAPIN" (ByVal hCtx As Long, ByVal AppName As String, ByVal DbName As String, DbState As ESB_DBSTATE_T) As Long

Sub ESB_SetDatabaseState ()
    Dim sts As Long
    Dim AppName As String
    Dim DbName As String
    Dim DbState As ESB_DBSTATE_T
    AppName = "Sample"
    DbName = "Basic"
    ' Initialize DbState structure fields
    DbState.Description = "This is Sample/Basic"
    DbState.Loadable = "1"
    ' Set Database state structure
    sts = EsbSetDatabaseState (hCtx, AppName, DbName, DbState)
End Sub

**See Also**

- EsbGetDatabaseState
- EsbSetApplicationState

---

1414 Visual Basic Main API Functions
**EsbSetDefaultCalc**

Sets the default calc script for the active database.

**Syntax**

```vbc
EsbSetDefaultCalc (hCtx, cscString)
ByVal hCtx As Long
ByVal cscString As String
```

**Parameter Description**

- **hCtx**: VB API context handle.
- **cscString**: Default calc script string.

**Notes**

The calc script string must be less than 64 KB long.

**Return Value**

None.

**Access**

This function requires the caller to have calc privilege (ESB_PRIV_CALC) to the active database.

**Example**

```vbc
Declare Function EsbSetDefaultCalc Lib "ESBAPIN" (ByVal hCtx As Long, ByVal Script As String) As Long

Sub ESB_SetDefaultCalc ()
    Dim sts As Long
    Dim Script As String
    Script = "CALC ALL;" '***************
    ' Set default calc
    '***************
    sts = EsbSetDefaultCalc (hCtx, Script)
End Sub
```

**See Also**

- `EsbDefaultCalc`
- `EsbGetDefaultCalc`
- `EsbSetDefaultCalcFile`
- `EsbSetActive`

---

**EsbSetDefaultCalcFile**

Sets the default calc script for the active database from a calc script file.

**Syntax**

```vbc
EsbSetDefaultCalcFile (hDestCtx, hSrcCtx, AppName, DbName, FileName)
ByVal hDestCtx As Long
ByVal hSrcCtx As Long
```

---

Visual Basic Main API Function Reference 1415
ByVal AppName As String
ByVal DbName As String
ByVal FileName As String

Parameter Description

hDestCtx VB API context handle of a target database on the server.

hSrcCtx VB API context handle for the calc script file location. The calc script file can reside on the client or on the same server as the target database.

AppName Application name for the calc script file location.

DbName Database name for the calc script file location.

FileName Name of the default calc script file.

Notes

The default calc script must not be greater than 64 KB long.

Return Value

None.

Access

This function requires the caller to have calc privilege (ESB_PRIV_CALC) to the active database.

Example

Declare Function EsbSetDefaultCalcFile Lib "ESBAPIN" (ByVal hDestCtx As Long, ByVal hSrcCtx As Long, ByVal AppName As String, ByVal DbName As String, ByVal FileName As String) As Long

Sub ESB_SetDefaultCalcFile ()
    Dim sts As Long
    Dim AppName As String
    Dim DbName As String
    Dim FileName As String
    Dim hSrcCtx As Long
    AppName = "Sample"
    DbName = "Basic"
    'Calc script is an object at the server *
    hSrcCtx = hCtx
    FileName = "calc"
    'Calc File
    sts = EsbSetDefaultCalcFile (hCtx, hSrcCtx, AppName, DbName, FileName)
End Sub

See Also

- EsbDefaultCalc
- EsbGetDefaultCalc
- EsbSetDefaultCalc
EsbSetFilter

Starts setting the contents of a filter.

Syntax

```vba
EsbSetFilter (hCtx, AppName, DbName, FltName, isActive, pAccess)
```

**Parameter Description**

- **hCtx** VB API context handle.
- **AppName** Application name.
- **DbName** Database name.
- **FltName** Filter name. See “Filter Name Limits” on page 1728.
- **isActive** Filter active flag. If TRUE, the filter is set active, otherwise it is set inactive.
- **pAccess** The default filter access level.

**Notes**

- If the filter does not already exist, it will first be created by this call.
- This call must be followed by successive calls to `EsbSetFilterRow()` to set all the rows for the filter.

**Return Value**

None.

**Access**

This function requires the caller to have Database Design privilege (ESB_PRIV_DBDESIGN) for the specified database.

**Example**

```vba
Declare Function EsbSetFilter Lib "ESBAPIN" (ByVal hCtx As Long, ByVal AppName As String, ByVal DbName As String, ByVal FltName As String, ByVal isActive As Integer, ByVal pAccess As Integer) As Long

Sub ESB_SetFilter()
    Dim sts As Long
    Dim AppName As String
    Dim DbName As String
    Dim FltName As String
    Dim Active As Integer
    Dim pAccess As Integer
    Dim Row As String
    AppName = "Demo"
    DbName = "Basic"
```

Visual Basic Main API Function Reference 1417
FilterName = "Filter"
Active = ESB_YES
pAccess = ESB_ACCESS_DBCREATE '***********
' Set Filter
'***********
sts = EsbSetFilter(hCtx, AppName, DbName,
  FilterName, Active, pAccess)
pAccess = ESB_ACCESS_READ
Row = "@IDESCENDANTS(Scenario)"
sts = EsbSetFilterRow(hCtx, Row, pAccess)
pAccess = ESB_ACCESS_WRITE
Row = "@IDESCENDANTS(Scenario); East"
sts = EsbSetFilterRow(hCtx, Row, pAccess)
sts = EsbSetFilterRow(hCtx, ByVal 0&, pAccess)
End Sub

See Also
- EsbGetFilter
- EsbListFilters
- EsbSetFilterRow

EsbSetFilterList

Sets the list of groups or users that are assigned to a filter. The Items parameter controls the number of groups or users assigned to the filter. A Items of zero will remove all the groups or users from the list.

Syntax

```
EsbSetFilterList (hCtx, AppName, DbName, FltName, UserList, Items)
ByVal hCtx As Long
ByVal AppName As String
ByVal DbName As String
ByVal FltName As String
ByVal UserList As String
ByVal Items As Integer
```

Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>VB API context handle.</td>
</tr>
<tr>
<td>AppName</td>
<td>Application name.</td>
</tr>
<tr>
<td>DbName</td>
<td>Database name.</td>
</tr>
<tr>
<td>FltName</td>
<td>Filter name.</td>
</tr>
<tr>
<td>UserList</td>
<td>String of user names (CR and EOL delimited), must be less than 64 KB in length.</td>
</tr>
<tr>
<td>Items</td>
<td>Number of user names in the string.</td>
</tr>
</tbody>
</table>

Return Value

None.
**Access**

This function requires the caller to have Database Design privilege (ESB_PRIV_DBDESIGN) for the specified database.

**Example**

Declare Function EsbSetFilterList Lib "ESBAPIN" (ByVal hCtx As Long, ByVal AppName As String, ByVal DbName As String, ByVal FltName As String, ByVal UserList As String, ByVal Items As Integer) As Long

Sub ESB_SetFilterList ()
    Dim sts As Long
    Dim AppName As String
    Dim DbName As String
    Dim FilterName As String
    Dim UserList As String
    Dim Items As Integer   AppName = "Sample"
    DbName = "Basic"
    FilterName = "Filter"
    '****************************
    ' Initialize UserList and Items
    '****************************
    Items = 2
    UserList = "Admin"+Chr$(13)+Chr$(10)+"Truc"
    '********************
    ' Set Filter List
    '********************
    sts = EsbSetFilterList (hCtx, AppName, DbName, FilterName, UserList, Items)
End Sub

See Also

- EsbGetFilterList
- EsbListFilters
- EsbSetFilter

**EsbSetFilterRow**

Sets the next row of a filter.

**Syntax**

`EsbSetFilterRow (hCtx, FltRow, pAccess)`

*ByVal* **hCtx**  As Long  
*ByVal* **FltRow**  As Any  
*ByVal* **pAccess**  As Integer

**Parameter Description**

- **hCtx**  VB API context handle.
- **FltRow**  Pointer to the next row of the filter.
- **pAccess**  Access level for the filter row.
Notes
This function should be called repeatedly after calling EsbSetFilter(), once for each row of the filter, terminating the row list with NULL.

Return Value
None.

Access
This function requires the caller to have Database Design privilege (ESB_PRIV_DBDESIGN) for the specified database.

Example
Declare Function EsbSetFilterRow Lib "ESBAPIN" (ByVal hCtx As Long, ByVal FltRow As Any, ByVal pAccess As Integer) As Long

See the example for EsbSetFilter.

See Also
- EsbListFilters
- EsbSetFilter

EsbSetGlobalState
Sets the server global state structure which contains parameters for system administration.

Syntax
EsbSetGlobalState (hCtx, pGlobal)
ByVal hCtx As Long
pGlobal As ESB_GLOBAL_T

Parameter Description
hCtx VB API context handle.
pGlobal Pointer to global state structure.

Notes
When changing parameter values, it is advisable to call EsbGetGlobalState() first to get the correct values of any parameters you do not wish to change.

Return Value
None.

Access
This function requires the caller to be an administrator.
Example

Declare Function EsbSetGlobalState Lib "ESBAPIN" (ByVal hCtx As Long, Global As ESB_GLOBAL_T) As Long

Sub ESB_SetGlobalState ()
    Dim sts As Long
    Dim pGlobal As ESB_GLOBAL_T   pGlobal.Security = ESB_TRUE
    pGlobal.Logins = ESB_TRUE
    pGlobal.Access = ESB_ACCESS_READ
    pGlobal.Validity = 14
    pGlobal.Currency = ESB_FALSE
    pGlobal.PwMin = 6
    pGlobal.InactivityTime = 300
    pGlobal.InactivityCheck = 40   '*****************
    ' Set Global State
    '*****************
    sts = EsbSetGlobalState (hCtx, pGlobal)
End Sub

See Also

- EsbGetGlobalState

EsbSetGroup

Sets a group information structure, which contains security information for the group.

Syntax

EsbSetGroup (hCtx, pUserInfo)
ByVal hCtx As Long
    pUserInfo As ESB_USERINFO_T

Parameter Description

hCtx   VB API context handle.

pUserInfo  Pointer to group info structure.

Notes

- The name of the group to set is a field in the group info structure, which must always be specified.

- The only field in the group info structure which may be changed using this function is the Access field (the other fields are used for users of for information only). See the description of the ESB_GROUPINFO_T structure for more information.

Return Value

None.

Access

This function requires the caller to have Create/Delete User privilege (ESB_PRIV_USERCREATE) for the logged in server.
Example

Declare Function EsbSetGroup Lib "ESBAPIN" (ByVal hCtx As Long, GroupInfo As ESB_USERINFO_T) As Long

Sub ESB_SetGroup ()
    Dim sts As Long
    Dim GroupInfo As ESB_USERINFO_T '*******************************
    ' Initialize GroupInfo structure
    '*******************************
    GroupInfo.Name = "PowerUsers"
    GroupInfo.Type = ESB_TYPE_GROUP
    GroupInfo.Access = ESB_PRIV_APPCREATE
    GroupInfo.MaxAccess = ESB_PRIV_APPCREATE '************************
    ' Set GroupInfo structure
    '************************
    sts = EsbSetGroup (hCtx, GroupInfo)
End Sub

See Also

● EsbGetGroup
● EsbListGroups

EsbSetGroupList

Sets the list of users who are members of a group.

Syntax

**EsbSetGroupList** (hCtx, GrpName, GrpList, Items)

ByVal hCtx As Long
ByVal GrpName As String
ByVal GrpList As String
ByVal Items As Integer

Parameter Description

hCtx VB API context handle.

GrpName Group or user name.

GrpList String of user names (EOL, CR delimited), must be less than 64 KB in length.

Items Items of user names.

Notes

To set the list of groups to which a user belongs, enter a user name as the **GroupName** argument and pass a list of groups as the **UserList** argument.

Return Value

None.
**Access**

This function requires the caller to have Create/Delete User privilege (ESB_PRIV_USERCREATE) for the logged in server.

**Example**

Declare Function EsbSetGroupList Lib "ESBAPIN" (ByVal hCtx As Long, ByVal GroupName As String, ByVal UserList As String, ByVal Items As Integer) As Long

Sub ESB_SetGroupList()
    Dim sts As Long
    Dim GroupName As String
    Dim UserList As String
    Dim Items As Integer
    Dim CRLF As String
    CRLF = Chr$(13) + Chr$(10)
    GroupName = "PowerUsers"  '**************************
    ' Initialize UserList and Items
    '**************************
    Items = 2
    UserList = "Admin" + CRLF + "Bob"  '***************
    ' Set Group List
    '***************
    sts = EsbSetGroupList (hCtx, GroupName, UserList, Items)
End Sub

**See Also**

- EsbAddToGroup
- EsbDeleteFromGroup
- EsbListGroups
- EsbSetGroupList

**EsbSetPassword**

Sets a user’s password, erasing the existing password.

**Syntax**

\[ \texttt{EsbSetPassword} (hCtx, userName, Password) \]

\begin{verbatim}
ByVal hCtx As Long
ByVal userName As String
ByVal Password As String
\end{verbatim}

**Parameter Description**

- **hCtx**: VB API context handle.
- **userName**: User name.
- **Password**: New password for a user.
Notes

- To change a password, the caller must either have administrator access, or be changing their own password.
- The new password takes effect the next time the user logs in.

Return Value

None.

Access

This function requires the caller to have Create/Delete User privilege (ESB_PRIV_USERCREATE) for the logged in server, unless they are setting their own password.

Example

Declare Function EsbSetPassword Lib "ESBAPIN" (ByVal hCtx As Long, ByVal UserName As String, ByVal Password As String) As Long

Sub ESB_SetPassword ()
  Dim sts As Long
  Dim UserName As String
  Dim Password As String
  UserName = "Joseph"
  Password = "NewPassword"
  ' *****************
  ' Set New password
  ' *****************
  sts = EsbSetPassword (hCtx, UserName, Password)
End Sub

See Also

- EsbListUsers
- EsbSetUser

EsbSetPath

Sets the ESSBASEPATH environment variable for the runtime process.

Syntax

EsbSetPath (Path)
ByVal Path As String

Parameter Description

Path String describing the ESSBASEPATH environment variable

Notes

- Call EsbSetPath() before calling EsbInit().
- Path cannot exceed 120 characters, as defined in ESB_PATHLEN.
- Path applies only to the current process.
Essbase DLLs must be accessible from the system path. **EsbSetPath()** does not resolve the path for the Essbase DLLs.

**Return Value**

- If successful, returns ESB_STS_NOERR.
- If *Path* is too long, returns API_NAME_TOO_LONG (1030009).

**Example**

```vbnet
Sub ESB_SetPath
    Dim sts As Long
    Dim Path As String

    Path = "C:\Hyperion\products\Essbase"
    sts  = EsbSetPath(Path)
End Sub
```

**EsbSetSpanRelationalSource**

Sets the Boolean bSpanRelPart field informing Essbase that pertinent data exists in an attached relational store. Some other API functions, such as EsbQueryDatabaseMembers, read bSpanRelPart and access the relational store if bSpanRelPart is set.

**Syntax**

```vbnet
Declare Function EsbSetSpanRelationalSource Lib "esbapin" (ByVal hCtx As Long) As Long
```

**Parameter Description**

- **hCtx** - API context handle.

**Notes**

Several API functions have been enhanced to retrieve information from relational stores.

- **EsbQueryDatabaseMembers** - returns member names from the relational store.
- **EsbGetMemberInfo** - returns information on members in the relational store.
- **EsbCheckMemberName** - checks in the relational store for valid member names.
- **EsbGetMemberCalc** - recognizes a relational member passed as input and returns a null string for all relational members.

**Return Value**

None.

**Access**

This function requires the caller to have read privilege (ESS_PRIV_READ) to one or more members in the active database.
Example

Declare Function EsbSetRelationalSource Lib "ESBAPIN" (ByVal hCtx As Long) As Long

Sub ESB_SetRelationalSource()
    Dim sts As Long

    '*************************
    ' Set the bSpanRelPart field
    '*************************
    sts = EsbSetRelationalSource (hCtx)

End Sub

See Also

- EsbClrSpanRelationalSource

EsbSetUser

Sets a user information structure, which contains security information for the user.

Syntax

EsbSetUser (hCtx, pUserInfo)
ByVal hCtx As Long
    pUserInfo As ESB_USERINFO_T

Parameter Description

<table>
<thead>
<tr>
<th>hCtx</th>
<th>VB API context handle.</th>
</tr>
</thead>
<tbody>
<tr>
<td>pUserInfo</td>
<td>Pointer to user info structure.</td>
</tr>
</tbody>
</table>

Notes

- The name of the user to set is a field in the user info structure, which must always be specified.
- The only fields in the user info structure which may be changed using this function are the Access, Expiration, and PwdChgNow fields (the other fields are for information only). See “ESB_USERINFO_T, ESB_GROUPINFO_T” on page 1193 for more information.
- The caller cannot give the specified user any access privileges that they themselves do not already have.
- The new user settings will take effect the next time the user logs in.

Return Value

None.

Access

This function requires the caller to have Create/Delete User privilege (ESB_PRIV_USERCREATE) for the logged in server.
Example

Declare Function EsbSetUser Lib "ESBAPIN" (ByVal hCtx As Long, UserInfo As ESB_USERINFO_T) As Long

Sub ESB_SetUser ()
    Dim sts As long
    Dim UserInfo As ESB_USERINFO_T  '****************************
    ' Initialize fields for UserInfo
    '****************************
    UserInfo.Name = "Joseph"
    UserInfo.Type = ESB_TYPE_USER
    UserInfo.Access = ESB_ACCESS_SUPER
    UserInfo.MaxAccess = ESB_ACCESS_SUPER
    '****************************
    ' Set User Info structure
    '****************************
    sts = EsbSetUser (hCtx, UserInfo)
End Sub

See Also

- EsbGetUser
- EsbListUsers
- EsbSetApplicationAccess
- EsbSetPassword

EsbSetUserEx

Sets a user information structure, which contains security information for the user.

Syntax

Declare Function EsbSetUserEx Lib "esbapin" (ByVal hCtx As Long, pUserInfo As ESB_USERINFOEX_T) As Long

Parameter Description

hCtx API context handle.

pUserInfoEx Pointer to info structure of externally authenticated user.

Notes

- This function operates similarly to EsbSetUser. The difference is that this function sets the extended user information structure ESB_USERINFOEX_T.
- The name of the user to set is a field in the user info structure, which must always be specified.
- The only fields in the user info structure which may be changed using this function are the Access, Expiration, and PwdChgNow fields (the other fields are for information only). For more information, see “ESB_USERINFO_T, ESB_GROUPINFO_T” on page 1193.
- The caller cannot give the specified user any access privileges that they themselves do not already have.
- The new user settings will take effect the next time the user logs in.
Return Value

Returns zero (0) if successful.

Access

This function requires the caller to have Create/Delete User privilege (ESS_PRIV_USERCREATE) for the logged in server.

EsbShutdownServer

Stops the Agent.

Syntax

EsbShutdownServer (hInst, Server, User, Password)
ByVal hInst As Long
ByVal Server As String
ByVal User As String
ByVal Password As String

Parameter Description

hInst VB API instance handle.
Server Network server name string. Specifies the name of the server to shut down.
User User name string. Specifies the user who is requesting the shutdown.
Password Password string. Specifies the password of the user requesting the shutdown.

Notes

- This function sends a request to the Agent (ESSBASE.EXE) to shut itself down. The Agent then goes through its normal shutdown procedure, including committing data, stopping all applications and databases, and logging users off before stopping.
- Only users with Administrator privilege can shut down the Agent.
- This function can be called at any time, however, it is normally called to shut down an Agent process which was started in the background. See the Oracle Essbase Database Administrator’s Guide for details.

Return Value

None.

Possible error conditions resulting from this function include:

- Insufficient privilege for this operation, AD_AMSG_IPO
- Incorrect password, AD_AMSG_IPW
- User does not exist, AD_AMSG_UNE
- Cannot shutdown application, AD_MSGAR_NOSHUTDOWN
- Network Error: Unable To Locate In Hosts File, NET_TCP_HOSTS
Network error: Cannot locate server, NET_NP_NOSERVER

Access

This function requires administrator privilege.

Example

Declare Function EsbShutdownServer Lib "ESBAPIN" (ByVal hInst As Long, ByVal Server As String, ByVal UserName As String, ByVal Password As String) As Long

Sub ESB_ShutdownServer()
    Dim sts As Long
    Dim Server As String
    Dim UserName As String
    Dim Password As String
    Server = "Rainbow"
    UserName = "Admin"
    Password = "password"
    ' Shut down Server
    '******************
    sts = EsbShutdownServer(hInst, Server, UserName, Password)
End Sub

See Also

- EsbSetPassword
- EsbUnloadApplication
- EsbUnloadDatabase

EsbTerm

Terminates the VB API and releases all system resources used by the VB API.

Syntax

EsbTerm (hInst)
ByVal hInst As Long

Parameter Description

hInst VB API instance handle.

Notes

- This function should normally be called after all other VB API calls have been completed, immediately prior to terminating your program.
- Because this function terminates use of the VB API, any VB API functions (other than EsbInit()) called after this function has been executed will return an error.

Return Value

None.

Access

This function requires no special access.
Example

Declare Function EsbTerm Lib "ESBAPIN" (ByVal hInst As Long) As Long
Sub ESB_Term ()
    Dim sts As Long   '*****************************
    ' Terminate the VB API
    '*****************************
    sts = EsbTerm (hInst)
End Sub

See Also

- EsbInit

EsbUnloadApplication

Stops an application on the server.

Syntax

EsbUnloadApplication (hCtx, AppName)
ByVal hCtx As Long
ByVal AppName As String

Parameter Description

hCtx VB API context handle.

AppName Name of an application to unload.

Notes

- To unload an application, the connected user must have load access to the application.
- An application cannot be unloaded if Essbase Server is restructuring a database associated with the application.

Return Value

None.

Access

This function requires the caller to have Application Load/Unload privilege (ESB_PRIV_APPLOAD) for the specified application.

Example

Declare Function EsbUnloadApplication Lib "ESBAPIN" (ByVal hCtx As Long, ByVal AppName As String) As Long
Sub ESB_UnloadApplication ()
    Dim sts As Long
    Dim AppName As String    AppName = "Sample"   '*****************
    ' Unload Application
    '*****************
    sts = EsbUnloadApplication (hCtx, AppName)
End Sub
EsbUnloadDatabase

Stops a database within an application on the server.

Syntax

```
EsbUnloadDatabase (hCtx,AppName,DbName)
ByVal hCtx As Long
ByVal AppName As String
ByVal DbName As String
```

Parameter Description

- **hCtx**: VB API context handle.
- **AppName**: Application name.
- **DbName**: Name of a database to unload.

Return Value

None.

Access

This function requires the caller to have Database Load/Unload privilege (ESB_PRIV_APPLOAD).

Example

```
Declare Function EsbUnloadDatabase Lib "ESBAPIN" (ByVal hCtx As Long, ByVal AppName As String, ByVal DbName As String) As Long

Sub ESB_UnloadDatabase ()
    Dim sts As Long
    Dim AppName As String
    Dim DbName As String    AppName = "Sample"
    DbName = "Basic"        '***************
    ' Unload Database
    '***************
    sts = EsbUnloadDatabase (hCtx, AppName, DbName)
End Sub
```

See Also

- EsbLoadApplication
- EsbUnloadDatabase

EsbUnlockObject

Unlocks a locked object on the server or client object system.
Syntax

`EsbUnlockObject (hCtx, ObjType, AppName, DbName, ObjName)`

ByVal hCtx As Long
ByVal ObjType As Long
ByVal AppName As String
ByVal DbName As String
ByVal ObjName As String

Parameter Description

hCtx VB API context handle. Can be local context handle returned by EsbCreateLocalContext().

ObjType Object type (must be single type). Refer to Table 15 for a list of possible values.

AppName Application name.

DbName Database name. If an empty string, uses the application sub-directory.

ObjName Name of an object to unlock.

Notes

To unlock an object, the object must already exist and be locked by the caller.

Return Value

None.

Access

This function requires the caller to have Application or Database Design privilege (ESB_PRIV_APPDESIGN or ESB_PRIV_DBDESIGN) for the specified application or database containing the object.

Example

Declare Function EsbUnlockObject Lib "ESBAPIN" (ByVal hCtx As Long, ByVal ObjType As Integer, ByVal AppName As String, ByVal DbName As String, ByVal ObjName As String) As Long

Sub ESB_UnlockObject ()
    Dim sts As Long
    Dim AppName As String
    Dim DbName As String
    Dim ObjName As String
    Dim ObjType As Integer
    AppName = "Sample"
    DbName = "Basic"
    ObjName = "Basic"
    ObjType = ESB_OBJTYPE_OUTLINE '***************
    ' UnLock Rules Object
    '***************
    sts = EsbUnlockObject (hCtx, ObjType, AppName, DbName, ObjName)
End Sub

See Also

- EsbGetObject
EsbUpdate

Sends an update specification to the active database as a single string.

Syntax

EsbUpdate (hCtx, isStore, isUnlock, updQuery)
ByVal hCtx As Long
ByVal isStore As Integer
ByVal isUnlock As Integer
ByVal updQuery As String

Parameter Description

hCtx VB API context handle.

isStore Controls storage of data. If TRUE, data is stored in the server; if FALSE, no data is stored.

isUnlock Controls unlocking of data blocks. If TRUE, all relevant blocks which are locked will be unlocked (after data is stored, if necessary). If FALSE, no blocks are unlocked.

updQuery The update specification, as a single string (must be less than 32 KB).

Notes

- This function is equivalent to making a call to EsbBeginUpdate(), followed by calls to EsbSendString() and finally EsbEndUpdate(). The updated data can either be stored in the database, or just verified and any errors returned. Also, any data blocks locked for update can be unlocked by this call.

- The update specification string must be less than 32 KB long.

- If this function causes data to be stored (Store flag is TRUE), the relevant data blocks must previously have been locked for update (e.g. by calling EsbReport() with the Lock flag set to TRUE).

- If the caller attempts to write data to a member it does not have permission to write to, a warning is generated, and the member is not updated.

- If both the Store and Unlock flags are set to FALSE, the database merely performs a syntax check of the update specification.

Return Value

None.

Access

This function requires the caller to have write privilege (ESB_PRIV_WRITE) to the active database.
Example

Declare Function EsbUpdate Lib "ESBAPIN" (ByVal hCtx As Long, ByVal Store As Integer, ByVal Unlock As Integer, ByVal Query As String) As Long

Sub ESB_Update ()
    Dim sts As Long
    Dim Store As Integer
    Dim pUnlock As Integer
    Dim Query As String
    Query = "Year Market Scenario AcItemss Product 12345"
    Store = ESB_YES
    pUnlock = ESB_NO
    '******
    ' Update
    '******
    sts = EsbUpdate (hCtx, Store, pUnlock, Query)
End Sub

See Also
- EsbBeginUpdate
- EsbEndUpdate
- EsbReport
- EsbSendString
- EsbUpdateFile

EsbUpdateDrillThruURL

Updates a drill-through URL, with the given name, within the active database outline.

“Drill-through URL Limits” on page 1729.

Syntax

Declare Function EsbUpdateDrillThruURL Lib "esbapin" (ByVal hCtx As Long, ByRef symRegions() As String, ByRef pUrl As ESB_DURLINFO_T, ByVal bMerge As Integer) As Long

Parameter Description

hCtx Visual Basic API context handle

symRegions() Array containing the symmetric region specification

pUrl URL definition

bMerge
- If True, add drill-through region definitions in pUrl to the existing list of drill-through regions in the named URL definition
- If False, replace the existing list of drill-through region definitions with the list in pUrl

Return Value
- If successful, updates the named drill-through URL in the active database by replacing the URL XML and either updating or replacing the drill-through region list with the corresponding fields in pUrl.
- If there is no URL with the given name, returns an error code.
Access

- Caller must have database Design privilege (ESB_PRIV_DBDESIGN) for the specified database.
- Caller must have selected the specified database as their active database using EsbSetActive().

Example

```vba
Sub ESB_UpdateGLDrillThru()
    Dim sts As Long
    Dim url As ESB_DURLINFO_T
    Dim cppDrillRegions(0 To 1) As String
    Dim bMerge As Integer

    '***************************************************************
    ' Need to create a local context, if files are not on the server
    '***************************************************************
    url.bIsLevel0 = 0
    bMerge = ESB_TRUE

    cppDrillRegions(0) = "qtr1"
    url.cpURLXML = "Testing"
    url.cpURLName = "VB URL7"
    url.iURLXMLSize = 8

    sts = EsbUpdateDrillThruURL(hCtx, cppDrillRegions, url, bMerge)

    Debug.Print "EsbUpdateDrillThruURL sts: " & sts
End Sub
```

See also an extended example in “Drill-through Visual Basic API Example” on page 1132.

**EsbUpdateFile**

Sends an update specification to the active database from a file. The update data can either be stored in the database, or just verified and any errors returned. Also, any data blocks locked for update can be unlocked by this call.

Syntax

```
EsbUpdateFile (hDestCtx, hSrcCtx, AppName, DbName, FileName, isStore, isUnlock)
ByVal hDestCtx As Long
ByVal hSrcCtx As Long
ByVal AppName As String
ByVal DbName As String
ByVal FileName As String
ByVal isStore As Integer
ByVal isUnlock As Integer
```

Parameter Description

- **hDestCtx**  VB API context handle of a target database on the server.
- **hSrcCtx**  VB API context handle for the update file location. The report file can reside on the client or on the same server as the target database.
### Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AppName</td>
<td>Application name for the update file location.</td>
</tr>
<tr>
<td>DbName</td>
<td>Database name for the update file location.</td>
</tr>
<tr>
<td>FileName</td>
<td>Name of the update specification file.</td>
</tr>
<tr>
<td>isStore</td>
<td>Controls storage of data. If TRUE, data is stored in the server; if FALSE, no data is stored.</td>
</tr>
<tr>
<td>isUnlock</td>
<td>Controls unlocking of data blocks. If TRUE, all relevant blocks which are locked will be unlocked (after data is stored, if necessary). If FALSE, no blocks are unlocked.</td>
</tr>
</tbody>
</table>

### Notes

- If this function causes data to be stored (Store flag is TRUE), the relevant data blocks must previously have been locked for update (e.g. by calling `EsbReport()` with the Lock flag set to TRUE).
- If both the Store and Unlock flags are set to FALSE, the database merely performs a syntax check of the update specification.

### Return Value

None.

### Access

This function requires the caller to have write privilege (ESB_PRIV_WRITE) to the active database.

### Example

```visualbasic
Declare Function EsbUpdateFile Lib "ESBAPIN" (ByVal hDestCtx As Long, ByVal hSrcCtx As Long, ByVal AppName As String, ByVal DbName As String, ByVal FileName As String, ByVal Store As Integer, ByVal Unlock As Integer) As Long

Sub ESB_UpdateFile()
  Dim sts As Long
  Dim AppName As String
  Dim DbName As String
  Dim FileName As String
  Dim Store As Integer
  Dim pUnlock As Integer
  Dim hSrcCtx As Long
  AppName = "Sample"
  DbName = "Basic" '**************************************
  ' Update file is an object at the server *
  '**************************************
  hSrcCtx = hCtx
  FileName = "update"    Store = ESB_YES
  pUnlock = ESB_NO   '************
  ' Update File
  '************
  sts = EsbUpdateFile (hCtx, hSrcCtx, AppName,
                       DbName, FileName, Store, pUnlock)
End Sub
```

1436 Visual Basic Main API Functions
**EsbValidateDB**

Validates the integrity of the database.

**Syntax**

```
EsbValidateDB (hCtx, DbName, FileName)
ByVal hCtx   As Long
ByVal DbName As String
ByVal FileName As String
```

**Parameter Description**

- **hCtx**  API context handle.
- **DbName**  Database name. Required, cannot be NULL.
- **FileName**  Error log file name, to be placed in the app\db directory on the server. Required.

**Notes**

- This function runs validation checks to ensure the integrity of the database.
- This command validates the current database. You must select a database before issuing the `EsbValidateDB()` command.
- `EsbValidateDB()` checks for data integrity in each block. Reading from top to bottom, the validation process goes through the entire database and checks blocks, sections, block type, and block length, and checks for validity in floating point numbers.
- This command writes blocks and information about bad blocks to the log file.
- If this command finds integrity errors, it writes validation process error messages to a text-format log file. The default location for the file is in the application\database directory; for example: ESSBASE\APP\DB\VALIDATE.LST
- Precede this call with a call to `EsbSetActive()`.
- This function is asynchronous, so you must continue to call `EsbGetProcessState()` until the validation process is finished.
- The index contains an index for every data block. For every Read operation, this command automatically compares the index key in the index page with the index key in the corresponding data block and checks other header information in the block. If it encounters a mismatch, `EsbValidateDB()` displays an error message and continues processing until it has checked the entire database.

**Return Value**

None.
**Access**

This function requires the caller to have Database Manager privilege (ESB_PRIV_DBDESIGN) for the specified database.

**Example**

```
Declare Function EsbValidateDB Lib "ESBAPIN" (ByVal hCtx As Long, ByVal DbName As String, ByVal FileName As String) As Long

Sub ESB_ValidateDB()
    Dim sts As Long
    Dim DbName As String
    Dim FileName As String
    Dim ProcState As ESB_PROCSTATE_T   DbName = "Basic"
    FileName = "D:\Essbase\App\Sample\Basic\Validate.lst"   sts = EsbValidateDB(hCtx, DbName, FileName)
    If Not sts Then
        '*************************************
        'Check process state until it is done
        '*************************************
        sts = EsbGetProcessState(hCtx, ProcState)
        Do While sts = 0 And ProcState.State = ESB_STATE_INPROGRESS
            sts = EsbGetProcessState(hCtx, ProcState)
        Loop
    End If
End Sub
```

**See Also**

- EsbSetActive
- EsbGetProcessState

---

**EsbValidateHCtx**

Validates a specific context handle (hCtx).

**Syntax**

EsbValidateHCtx (hCtx)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>ESB_HCTX_T</td>
<td>The API context handle to validate</td>
</tr>
</tbody>
</table>

**Notes**

This function can be used after any extended wait period to insure the program's context handle is still recognized by the server.
**Return Value**

This function returns 0 if the context handle is valid, otherwise it returns an error code to indicate the invalid context handle. Possible reasons for an invalid context handle include the login might have timed out or the user was explicitly logged out by the administrator.

**Access**

This function requires no special access.

**Example**

```vbnet
Dim sts As Long
Dim Count As Integer
Dim pAccess As Integer
sts = EsbLogin(hInst, "localhost", "test", "testing", Count, hCtx)
sts = EsbSetActive(hCtx, "sample", "Basic", pAccess)

' Do something else not related to Essbase Server
sts = EsbValidateHCtx(hCtx)
If (sts <> 0) Then
  'if Context no longer valid, re-login
  sts = EsbLogin(hInst, "localhost", "test", "testing", Count, hCtx)
  sts = EsbSetActive(hCtx, "Sample", "Basic", pAccess)
End If

' Proceed
```

**See Also**

- [EsbLogin](#)
- [EsbAutoLogin](#)
- [EsbTerm](#)

**EsbVerifyFilter**

Verifies the syntax of a series of filter row strings against a specified database.

**Syntax**

```vbnet
EsbVerifyFilter (hCtx, AppName, DbName)
ByVal hCtx   As Long
ByVal AppName As String
ByVal DbName   As String
```

**Parameter**

- **hCtx**  
  VB API context handle.

- **AppName**  
  Application name.

- **DbName**  
  Database name.
Notes

This call must be followed by successive calls to EsbVerifyFilterRow() to verify all the rows for the filter.

Return Value

None.

Access

This function requires the caller to have Database Design privilege (ESB_PRIV_DBDESIGN) for the specified database.

Example

Declare Function EsbVerifyFilter Lib "ESBAPIN" (ByVal hCtx As Long, ByVal AppName As String, ByVal DbName As String) As Long
Sub ESB_VerifyFilter()
  Dim sts As Long
  Dim AppName As String
  Dim DbName As String
  Dim Row As String
  AppName = "Sample"
  DbName = "Basic"
  '***************
  ' Verify Filter
  '***************
  sts = EsbVerifyFilter(hCtx, AppName, DbName) ' Initialize Filter Row
  Row = '@IDESCENDANTS(Scenario)'
  sts = EsbVerifyFilterRow(hCtx, Row) ' Initialize Filter Row
  Row = '@IDESCENDANTS(Product)'
  sts = EsbVerifyFilterRow(hCtx, Row) sts = EsbVerifyFilterRow(hCtx, ByVal 0&)
End Sub

See Also

- EsbGetFilter
- EsbVerifyFilterRow

EsbVerifyFilterRow

Verifies the syntax of a single filter row strings against a specified Database.

Syntax

EsbVerifyFilterRow (hCtx, FltRow)
ByVal hCtx As Long
ByVal FltRow As Any

Parameter Description

hCtx VB API context handle.
Parameter Description

FltRow Filter row string.

Notes
This function should be called repeatedly after calling **EsbVerifyFilter**(), once for each row of the filter, terminating the row list with NULL.

Return Value
None.

Access
This function requires the caller to have Database Manager privilege (ESB_PRIV_DBDESIGN) for the specified database.

Example

Declare Function EsbVerifyFilterRow Lib "ESBAPIN" (ByVal hCtx As Long, ByVal FltRow As Any) As Long

See the example for **EsbVerifyFilter**.

See Also
- **EsbGetFilter**
- **EsbVerifyFilter**

**EsbWriteToLogFile**

Writes a message to the Essbase Server log file (**essbase.log**), or to the application log file (**appname.log**).

Syntax

Declare Function EsbWriteToLogFile Lib "esbapin" ( ByVal hCtx As Long, ByVal isAgentLog As Boolean, ByVal Message As String) As Long

Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>API context handle.</td>
</tr>
<tr>
<td>isAgentLog</td>
<td>If TRUE, message is written to the Essbase Server log file, <strong>essbase.log</strong>. If FALSE, message is written to the application log file, <strong>appname.log</strong>.</td>
</tr>
<tr>
<td>Message</td>
<td>Message to be logged to the Essbase Server log file (<strong>essbase.log</strong>), or to the application log file (<strong>appname.log</strong>).</td>
</tr>
</tbody>
</table>

Notes
- Use **EsbGetLogFile()** to view message logs.
- For the locations of `essbase.log` and `appname.log`, see the *Oracle Essbase Database Administrator's Guide*.

**Return Value**

Returns a zero if successful.

**Access**

The caller must have administrator privilege (ESS_ACCESS_SUPER) for the specified application.
Part VI

Visual Basic Outline API

In Visual Basic Outline API:

- Using the Visual Basic Outline API
- Visual Basic Outline API Declarations
- Visual Basic Outline API Functions
- Example of Traversing an Outline (VB)
About the Visual Basic Outline API

The Outline API is a set of functions for creating, maintaining, and manipulating Essbase outlines from within a custom application. With the Outline API, you have the same ability to manipulate database outlines from within code as you have using the Outline Editor in Administration Services.

The Outline API is an important part of the Essbase API, with interfaces for C and Visual Basic. The Outline API is used in conjunction with the Essbase API and requires a server connection.

Visual Basic Outline API Error Handling

Outline API functions return 0 when they succeed; if they fail they return an error status value as defined in esserror.h for C and esberror.bas for Visual Basic. Functions of the main API use the error message callback routine and pass an error number to the message handler. The handler uses the essbase.mdb message database to determine the error message and display an error message to the user.

Outline API functions do not ordinarily use the error message callback routine when returning an error status. The error callback routine is called in the following situations:

- If you call functions that use the network (EsxOtlOpenOutline(), EsxOtlWriteOutline(), and EsxOtlRestructure()), and they incur errors on non-outline related actions.
- If a NULL is found during routine checking when passed into the Outline API, and API_NULL_ARG is returned.
If a bad outline handle (HOUTLINE) is passed into any call requiring an outline handle, and OTLAPI_BAD_HOUTLINE is returned.

**Visual Basic Server Outline Queries**

Several functions support a query interface to the outline API such that the outline does not need to be downloaded from the server and completely read into memory. These Outline API functions support only server outlines. Prior to opening the outline, the user must log in to a server, setting up a valid Essbase login context.

Error handling for these functions is done via the standard API error handling mechanism. Therefore, any message callback that the caller has specified from EsxInit() is called on errors. Here’s the way it works:

1. The programmer initializes the API as always by calling EsxInit() and EsxLogin().
2. The programmer calls EsxOtlOpenOutlineQuery() to "open" the outline from the server and bring back some initial information. The information brought back from the server is all information in the ESX_OUTLINEINFO_T structure and for each dimension, all relevant information in the ESX_OTLMBR_T internal structure, which includes the ESX_MBRINFO_T structure.
3. The caller needs to get information about members, so he calls EsxOtlQueryMembers() with the appropriate flags to get an array of member handles back. The EsxOtlQueryMembers() call returns all relevant information in the ESX_OTLMBR_T internal structure. The user can then call any of the EsxOtlGetXxxx() calls that relate to a specific member by passing in one of the returned member handles. See the comments section in the EsxOtlQueryMembers() call for more information about which calls are supported when the outline is opened in "query" mode.
4. When the caller is done with the data returned from an EsxOtlQueryMembers() call, he should call EsxOtlFreeMembers() or EsbOtlFreeMember() to free the array of members.
5. The caller should call EsxOtlCloseOutline() when complete to clean up internal data structures.
6. The caller terminates the API as always by calling EsxLogout() and EsxTerm().

**Visual Basic Outline API Outline Verification**

The Outline API is designed to prevent the caller from creating an illegal outline. To check the outline, use the EsxOtlVerifyOutline() function to verify it before saving it to the server. The Outline API calls EsxOtlVerifyOutline() automatically when an outline is written to the server, if it was not called previously.

Each function call in the Outline API verifies that processing by the caller does not result in an illegal outline. For example, EsxOtlRenameMember() checks a new member name to make sure that it is valid and does not already exist in the outline. Here are a few exceptions to this automatic validation:
- `EsxOtlOpenOutline()` allows the caller to read in a previously created outline that is illegal. This outline could be illegal because the Outline Editor in the Application Manager allows you to save an invalid outline to a local file. Any existing errors are detected when `EsxOtlVerifyOutline()` is called. Also, some individual operations are illegal during processing if the outline starts out as illegal.

- `EsxOtlDeleteMember()` and `EsxOtlDeleteDimension()` do not check for any alias combinations that contain a deleted member. `EsxOtlVerifyOutline()` detects this condition.

- `EsxOtlSetMemberFormula()` allows you to enter an illegal formula, and `EsxOtlVerifyOutline()` does not check member formulas. An illegal member formula causes failure during restructure. `EsxGetProcessState()` displays the error message returned from the server.

### Visual Basic Outline API Memory Allocation

The Essbase API provides a set of memory management functions, `EsxAlloc()`, `EsxRealloc()`, and `EsxFree()`. These functions, plus all internal API memory allocations, call memory allocation routines pointed to by the `AllocFunc`, `ReallocFunc`, and `FreeFunc` fields of the `ESX_INIT_T` initialization structure.

If you are using your own custom memory allocation functions, make sure your memory allocation scheme can handle allocating many small memory buffers.

### Visual Basic Outline API Security Requirements

Because you can use the Outline API to create, edit, and delete outlines, you must be aware of some security issues when creating an application that uses the Outline API. These issues impact only programs that create, edit, or save outlines during a session.

To manipulate outlines through the Outline Editor in the Application Manager, you must have Application Designer or higher privileges. You also need these privileges to use a program that uses the Outline API during execution. If you do not have these privileges, Outline API calls that read or write outlines from the server do not work. See the Oracle Essbase Database Administrator’s Guide for more detailed information on security and privilege levels.

For example, you are writing a new EIS end-user application that allows your users to explore a number of "what-if" situations during a session. To do this, the program dynamically creates a number of Essbase databases during a session. These databases (and their outlines) are temporary and are not saved after the session terminates. You can approach this situation in several ways:

- If you want the user to be able to create an application and multiple databases during a session, give the user the Create/Delete Application privilege. This privilege must be assigned by an Essbase Supervisor prior to running the program. This is a relatively high privilege level in Essbase, but if the user does not have access to other programs, such as the Application Manager, there is little impact on the overall system security.
If you do not need multiple databases available at the same time, you can have the Essbase Supervisor create a temporary application and database during the installation of your program. The program itself manipulates the temporary database without having to create a new database for each "what-if" situation.

With the second approach, a user requires only the lower and more restricted Database Designer privilege. You could have the Essbase Supervisor set up a special group with Database Designer privilege only for your temporary application and database. Users can be assigned to that group. The users would revert to ordinary user privilege for any other access to the system. This approach offers less security exposure, but does require more set up prior to running your program.

**Visual Basic Outline API Function Call Sequence**

When you use the Outline API, your program must call some API functions before others. Follow this basic call sequence:

1. Call `EsxInit()` before any other API function.
   The API returns an instance handle.
2. Call `EsxLogin()` or `EsxAutoLogin()` to log on to the server.
   The API returns a context handle.
3. Call `EsxOtlOpenOutline()` or `EsxOtlNewOutline()` to open or create an outline.
   The API returns an outline handle.
4. Call `EsxOtlWriteOutline()` to write the current outline to the server.
   `EsxOtlVerifyOutline()` is called automatically by the API before the outline is saved, unless you call it before this.
5. Call `EsxOtlRestructure()` to restructure the database based on the changes made to the outline.
6. Call `EsxUnlockObject()` to unlock the outline object if it is locked when the outline is opened.
7. Call `EsxOtlCloseOutline()` to free resources associated with the outline.
8. Call `EsxLogout()` to log off the server.
   This invalidates the context handle.
9. Call `EsxTerm()` to end the session.
   This invalidates the instance handle.

**Typical Visual Basic Outline API Task Sequence**

This is a typical order of operations for a simple Outline API application.

1. Create and initialize an `ESX_INIT_T` structure.
2. Initialize the Outline API by calling `EsxInit()`.
3. Allocate any local static or global structures.

4. Log on to the required server by calling EsxLogin() or EsxAutoLogin().

5. Create and initialize an ESX_OUTLINEINFO_T structure (only for a new outline).

6. Open an existing outline or create a new outline by calling EsxOtlOpenOutline() or EsxOtlNewOutline().

7. Work on the outline.

8. Verify the outline by calling EsxOtlVerifyOutline().

9. Write the verified outline to the server by calling EsxOtlWriteOutline().

   The outline is saved with an .OTN extension.

10. Restructure the database by calling EsxOtlRestructure().

   The .OTN file is changed to an .OTL file. This is an asynchronous function call; therefore, you should call EsxGetProcessState() until the process is complete.

11. Unlock the outline (if it was locked on opening) by calling EsxUnlockObject().

12. Free all information associated with the outline by calling EsxOtlCloseOutline().

13. Log off the server by calling EsxLogout().

14. Free any local static or global structures.

15. Terminate the API by calling EsxTerm().
Using the Visual Basic Outline API
In This Chapter

- VB Outline Error Return Values .................................................. 1451
- VB Outline Symbolic Constant Definitions ........................................ 1454
  - ESB_ATTRIBUTEQUERY_T ................................................. 1458
  - ESB_GENLEVELNAME_T .................................................. 1459
  - ESB_MBCOUNTS_T .......................................................... 1459
  - ESB_MBRINFO_T ............................................................. 1460
  - ESB_OUTERROR_T ........................................................... 1462
  - ESB_OUTLINEINFO_T ....................................................... 1463
  - ESB_PREDICATE_T .......................................................... 1464

**VB Outline Error Return Values**

The following table describes the error status constants returned when an Outline API call fails. These values are defined in the Essbase Outline API Visual Basic global text file esberror.bas.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OTLAPI_BAD_ALIASTABLE</td>
<td>Illegal alias table</td>
</tr>
<tr>
<td>OTLAPI_BAD_CONSOL</td>
<td>Invalid consolidation type (+,-,etc)</td>
</tr>
<tr>
<td>OTLAPI_BAD_GENLEVELNAME</td>
<td>Invalid generation or level name</td>
</tr>
<tr>
<td>OTLAPI_BAD_HOUTLINE</td>
<td>Invalid outline handle passed to EsbOtl... function</td>
</tr>
<tr>
<td>OTLAPI_BAD_MBRNAME</td>
<td>Invalid member name</td>
</tr>
<tr>
<td>OTLAPI_BAD_MBRNAME</td>
<td>Invalid member handle</td>
</tr>
<tr>
<td>OTLAPI_BAD_MOVE</td>
<td>Illegal move of member; for example, can't move a member to its descendant</td>
</tr>
<tr>
<td>OTLAPI_BAD_OBJTYPE</td>
<td>Illegal object type</td>
</tr>
<tr>
<td>OTLAPI_BAD_OUTLINETYPE</td>
<td>Invalid outline type</td>
</tr>
<tr>
<td>OTLAPI_BAD_RENAMESHARE</td>
<td>A shared member cannot be renamed</td>
</tr>
<tr>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>OTLAPI_BAD_RESTRUCTTYPE</td>
<td>Invalid restructure type</td>
</tr>
<tr>
<td>OTLAPI_BAD_SORTCOMPAREFUNC</td>
<td>Invalid sorting compare function</td>
</tr>
<tr>
<td>OTLAPI_BAD_SORTTYPE</td>
<td>Invalid sort type</td>
</tr>
<tr>
<td>OTLAPI_BAD_TRANSTYPE</td>
<td>Unknown transaction type when creating a transaction (internal error)</td>
</tr>
<tr>
<td>OTLAPI_BAD_USERATTR</td>
<td>Invalid user attribute</td>
</tr>
<tr>
<td>OTLAPI_CUR_NOACCOUNTS</td>
<td>There is no Accounts dimension. You need an Accounts dimension to create a currency database</td>
</tr>
<tr>
<td>OTLAPI_CUR_NOCOUNTRY</td>
<td>There is no Country dimension. You need a Country dimension to create a currency database</td>
</tr>
<tr>
<td>OTLAPI_CUR_NOTIME</td>
<td>There is no Time dimension. You need a Time dimension to create a currency database</td>
</tr>
<tr>
<td>OTLAPI_ERR_ADDNAMEUSED</td>
<td>Member name already used (add operation)</td>
</tr>
<tr>
<td>OTLAPI_ERR_ALIASSHARED</td>
<td>A shared member cannot have an alias</td>
</tr>
<tr>
<td>OTLAPI_ERR_ALIASSTABLEEXISTS</td>
<td>Alias table already exists</td>
</tr>
<tr>
<td>OTLAPI_ERR_ALIASSTABLENAME</td>
<td>Illegal alias table name</td>
</tr>
<tr>
<td>OTLAPI_ERR_ALREADYCURRENCY</td>
<td>The outline is a currency outline. You are trying to create a currency outline, and the initial outline is already a currency outline.</td>
</tr>
<tr>
<td>OTLAPI_ERR_BADSHARE</td>
<td>Illegal share value</td>
</tr>
<tr>
<td>OTLAPI_ERR_BAD_SKIP</td>
<td>Illegal time balance skip value</td>
</tr>
<tr>
<td>OTLAPI_ERR_BAD_STORAGE</td>
<td>Illegal dimension storage value</td>
</tr>
<tr>
<td>OTLAPI_ERR_BAD_STORAGECATEGORY</td>
<td>Illegal storage category</td>
</tr>
<tr>
<td>OTLAPI_ERR_BAD_TIME_BAL</td>
<td>Illegal time balance value</td>
</tr>
<tr>
<td>OTLAPI_ERR_CONFIGTOOMANYDIMS</td>
<td>Too many dimensions to configure automatically</td>
</tr>
<tr>
<td>OTLAPI_ERR_COPY_ALIASSTABLE</td>
<td>Source and destination tables are the same</td>
</tr>
<tr>
<td>OTLAPI_ERR_CREATE_TEMP</td>
<td>Cannot create temporary file name. You are probably trying to create it on a read-only drive. We create a temporary file on the client every time you open or write an outline from/to the server.</td>
</tr>
<tr>
<td>OTLAPI_ERR_CURTOOMANYDIMS</td>
<td>Too many dimensions in a currency outline. A currency outline is limited to four dimensions.</td>
</tr>
<tr>
<td>OTLAPI_ERR_DELETE_DEF_ALIAS</td>
<td>Cannot delete the default alias table</td>
</tr>
<tr>
<td>OTLAPI_ERR_DUP_LANGCODE</td>
<td>The language code is assigned to another alias table within the same database</td>
</tr>
<tr>
<td>OTLAPI_ERR_DUP_GENERATIONLEVNAME</td>
<td>Cannot add, rename, or set a member name or alias that duplicates a generation or level name</td>
</tr>
<tr>
<td>OTLAPI_ERR_DUPLICATE_ALIAS</td>
<td>Duplicate alias</td>
</tr>
<tr>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>OTLAPI_ERR_DUPLICATENAME</td>
<td>Duplicate member name</td>
</tr>
<tr>
<td>OTLAPI_ERR_FILEIO</td>
<td>Could not read from or write to file</td>
</tr>
<tr>
<td>OTLAPI_ERR_FILEOPEN</td>
<td>Could not open file</td>
</tr>
<tr>
<td>OTLAPI_ERR_GENLEVELEXISTS</td>
<td>Generation or level already has a name</td>
</tr>
<tr>
<td>OTLAPI_ERR_GENLEVELNAMEEXISTS</td>
<td>Generation or level name already exists</td>
</tr>
<tr>
<td>OTLAPI_ERR_GENLEVELVALUE</td>
<td>Illegal generation or level value</td>
</tr>
<tr>
<td>OTLAPI_ERR_ILLEGALALIASSTRING</td>
<td>Illegal member combinational for alias</td>
</tr>
<tr>
<td>OTLAPI_ERR_ILLEGALCOMBOALIAS</td>
<td>Illegal combinational alias name</td>
</tr>
<tr>
<td>OTLAPI_ERR_ILLEGALCURRENCY</td>
<td>Illegal currency member</td>
</tr>
<tr>
<td>OTLAPI_ERR_ILLEGALDEFAILIAS</td>
<td>Illegal default alias name</td>
</tr>
<tr>
<td>OTLAPI_ERR_ILLEGALNAME</td>
<td>Illegal member name</td>
</tr>
<tr>
<td>OTLAPI_ERR_ILLEGALTAG</td>
<td>Illegal dimension tag (category)</td>
</tr>
<tr>
<td>OTLAPI_ERR_INVALIDOPTION</td>
<td>Occurs when the user passes in an invalid option to EssOtlGetGenNames() or EssOtlGetLevelNames()</td>
</tr>
<tr>
<td>OTLAPI_ERR_LEAFLABEL</td>
<td>Leaf member defined as a label member</td>
</tr>
<tr>
<td>OTLAPI_ERR_MAXALIASTABLES</td>
<td>Maximum number of alias tables has been reached</td>
</tr>
<tr>
<td>OTLAPI_ERR_MEMBERCALC</td>
<td>Illegal member formula</td>
</tr>
<tr>
<td>OTLAPI_ERR_NOALIAS</td>
<td>No alias for this member</td>
</tr>
<tr>
<td>OTLAPI_ERR_NOALIASCOMBO</td>
<td>No alias combination</td>
</tr>
<tr>
<td>OTLAPI_ERR_NODTSMBRANDGENMATCH</td>
<td>DTS member is not enabled for this generation</td>
</tr>
<tr>
<td>OTLAPI_ERR_NOFORMULA</td>
<td>No formula for this member</td>
</tr>
<tr>
<td>OTLAPI_ERR_NOSHAREPROTO</td>
<td>Shared member with no actual member</td>
</tr>
<tr>
<td>OTLAPI_ERR_NOTADIM</td>
<td>Dimension name expected</td>
</tr>
<tr>
<td>OTLAPI_ERR_NOTIMEDIM</td>
<td>No time dimension defined (can't do time balance operations without a time dimension)</td>
</tr>
<tr>
<td>OTLAPI_ERR_NOTVERIFIED</td>
<td>Outline has errors (when saving to the server)</td>
</tr>
<tr>
<td>OTLAPI_ERR_OPENMODE</td>
<td>File was opened in the wrong mode to make this call. If you call EssOtlOpenOutlineQuery() to open the outline, not all of the calls will work.</td>
</tr>
<tr>
<td>OTLAPI_ERR_RENAMEDEFAILIAS</td>
<td>Cannot rename the default alias table</td>
</tr>
<tr>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------------</td>
<td>------------------------------------------------------------------</td>
</tr>
<tr>
<td>OTLAPI_ERR_RENAMENAMEUSED</td>
<td>Member name already used (rename operation)</td>
</tr>
<tr>
<td>OTLAPI_ERR_SHAREDMEMBERFORMULA</td>
<td>Shared member cannot have a formula</td>
</tr>
<tr>
<td>OTLAPI_ERR_SHARENOTLEVEL0</td>
<td>Shared member not at level 0 (a shared member cannot be a parent of another member)</td>
</tr>
<tr>
<td>OTLAPI_ERR_SHAREUDA</td>
<td>Cannot set a user attribute for a shared member</td>
</tr>
<tr>
<td>OTLAPI_ERR_TIMESPARSE</td>
<td>Accounts dimension is dense and time dimension sparse-is not used</td>
</tr>
<tr>
<td>OTLAPI_NULL_ARG</td>
<td>NULL argument passed to EsbOtl... function</td>
</tr>
<tr>
<td>OTLAPI_NO_GENLEVELNAME</td>
<td>Cannot find generation or level name</td>
</tr>
<tr>
<td>OTLAPI_NO_USERATTR</td>
<td>Cannot find user attribute</td>
</tr>
<tr>
<td>OTLAPI_SORT_TOOMANY</td>
<td>Too many members to sort (64K / 4 members is the maximum sorting capacity)</td>
</tr>
</tbody>
</table>

**VB Outline Symbolic Constant Definitions**

The following tables describe the symbolic constants used specifically by the Outline API. These constants are defined in the Essbase Visual Basic global text file `esb32.bas`.

- Table 22, “Restructure Values,” on page 1454
- Table 23, “Account Member Currency Conversion Category Values,” on page 1455
- Table 24, “Account Member Time Balance Values,” on page 1455
- Table 25, “Account Member Time Balance Skip Values,” on page 1455
- Table 26, “Share Constants,” on page 1455
- Table 27, “Dimension Categories (Tags),” on page 1456
- Table 28, “Dimension Categories Used For Optimizing Auto-Configure Storage,” on page 1456
- Table 29, “Sorting Options,” on page 1456
- the section called “Query Types”
- Table 30, “Query Options,” on page 1457
- Table 31, “Generation and Level Options,” on page 1457

**Table 22  Restructure Values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESB_DOR_ALLDATA</td>
<td>Keep all data</td>
</tr>
<tr>
<td>ESB_DOR_NODATA</td>
<td>Discard all data</td>
</tr>
<tr>
<td>ESB_DOR_LOWDATA</td>
<td>Keep only level 0 data</td>
</tr>
<tr>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>ESB_DOR_INDATA</td>
<td>Keep only input data</td>
</tr>
</tbody>
</table>

**Table 23**  Account Member Currency Conversion Category Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESB_CONV_NONE</td>
<td>Default conversion category. Member inherits category from parent.</td>
</tr>
<tr>
<td>ESB_CONV_CATEGORY</td>
<td>Define a currency conversion category for this member</td>
</tr>
<tr>
<td>ESB_CONV_NOCONV</td>
<td>No conversion for this member</td>
</tr>
</tbody>
</table>

**Table 24**  Account Member Time Balance Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESB_TIMEBAL_NONE</td>
<td>No time balance</td>
</tr>
<tr>
<td>ESB_TIMEBAL_FIRST</td>
<td>First time balance member</td>
</tr>
<tr>
<td>ESB_TIMEBAL_LAST</td>
<td>Last time balance member</td>
</tr>
<tr>
<td>ESB_TIMEBAL_AVG</td>
<td>Average time balance member</td>
</tr>
</tbody>
</table>

**Table 25**  Account Member Time Balance Skip Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESB_SKIP_NONE</td>
<td>Don't skip anything</td>
</tr>
<tr>
<td>ESB_SKIP_MISSING</td>
<td>Skip the value if the data is #missing</td>
</tr>
<tr>
<td>ESB_SKIP_ZEROS</td>
<td>Skip the value if the data is 0</td>
</tr>
<tr>
<td>ESB_SKIP_BOTH</td>
<td>Skip the value if the data is #missing or 0</td>
</tr>
</tbody>
</table>

**Note:** Account Member Time Balance Skip values are only valid if time balance does not equal ESB_TIMEBAL_NONE.

**Table 26**  Share Constants

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESB_SHARE_DATA</td>
<td>Normal member (default value)</td>
</tr>
<tr>
<td>ESB_SHARE_NEVER</td>
<td>Never share this member, even if it would normally be an implicit share.</td>
</tr>
<tr>
<td>ESB_SHARE_LABEL</td>
<td>Label member. Do not store data for this member.</td>
</tr>
<tr>
<td>ESB_SHARE_SHARE</td>
<td>Shared member. This member cannot have children and must have the actual member with the same name in the same dimension.</td>
</tr>
<tr>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ESB_CAT_NONE</td>
<td>No category</td>
</tr>
<tr>
<td>ESB_CAT_ACCOUNTS</td>
<td>Accounts dimension</td>
</tr>
<tr>
<td>ESB_CAT_TIME</td>
<td>Time dimension</td>
</tr>
<tr>
<td>ESB_CAT_COUNTRY</td>
<td>Country dimension</td>
</tr>
<tr>
<td>ESB_CAT_TYPE</td>
<td>Type dimension. This dimension is valid only in currency databases.</td>
</tr>
<tr>
<td>ESB_CAT_CURPARTITION</td>
<td>Currency partition dimension. Valid only in non-currency databases.</td>
</tr>
</tbody>
</table>

Table 28  Dimension Categories Used For Optimizing Auto-Configure Storage

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESB_STORECAT_OTHER</td>
<td>None or don't know storage category</td>
</tr>
<tr>
<td>ESB_STORECAT_TIME</td>
<td>Time storage category</td>
</tr>
<tr>
<td>ESB_STORECAT_UNITS</td>
<td>Units storage category</td>
</tr>
<tr>
<td>ESB_STORECAT_SCENARIO</td>
<td>Scenario storage category</td>
</tr>
<tr>
<td>ESB_STORECAT_ACCOUNTS</td>
<td>Accounts storage category</td>
</tr>
<tr>
<td>ESB_STORECAT_PRODUCT</td>
<td>Product storage category</td>
</tr>
<tr>
<td>ESB_STORECAT_ORGAN</td>
<td>Organization storage category</td>
</tr>
<tr>
<td>ESB_STORECAT_MARKET</td>
<td>Market storage category</td>
</tr>
<tr>
<td>ESB_STORECAT_CUSTOMER</td>
<td>Customer storage category</td>
</tr>
<tr>
<td>ESB_STORECAT_DIST</td>
<td>Distribution Channel storage category</td>
</tr>
<tr>
<td>ESB_STORECAT_BUSUNIT</td>
<td>Business Unit storage category</td>
</tr>
<tr>
<td>ESB_STORECAT_GEOG</td>
<td>Geographical Location storage category</td>
</tr>
</tbody>
</table>

Note: Used for optimizing storage when using storage auto-configure

Table 29  Sorting Options

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESB_SORT_ASCENDING</td>
<td>Sort in ascending order</td>
</tr>
<tr>
<td>ESB_SORT_DESCENDING</td>
<td>Sort in descending order</td>
</tr>
</tbody>
</table>
Query Types
Used for defining the operation to perform in ESB_PREDICATE_T.

- ESB_CHILDREN
- ESB_DESCENDANTS
- ESB_BOTTOMLEVEL
- ESB_SIBLINGS
- ESBSAMELEVEL
- ESBSAMEGENERATION
- ESB_PARENT
- ESB_DIMENSION
- ESB_NAMEDGENERATION
- ESB_NAMEDLEVEL
- ESB_SEARCH
- ESB_WILDSEARCH
- ESB_USERATTRIBUTE
- ESB_ANCESTORS

Table 30  Query Options

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESB_MEMBERSONLY</td>
<td>Valid for ESB_SEARCH, ESB_WILDSEARCH</td>
</tr>
<tr>
<td>ESB_ALIASONLY</td>
<td>Valid for ESB_SEARCH, ESB_WILDSEARCH</td>
</tr>
<tr>
<td>ESB_MEMBERSANDALIASES</td>
<td>Valid for ESB_SEARCH, ESB_WILDSEARCH</td>
</tr>
<tr>
<td>ESB_COUNTONLY</td>
<td>Valid for any query type. Queries the outline without returning any data. Returns a count of how many members meet the query type by filling in the ulTotalCount field in ESB_PMBRCOUNTS_T.</td>
</tr>
</tbody>
</table>

Note: You can specify for certain query types in ESB_PREDICATE_T.

Table 31  Generation and Level Options

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESB_GENLEV_ALL</td>
<td>Returns default and user-defined names</td>
</tr>
<tr>
<td>ESB_GENLEV_ACTUAL</td>
<td>Returns only names that are user-defined</td>
</tr>
<tr>
<td>ESB_GENLEV_DEFAULT</td>
<td>Returns all default names, including the default names for generations and levels that also have user-defined names</td>
</tr>
<tr>
<td>ESB_GENLEV_NOACTUAL</td>
<td>Returns all default names, excluding the default names for generations and levels that also have user-defined names</td>
</tr>
</tbody>
</table>
Note: You can use with EsbOtlGetGenNames and EsbOtlGetLevelNames.

**ESB_ATTRIBUTEQUERY_T**

Contains attribute information for a given attribute member or dimension. It is used by EsbGetAttributeInfo. The fields are:

```vbnet
type ESB_ATTRIBUTEQUERY_T
    InputMember As Variant
    InputMemberType As Integer
    OutputMemberType As Integer
    Operation As Integer
    Attribute As Variant
end type
```

<table>
<thead>
<tr>
<th>VB Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>As Variant</td>
<td>InputMember</td>
<td>Attribute member or dimension</td>
</tr>
<tr>
<td>As Integer</td>
<td>InputMemberType</td>
<td>One of the following member types:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESB_ATTRIBUTE_DIMENSION</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESB_ATTRIBUTE_MEMBER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESB_STANDARD_DIMENSION</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESB_STANDARD_MEMBER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESB_BASE_DIMENSION</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESB_BASE_MEMBER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESB_ATTRIBUTED_MEMBER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESB_INVALID_MEMBER</td>
</tr>
<tr>
<td></td>
<td>OutputMemberType</td>
<td>One of the following member types:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESB_ATTRIBUTE_DIMENSION</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESB_ATTRIBUTE_MEMBER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESB_STANDARD_DIMENSION</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESB_STANDARD_MEMBER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESB_BASE_DIMENSION</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESB_BASE_MEMBER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESB_ATTRIBUTED_MEMBER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESB_INVALID_MEMBER</td>
</tr>
</tbody>
</table>

See Table 20, "VB API Attributes Terminology," on page 1164.
<table>
<thead>
<tr>
<th>VB Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
</table>
| As Integer   | Operation | One of the following operations:  
|              |         | - ESB_EQ: equal to  
|              |         | - ESB_NEQ: not equal to  
|              |         | - ESB_GT: greater than  
|              |         | - ESB_LT: less than  
|              |         | - ESB_GTE: greater than or equal to  
|              |         | - ESB_LTE: lesser than or equal to  
|              |         | - ESB_TYPEOF  
|              |         | - ESB_ALL |
| As Variant   | Attribute | Attribute value |

**ESB_GENLEVELNAME_T**
Contains information about generation and level names.

Type ESB_GENLEVELNAME_T

```
usNumber As Integer
szName    As String * ESB_MBRNAMELEN
```

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td>usNumber</td>
<td>Generation or level number.</td>
</tr>
<tr>
<td>String * ESB_MBRNAMELEN</td>
<td>szName</td>
<td>Generation or level name.</td>
</tr>
</tbody>
</table>

**ESB_MBRCOUNTS_T**

Type ESB_MBRCOUNTS_T

```
ulStart        As Long
ulMaxCount     As Long
ulTotalCount   As Long
ulReturnCount  As Long
```

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Long</td>
<td>ulStart</td>
<td>Starting member for retrieval of information.</td>
</tr>
<tr>
<td>Long</td>
<td>ulMaxCount</td>
<td>Maximum number of members to retrieve.</td>
</tr>
<tr>
<td>Long</td>
<td>ulTotalCount</td>
<td>Return of the total count of members that exist in the results of the query.</td>
</tr>
<tr>
<td>Long</td>
<td>ulReturnCount</td>
<td>Return count of returned member handles.</td>
</tr>
</tbody>
</table>
**ESB_MBRINFO_T**

Contains information about an outline member.

**Type ESB_MBRINFO_T**

```vbnet
szMember          As String * ESB_MBRNAMELEN
usLevel           As Integer
usGen             As Integer
usConsolidation   As Integer
fTwoPass          As Integer
fExpense          As Integer
usConversion      As Integer
szCurMember       As String * ESB_MBRNAMELEN
usTimeBalance     As Integer
usSkip            As Integer
usShare           As Integer
usStorage         As Integer
usCategory        As Integer
usStorageCategory As Integer
ulChildCount      As Long
szComment         As String * ESB_MBRCOMMENTLEN
szDimName         As String * ESB_MBRNAMELEN
Attribute         As Variant
IsAttributed      As Integer
End Type
```

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String * ESB_MBRNAMELEN</td>
<td>szMember</td>
<td>Member name. This field can be set only by the caller when creating the member.</td>
</tr>
<tr>
<td>Integer</td>
<td>usLevel</td>
<td>Level of the member in the outline. This field cannot be modified.</td>
</tr>
<tr>
<td>Integer</td>
<td>usGen</td>
<td>Generation of the member in the outline. This field cannot be modified.</td>
</tr>
<tr>
<td>Integer</td>
<td>usConsolidation</td>
<td>Unary consolidation type. It can be one of the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESB_UCALC_ADD</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESB_UCALC_SUB</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESB_UCALC_MULT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESB_UCALC_DIV</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESB_UCALC_PERCENT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESB_UCALC_NOOP</td>
</tr>
<tr>
<td>Integer</td>
<td>fTwoPass</td>
<td>ESB_TRUE if two-pass calculation member</td>
</tr>
<tr>
<td>Integer</td>
<td>fExpense</td>
<td>ESB_TRUE if expense member</td>
</tr>
<tr>
<td>Integer</td>
<td>usConversion</td>
<td>Currency Conversion type. This is valid only for members of the Accounts dimension. It can be one of the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESB_CONV_NONE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESB_CONV_CATEGORY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESB_CONV_NOCONV</td>
</tr>
<tr>
<td>Data Type</td>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>---------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>String * ESB_MBRNAMELEN</td>
<td>szCurMember</td>
<td>If the member is of the Accounts dimension and usConversion is ESB_CONV_CATEGORY, this field defines the currency category. If the member is of the Country Dimension, this field defines the currency name. This field is undefined in all other situations.</td>
</tr>
<tr>
<td>Integer</td>
<td>usTimeBalance</td>
<td>Time balance option. This is valid only for members of the Accounts dimension. It can be one of the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESB_TIMEBAL_NONE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESB_TIMEBAL_FIRST</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESB_TIMEBAL_LAST</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESB_TIMEBAL_AVG</td>
</tr>
<tr>
<td>Integer</td>
<td>usSkip</td>
<td>Time balance skip option. This is valid only for members of the Accounts dimension if usTimeBalance is not equal to ESB_TIMEBAL_NONE. It can be one of the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESB_SKIP_NONE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESB_SKIP_MISSING</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESB_SKIP_ZEROS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESB_SKIP_BOTH</td>
</tr>
<tr>
<td>Integer</td>
<td>usShare</td>
<td>Share option. It can be one of the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESB_SHARE_DATA (default value)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESB_SHARE_DYNCALCSTORE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESB_SHARE_DYNCALCNOSTORE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESB_SHARE_NEVER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESB_SHARE_LABEL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESB_SHARE_SHARE (Valid for level 0 members only)</td>
</tr>
<tr>
<td>Integer</td>
<td>usStorage</td>
<td>Dimension storage type. This field is valid only for dimension members and can be one of the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESB_DIMTYPE_DENSE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESB_DIMTYPE_SPARSE</td>
</tr>
<tr>
<td>Integer</td>
<td>usCategory</td>
<td>Dimension category. This field is valid only for dimensions and attribute members. It can be one of the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESB_CAT_ACCOUNTS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESB_CAT_ATTRCALC (for internal use only)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESB_CAT_ATTRIBUTE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESB_CAT_COUNTRY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESB_CAT_CURPARTITION (for non-currency databases only)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESB_CAT_NONE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESB_CAT_TIME (for currency databases only)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESB_CAT_TYPE</td>
</tr>
<tr>
<td>Data Type</td>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| Integer   | usStorageCategory | Dimension storage category. This field is valid only for dimensions and attribute members. It is used to optimize the storage types of dimensions when the outline is configured for automatic optimization. It can be one of the following:  
  - ESB_STORECAT_ACCOUNTS  
  - ESB_STORECAT_ATTRCALC (for internal use only)  
  - ESB_STORECAT_ATTRIBUTE  
  - ESB_STORECAT_BUSUNIT  
  - ESB_STORECAT_CUSTOMER  
  - ESB_STORECAT_DIST  
  - ESB_STORECAT_GEOG  
  - ESB_STORECAT_MARKET  
  - ESB_STORECAT_ORGAN  
  - ESB_STORECAT_OTHER  
  - ESB_STORECAT_PRODUCT  
  - ESB_STORECAT_SCENARIO  
  - ESB_STORECAT_TIME  
  - ESB_STORECAT_UNITS |
| Long      | ulChildCount      | This field contains the total number of children of the member specified in ESB_MBRNAME_T. |
| String * ESB_MBRCOMMENTLEN | szComment | Member comment array. |
| String * ESB_MBRNAMELEN | szDimName | Dimension name. |
| Variant   | Attribute         | Attribute value: For an attribute dimension or zero-level (leaf node) attribute member, one of the following data types:  
  - ESB_ATTRMBRDT_BOOL  
  - ESB_ATTRMBRDT_DATETIME  
  - ESB_ATTRMBRDT_DOUBLE  
  - ESB_ATTRMBRDT_STRING  
  For any attribute member, but not an attribute dimension:  
  - ESB_ATTRMBRDT_NONE  
  - ESB_ATTRMBRDT_AUTO |
| Integer   | IsAttributed      | Indicates whether the member has attributes associated with it. |

**ESB_OUTERROR_T**

Returns the errors for each member when verifying an outline. The errors are bit field values returned in a 32-bit status word. Each error value corresponds to a function call error return value described in Table 21, “Error Return Values,” on page 1451.
Type ESB_OUTERROR_T

hMember As Long
ulErrors As Long
End Type

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Long</td>
<td>hMember</td>
<td>Handle to member with errors.</td>
</tr>
<tr>
<td>Long</td>
<td>ulErrors</td>
<td>Bitmask of errors for the member. It can be any combination of the following values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESB_OUTERROR_ALIASHANCED</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESB_OUTERROR_BADCATEGORY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESB_OUTERROR_BADSHARE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESB_OUTERROR_BADSKIP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESB_OUTERROR_BADTIMEBAL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESB_OUTERROR_BADSTORAGE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESB_OUTERROR_BADSTORAGECATEGORY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESB_OUTERROR_BADTIMEBAL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESB_OUTERROR_CURTOOMANYDIMS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESB_OUTERROR_DUPGENLEVNAME</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESB_OUTERROR_DUPLICATEALIAS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESB_OUTERROR_DUPLICATEALIASNAME</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESB_OUTERROR_ILLEGALALIASSTRING</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESB_OUTERROR_ILLEGALALIASSTRING</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESB_OUTERROR_ILLEGALCURRENCY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESB_OUTERROR_ILLEGALDEFALIAS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESB_OUTERROR_ILLEGALNAME</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESB_OUTERROR_ILLEGALTAG</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESB_OUTERROR_LEAFLABEL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESB_OUTERROR_MEMBERCALC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESB_OUTERROR_NOHAREPROTO</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESB_OUTERROR_NOTIMEDIM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESB_OUTERROR_SHAREDMEMBERFORMULA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESB_OUTERROR_SHARENOTLEVELO</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESB_OUTERROR_SHAREUDA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESB_OUTERROR_TIMESPARSE</td>
</tr>
</tbody>
</table>

Type ESB_OUTLINEINFO_T

fCaseSensitive As String * 1
usOutlineType As Integer
fAutoConfigure As String * 1
End Type
### Data Type

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fCaseSensitive</td>
<td>ESB_TRUE if member names are case-sensitive.</td>
</tr>
<tr>
<td>usOutlineType</td>
<td>Type of the outline. It can be one of the following:</td>
</tr>
<tr>
<td></td>
<td>- ESB_DBTYPE_NORMAL</td>
</tr>
<tr>
<td></td>
<td>- ESB_DBTYPE_CURRENCY</td>
</tr>
<tr>
<td>fAutoConfigure</td>
<td>ESB_TRUE to automatically configure the dimension storage (dense/sparse) when the outline is saved.</td>
</tr>
</tbody>
</table>

### ESB_PREDICATE_T

Contains information about a query description.

Type ESB_PREDICATE_T

```
    ulQuery     As Long
    ulOptions   As Long
    pszDimension As String * ESB_MBRNAMELEN
    pszString1  As String * 256
    pszString2  As String * 256
End Type
```

### Data Type

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ulQuery</td>
<td>Type of query. See EsbOtlQueryMembers for more information.</td>
</tr>
<tr>
<td>ulOptions</td>
<td>Options dependent on the query type. See EsbOtlQueryMembers for more information.</td>
</tr>
<tr>
<td>pszDimension</td>
<td>Dimension name. See EsbOtlQueryMembers for more information.</td>
</tr>
<tr>
<td>pszString1</td>
<td>Input string value. See EsbOtlQueryMembers for more information.</td>
</tr>
<tr>
<td>pszString2</td>
<td>Input string value. See EsbOtlQueryMembers for more information.</td>
</tr>
</tbody>
</table>
## Visual Basic Outline API Function Categories

- “VB Outline API Alias Table Functions” on page 1465
- “VB Outline API Attributes Functions” on page 1466
- “VB Outline API Dynamic Time Series Functions” on page 1466
- “VB Outline API Generation Name Functions” on page 1467
- “VB Outline API Level Name Functions” on page 1467
- “VB Outline API Member Administration Functions” on page 1467
- “VB Outline API Member Alias Functions” on page 1468
- “VB Outline API Member Formula Functions” on page 1468
- “VB Outline API Member Traversal Functions” on page 1468
- “VB Outline API Outline Administration Functions” on page 1469
- “VB Outline API Outline Query Functions” on page 1469
- “VB Outline API Setup and Cleanup Functions” on page 1469
- “VB Outline API User Attribute Functions” on page 1470

### VB Outline API Alias Table Functions

These functions perform operations on alias tables.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EsbOtlCreateAliasTable</td>
<td>Creates an empty alias table in the outline</td>
</tr>
<tr>
<td>EsbOtlCopyAliasTable</td>
<td>Copies an alias table to another alias table</td>
</tr>
<tr>
<td>EsbOtlRenameAliasTable</td>
<td>Renames an existing alias table</td>
</tr>
</tbody>
</table>
### VB Outline API Attributes Functions

These Visual Basic Outline functions are for attributes.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EsbOtlAssociateAttributeDimension</td>
<td>Associates an attribute dimension with a base dimension</td>
</tr>
<tr>
<td>EsbOtlAssociateAttributeMember</td>
<td>Associates an attribute member with a base member</td>
</tr>
<tr>
<td>EsbOtlDisassociateAttributeDimension</td>
<td>Disassociates an attribute dimension from a base dimension</td>
</tr>
<tr>
<td>EsbOtlDisassociateAttributeMember</td>
<td>Disassociates an attribute member from a base member</td>
</tr>
<tr>
<td>EsbOtlFindAttributeMembers</td>
<td>Returns all base members that are associated with an attribute member</td>
</tr>
<tr>
<td>EsbOtlGetAssociatedAttributes</td>
<td>Returns all attribute members that are associated with a base member or dimension</td>
</tr>
<tr>
<td>EsbOtlGetAttributeInfo</td>
<td>Returns attribute information for a given attribute member or dimension</td>
</tr>
<tr>
<td>EsbOtlGetAttributeSpecifications</td>
<td>Retrieves attribute specifications for the outline</td>
</tr>
<tr>
<td>EsbOtlQueryAttributes</td>
<td>Does complex queries concerning attributes</td>
</tr>
<tr>
<td>EsbOtlSetAttributeSpecifications</td>
<td>Sets attribute specifications for the outline</td>
</tr>
</tbody>
</table>

See also the VB Main API the section called “VB Main API Attributes Functions”.

### VB Outline API Dynamic Time Series Functions

These functions enable and work with Dynamic Time Series members and aliases.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EsbOtlDeleteDTSMemberAlias</td>
<td>Deletes an alias name for a Dynamic Time Series member.</td>
</tr>
<tr>
<td>EsbOtlEnableDTSMember</td>
<td>Enables a new Dynamic Time Series members for the outline.</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>EsbOtlGetEnabledDTSMembers</td>
<td>Gets the defined Dynamic Time Series members for the outline.</td>
</tr>
<tr>
<td>EsbOtlGetDTSMemberAlias</td>
<td>Gets an alias name for a Dynamic Time Series member.</td>
</tr>
<tr>
<td>EsbOtlSetDTSMemberAlias</td>
<td>Sets an alias name for a Dynamic Time Series member.</td>
</tr>
</tbody>
</table>

**VB Outline API Generation Name Functions**

These functions perform operations on generation names.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EsbOtlGetGenName</td>
<td>Gets the generation name for the specified dimension and generation number</td>
</tr>
<tr>
<td>EsbOtlGetGenNames</td>
<td>Retrieves all generation names specified for a particular dimension</td>
</tr>
<tr>
<td>EsbOtlSetGenName</td>
<td>Sets the generation name for the specified dimension and generation number</td>
</tr>
<tr>
<td>EsbOtlDeleteGenName</td>
<td>Deletes the generation name of the specified dimension and level number</td>
</tr>
</tbody>
</table>

**VB Outline API Level Name Functions**

These functions perform operations on level names.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EsbOtlGetLevelName</td>
<td>Gets the level name of the specified dimension</td>
</tr>
<tr>
<td>EsbOtlGetLevelNames</td>
<td>Retrieves all level names specified for a particular dimension</td>
</tr>
<tr>
<td>EsbOtlSetLevelName</td>
<td>Sets the level name of the specified dimension</td>
</tr>
<tr>
<td>EsbOtlDeleteLevelName</td>
<td>Deletes the level name of the specified dimension</td>
</tr>
</tbody>
</table>

**VB Outline API Member Administration Functions**

These functions assist in managing the members of an outline.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EsbOtlAddMember</td>
<td>Adds a member</td>
</tr>
<tr>
<td>EsbOtlDeleteMember</td>
<td>Deletes a member</td>
</tr>
<tr>
<td>EsbOtlAddDimension</td>
<td>Adds a dimension</td>
</tr>
<tr>
<td>EsbOtlDeleteDimension</td>
<td>Deletes a dimension</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>------------------------------------</td>
</tr>
<tr>
<td>EsbOtlRenameMember</td>
<td>Renames a member</td>
</tr>
<tr>
<td>EsbOtlMoveMember</td>
<td>Moves a member</td>
</tr>
<tr>
<td>EsbOtlFindMember</td>
<td>Finds a member</td>
</tr>
<tr>
<td>EsbOtlGetMemberInfo</td>
<td>Gets member information</td>
</tr>
<tr>
<td>EsbOtlSetMemberInfo</td>
<td>Sets member information</td>
</tr>
</tbody>
</table>

**VB Outline API Member Alias Functions**

These functions perform operations on member aliases.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EsbOtlFindAlias</td>
<td>Finds a member with the specified alias name</td>
</tr>
<tr>
<td>EsbOtlGetMemberAlias</td>
<td>Gets the default member alias for a specific member in a specific alias table</td>
</tr>
<tr>
<td>EsbOtlSetMemberAlias</td>
<td>Sets the default member alias for a specific member in a specific alias table</td>
</tr>
<tr>
<td>EsbOtlDeleteMemberAlias</td>
<td>Deletes the default member alias for a specific member in a specific alias table</td>
</tr>
<tr>
<td>EsbOtlAddAliasCombination</td>
<td>Adds an alias combination to a member for a specific alias table</td>
</tr>
<tr>
<td>EsbOtlDeleteAliasCombination</td>
<td>Deletes an alias combination from a member for a specific alias table</td>
</tr>
<tr>
<td>EsbOtlGetNextAliasCombination</td>
<td>Returns the alias combinations for the specified member in the specified alias table</td>
</tr>
</tbody>
</table>

**VB Outline API Member Formula Functions**

These functions perform operations on member formulas.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EsbOtlGetMemberFormula</td>
<td>Gets the formula of the specified member</td>
</tr>
<tr>
<td>EsbOtlGetMemberLastFormula</td>
<td>Returns the last formula used to calculate the member</td>
</tr>
<tr>
<td>EsbOtlSetMemberFormula</td>
<td>Sets the formula for the specified member</td>
</tr>
<tr>
<td>EsbOtlDeleteMemberFormula</td>
<td>Deletes the formula of the specified member</td>
</tr>
</tbody>
</table>

**VB Outline API Member Traversal Functions**

These functions are used in traversing the outline tree.
### Function Description

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EsbOtlGetFirstMember</td>
<td>Returns a member handle to the first member in the outline; the first dimension defined in the outline</td>
</tr>
<tr>
<td>EsbOtlGetChild</td>
<td>Returns a member handle to the child of a member</td>
</tr>
<tr>
<td>EsbOtlGetParent</td>
<td>Returns a member handle to the parent of a member</td>
</tr>
<tr>
<td>EsbOtlGetNextSibling</td>
<td>Returns a member handle to the next sibling of a member</td>
</tr>
<tr>
<td>EsbOtlGetPrevSibling</td>
<td>Returns a member handle to the previous sibling of a member</td>
</tr>
<tr>
<td>EsbOtlGetNextSharedMember</td>
<td>Returns a member handle to the next shared member of a real member</td>
</tr>
</tbody>
</table>

### VB Outline API Outline Administration Functions

These functions assist in managing outlines.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EsbOtlGetOutlineInfo</td>
<td>Returns information about the outline file</td>
</tr>
<tr>
<td>EsbOtlGetUpdateTime</td>
<td>Returns the timestamp for the specified outline</td>
</tr>
<tr>
<td>EsbOtlSetOutlineInfo</td>
<td>Sets outline information</td>
</tr>
<tr>
<td>EsbOtlVerifyOutline</td>
<td>Verifies that an outline is correct</td>
</tr>
<tr>
<td>EsbOtlSortChildren</td>
<td>Sorts the children of an outline member</td>
</tr>
<tr>
<td>EsbOtlGenerateCurrencyOutline</td>
<td>Generates a currency outline based on the existing outline</td>
</tr>
</tbody>
</table>

### VB Outline API Outline Query Functions

These functions assist in making outline queries.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EsbOtlOpenOutlineQuery</td>
<td>Opens an existing outline</td>
</tr>
<tr>
<td>EsbOtlQueryMembers</td>
<td>Queries the outline, using a member handle</td>
</tr>
<tr>
<td>EsbOtlQueryMembersByName</td>
<td>Queries the outline, using a member name string</td>
</tr>
<tr>
<td>EsbOtlFreeMember</td>
<td>Frees the member array returned from EsbOtlQueryMembers()</td>
</tr>
</tbody>
</table>

### VB Outline API Setup and Cleanup Functions

These functions start and finish editing operations on an outline.
### VB Outline API User Attribute Functions

These functions perform operations on user attributes.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EsbOtlGetDimensionUserAttributes</td>
<td>Gets the user attributes of the specified dimension</td>
</tr>
<tr>
<td>EsbOtlGetUserAttributes</td>
<td>Gets the user attributes of the specified member</td>
</tr>
<tr>
<td>EsbOtlSetUserAttribute</td>
<td>Sets a user attribute for the specified member</td>
</tr>
<tr>
<td>EsbOtlDeleteUserAttribute</td>
<td>Deletes a user attribute of the specified member</td>
</tr>
</tbody>
</table>

### Visual Basic Outline API Function Reference

Consult the Contents pane for an alphabetical list of Visual Basic Outline API functions, which are prefaced with EsbOtl.

#### EsbOtlAddAliasCombination

Adds an alias combination to a member for a single alias table.

**Syntax**

```vba
EsbOtlAddAliasCombination (hOutline, hMember, pszAliasTable, pszAlias, pszCombination)
```

**Parameter**

- **hOutline** As Long
- **hMember** As Long
- **pszAliasTable** As String
- **pszAlias** As String
- **pszCombination** As String

**Description**

- **hOutline**: Outline context handle.
- **hMember**: Handle of member to create an alias combination for.
- **pszAliasTable**: Alias table to add the combination to. If this parameter is "", the default alias table is used.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pszAlias</td>
<td>Alias.</td>
</tr>
<tr>
<td>PszCombination</td>
<td>Member combination to associate with the alias. This can be a cross-dimensional member list.</td>
</tr>
</tbody>
</table>

**Notes**

The member handle cannot be a shared member. Aliases are not allowed for shared members.

**Return Value**

Returns 0 if successful; otherwise one of the following:

- OTLAPI_BAD_ALIASTABLE
- OTLAPI_ERR_ALIASSHARE
- OTLAPI_ERR_ILLEGALCOMBOALIAS
- OTLAPI_ERR_ILLEGALALIASSTRING
- OTLAPI_ERR_DUPLICATEALIAS

**Example**

```vbnet
Declare Function EsbOtlAddAliasCombination Lib "ESBOTLN" (ByVal hOutline As Long, ByVal hMember As Long, ByVal pszAliasTable As String, ByVal pszAlias As String, ByVal pszCombination As String) As Long

Sub ESB_EsbOtlAddAliasCombination()
    Dim sts As Long
    Dim Object As ESB_OBJDEF_T
    Dim hOutline As Long
    Dim MbrInfo As ESB_MBRINFO_T
    Dim hMemberJan As Long
    Object.hCtx = hCtx
    Object.Type = ESB_OBJTYPE_OUTLINE
    Object.AppName = "Sample"
    Object.DbName = "Basic"
    Object.FileName = "Basic"
    sts = EsbOtlOpenOutline(hCtx, Object, ESB_YES, ESB_YES, hOutline)
    If sts = 0 Then
        sts = EsbOtlFindMember(hOutline, "Jan", hMemberJan)
    End If
    If sts = 0 And hMemberJan <> 0 Then
        sts = EsbOtlAddAliasCombination(hOutline, hMemberJan, "Default", "alias combination", "Year->Market")
    End If
End Sub
```

**See Also**

- EsbOtlDeleteAliasCombination
- EsbOtlGetNextAliasCombination
EsbOtlAddDimension

Adds a dimension to the outline and sets the member's attributes.

Syntax

EsbOtlAddDimension (hOutline, pMemberInfo, hPrevSibling, pszDataMbr, phMember)

Parameter Description

hOutline Outline context handle.

pMemberInfo Member information structure defining the member and its attributes.

hPrevSibling Handle of previous sibling. If this field is ESB_NULL, the dimension becomes the first dimension in the outline. Otherwise, the dimension is placed after the dimension specified in hPrevSibling.

pszDataMbr Member name of a member in the new dimension that will receive the data values when the outline is restructured. If this field is ESB_NULL, the dimension member itself is used.

phMember Handle of new member returned from the API.

Notes

- This function specifies a member of the new dimension with which you can associate data when the outline is restructured.
- The ESB_MBRINFO_T structure must be created and filled before calling this function.
- To add an attribute dimension, you must call this function.
- To add a dimension that is not an attribute dimension, you can call this function or EsbOtlAddMember().
  - EsbOtlAddDimension() gives you the benefit of selecting any member in the added dimension to be assigned the data values associated with the existing dimensions.
  - If EsbOtlAddMember() is used, the top member (dimension) of the added dimension is used.
- In order for the pszDataMbr field to take effect, the outline must have been opened using EsbOtlOpenOutline() with the fKeepTrans flag set to ESB_YES.
- The member referred to in the pszDataMbr field is added to the new dimension using EsbOtlAddMember() after the dimension is created. If the referred to member doesn’t exist when restructuring takes place, the dimension member is used instead.
- For an attribute dimension, you must set the fields of ESB_MBRINFO_T as follows:

<table>
<thead>
<tr>
<th>Field</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>usConsolidation</td>
<td>ESB_UCALC_NOOP</td>
</tr>
<tr>
<td>Field</td>
<td>Setting</td>
</tr>
<tr>
<td>---------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>fTwoPass</td>
<td>ESB_FALSE</td>
</tr>
<tr>
<td>fExpense</td>
<td>ESB_FALSE</td>
</tr>
<tr>
<td>usConversion</td>
<td>ESB_CONV_NONE</td>
</tr>
<tr>
<td>usTimeBalance</td>
<td>ESB_TIMEBAL_NONE</td>
</tr>
<tr>
<td>usSkip</td>
<td>ESB_SKIP_NONE</td>
</tr>
<tr>
<td>usShare</td>
<td>ESB_SHARE_DYNCALEDCLUDENONE</td>
</tr>
<tr>
<td>usStorage</td>
<td>ESB_DIMTYPE_SPARSE</td>
</tr>
<tr>
<td>usCategory</td>
<td>ESB_CAT_ATTRIBUTE</td>
</tr>
<tr>
<td>usStorageCategory</td>
<td>ESB_STORECAT_ATTRIBUTE</td>
</tr>
</tbody>
</table>

**Attribute**

Attribute value. One of the following attribute member data types:

- ESB_ATTRMBRDT_BOOL
- ESB_ATTRMBRDT_DATETIME
- ESB_ATTRMBRDT_DOUBLE
- ESB_ATTRMBRDT_STRING

- An attribute dimension must be associated with a base dimension.
- Attribute dimensions must be placed after base dimensions and standard dimensions.

**Return Value**

Returns 0 if successful; otherwise one of the following:

- OTLAPI_BAD_CONSOL
- OTLAPI_BAD_MBRNAME
- OTLAPI_ERR_ADDDELETEDIMDYNAMICCALC
- OTLAPI_ERR_ADDNAMEUSED
- OTLAPI_ERR_BADSHARE
- OTLAPI_ERR_BADSKIP
- OTLAPI_ERR_BADSTORAGE
- OTLAPI_ERR_BADSTORAGECATEGORY
- OTLAPI_ERR_BADTIMEBAL
- OTLAPI_ERR_CURTOOMANYDIMS
- OTLAPI_ERR_ILLEGALBOOLEAN
- OTLAPI_ERR_ILLEGALCURRENCY
- OTLAPI_ERR_ILLEGALDATE
- OTLAPI_ERR_ILLEGALNUMERIC
Example

Declare Function EsbOtlAddDimension Lib "ESBOTLN" (ByVal hOutline As Long, pMemberInfo As ESB_MBRINFO_T, ByVal hPrevSibling As Long, ByVal pszDataMbr As String, phMember As Long) As Long

Sub ESB_OtlAddDimension()
Dim sts As Long
Dim NewInfo as ESB_OUTLINEINFO_T
Dim hOutline As Long
Dim MbrInfo As ESB_MBRINFO_T
Dim hDimMeasures As Long
NewInfo.usOutlineType = ESB_DBTYPE_NORMAL
NewInfo.fCaseSensitive = ESB_FALSE
NewInfo.fAutoConfigure = ESB_TRUE
sts = EsbOtlNewOutline(hLocalCtx, NewInfo, hOutline)
If sts = 0 Then
    MbrInfo.szMember = "Measures"
    sts = EsbOtlAddDimension(hOutline, MbrInfo, ESB_NULL, "Profit", hDimMeasures)
End If
End Sub

See Also

- EsbOtlAddMember
- EsbOtlDeleteDimension
- EsbOtlDeleteMember
- EsbOtlGetMemberInfo

EsbOtlAddMember

Adds a member to the outline and sets the member's attributes.

Syntax

EsbOtlAddMember (hOutline, pMemberInfo, hParent, hPrevSibling, phMember)

ByVal hOutline As Long
    pMemberInfo As ESB_MBRINFO_T
ByVal hParent As Long
ByVal hPrevSibling As Long
    phMember As Long

Parameter      Description
---            ---
hOutline       Outline context handle.
**Parameter**  
**Description**

<table>
<thead>
<tr>
<th>pMemberInfo</th>
<th>Member information structure defining the member and its attributes.</th>
</tr>
</thead>
<tbody>
<tr>
<td>hparent</td>
<td>Handle of parent. This field is used only if the hPrevSibling field is ESB_NULL.</td>
</tr>
<tr>
<td>hPrevSibling</td>
<td>Handle of previous sibling.</td>
</tr>
<tr>
<td>phMember</td>
<td>Handle of new member returned from the API.</td>
</tr>
</tbody>
</table>

**Notes**

- The ESB_MBRINFO_T structure must be created and filled before calling this function.
- The member name must be unique unless you are creating a shared member.
- Position of the added member:
  - The new member is inserted following the hPrevSibling member.
  - If the hPrevSibling field is ESB_NULL, the new member becomes the first child of the parent specified by hParent.
  - If both hParent and hPrevSibling are ESB_NULL, the new member becomes the first dimension in the outline.
- To add a shared member:
  - The shared member must be a zero-level (leaf node) member. (Shared members cannot have children.)
  - The actual member must already exist in the dimension.
  - Set the usShare field of the ESB_MBRINFO_T structure to ESB_SHARE_SHARE.
- To add a LABEL member:
  - You must first add the member without the label attribute set.
  - Next, add its children.
  - Then, use EsbOtlSetMemberInfo() to set the label tag of the label member. (A label member must have children.)
- To add an attribute member, set the fields of ESB_MBRINFO_T as follows:

<table>
<thead>
<tr>
<th>Field</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>usConsolidation</td>
<td>ESB_UCALC_NOOP</td>
</tr>
<tr>
<td>fTwoPass</td>
<td>ESB_FALSE</td>
</tr>
<tr>
<td>fExpense</td>
<td>ESB_FALSE</td>
</tr>
<tr>
<td>usConversion</td>
<td>ESB_CONV_NONE</td>
</tr>
<tr>
<td>usTimeBalance</td>
<td>ESB_TIMEBAL_NONE</td>
</tr>
<tr>
<td>usSkip</td>
<td>ESB_SKIP_NONE</td>
</tr>
<tr>
<td>usShare</td>
<td>ESB_SHARE_DYNCALCNOSTORE</td>
</tr>
<tr>
<td>Field</td>
<td>Setting</td>
</tr>
<tr>
<td>----------------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>usStorage</td>
<td>ESB_DIMTYPE_SPARSE</td>
</tr>
<tr>
<td>usCategory</td>
<td>ESB_CAT_ATTRIBUTE</td>
</tr>
<tr>
<td>usStorageCategory</td>
<td>ESB_STORECAT_ATTRIBUTE</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute value.</td>
<td>For an attribute dimension or zero-level (leaf node) attribute member, one of the following data types:</td>
</tr>
<tr>
<td></td>
<td>- Boolean (True</td>
</tr>
<tr>
<td></td>
<td>- Date (&quot;09/19/2006&quot;)</td>
</tr>
<tr>
<td></td>
<td>- Double (3.14)</td>
</tr>
<tr>
<td></td>
<td>- String (&quot;Hello&quot;)</td>
</tr>
<tr>
<td></td>
<td>For any attribute member, but not an attribute dimension:</td>
</tr>
<tr>
<td></td>
<td>- ESB_ATTRMBRDT_NONE = Everything else including empty.</td>
</tr>
</tbody>
</table>

- **Notes on Adding an Attribute Member:**
  - Adding a zero-level attribute member that is not of type ESB_ATTRMBRDT_STRING also sets the `szMember` field of the ESB_MBRINFO_T structure to the attribute member’s long name, using the specifications for the outline in the “ESB_ATTRSPECS_T” on page 1169 structure.
  - You must set `usCategory` and `usStorageCategory` for an attribute member, as well as an attribute dimension. (You need not set `usCategory` and `usStorageCategory` for a base member. You must set them for a base dimension only.)
  - Do not set the `szDimName` field of the ESB_MBRINFO_T structure.
  - For a zero-level attribute member that is not of type ESB_ATTRMBRDT_STRING, do not set the `Attribute` field. The attribute value is derived internally by converting the attribute member long name.
  - If you set an attribute member's data type to ESB_ATTRMBRDT_AUTO, Essbase does the following:
    - Sets the member’s data type to the data type of its dimension, if the member name can be converted to a value of that type.
    - If the member name cannot be converted to a value of the dimension's data type, sets the member’s data type to ESB_ATTRMBRDT_NONE.
    - For the first child member converted from ESB_ATTRMBRDT_AUTO to a data type other than ESB_ATTRMBRDT_NONE, converts the parent’s long name to a short name.
  - **To add a dimension:**
    - To add an attribute dimension, call `EsbOtlAddDimension()`. Do not call `EsbOtlAddMember()`.
    - To add a dimension that is not an attribute dimension, call either `EsbOtlAddDimension()` or `EsbOtlAddMember()`.
- `EsbOtlAddDimension()` gives you the benefit of selecting any member in the added dimension to be assigned the data values associated with the existing dimensions.
- If `EsbOtlAddMember()` is used, the top member (dimension) of the added dimension is assigned the data values associated with the existing dimensions.

**Return Value**

Returns 0 if successful; otherwise one of the following:

- OTLAPI_BAD_CONSOL
- OTLAPI_BAD_MBRNAME
- OTLAPI_ERR_ADDNAMEUSED
- OTLAPI_ERR_BADSHARE
- OTLAPI_ERR_BADSKIP
- OTLAPI_ERR_BADSTORAGE
- OTLAPI_ERR_BADSTORAGECATEGORY
- OTLAPI_ERR_BADTIMEBAL
- OTLAPI_ERR_CURTOOMANYDIMS
- OTLAPI_ERR_ILLEGALBOOLEAN
- OTLAPI_ERR_ILLEGALCURRENCY
- OTLAPI_ERR_ILLEGALDATE
- OTLAPI_ERR_ILLEGALNUMERIC
- OTLAPI_ERR_ILLEGALTAG
- OTLAPI_ERR_LEAFLABEL
- OTLAPI_ERR_NOSHAREPROTO
- OTLAPI_ERR_NOTIMEDIM

**Example**

```vba
Declare Function EsbOtlAddMember Lib "ESBOTLN" (ByVal hOutline As Long, pMemberInfo As ESB_MBRINFO_T, ByVal hParent As Long, ByVal hPrevSibling As Long, phMember As Long) As Long

Sub ESB_OtlAddMember()
    Dim sts As Long
    Dim Object As ESB_OBJDEF_T
    Dim hOutline As Long
    Dim MbrInfo As ESB_MBRINFO_T
    Dim hMemberProfit As Long
    Dim hNewMember As Long
    Object.hCtx = hCtx
    Object.Type = ESB_OBJTYPE_OUTLINE
    Object.AppName = "Sample"
    Object.DbName = "Basic"
    Object.FileName = "Basic"
    sts = EsbOtlOpenOutline(hCtx, Object, ESB_YES,
```
ESB_YES, hOutline)
If sts = 0 Then
  sts = EsbOtlFindMember(hOutline, "Profit", hMemberProfit)
End If
If sts = 0 And hMemberProfit <> 0 Then
  MbrInfo.szMember = "Inventory"
  sts = EsbOtlAddMember(hOutline, MbrInfo, ESB_NULL, hMemberProfit, hNewMember)
End If
End Sub

See Also

- EsbOtlAddDimension
- EsbOtlDeleteMember
- EsbOtlDeleteDimension
- EsbOtlSetMemberInfo
- EsbOtlFindMember

**EsbOtlAssociateAttributeDimension**

Associates an attribute dimension with a standard or base dimension.

**Syntax**

```vbnet
EsbOtl AssociateAttributeDimension (hOutline, BaseDimension, AttributeDimension)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>Handle to the outline</td>
</tr>
<tr>
<td>BaseDimension</td>
<td>Handle to the standard or base dimension</td>
</tr>
<tr>
<td>AttributeDimension</td>
<td>Handle to the base dimension</td>
</tr>
</tbody>
</table>

**Notes**

- The attribute dimension must be sparse.
- The standard or base dimension must be sparse.
- You must associate an attribute dimension with a standard or base dimension.
- You can associate more than one attribute dimension with a base dimension.
- You cannot associate an attribute dimension with more than one base dimension.

**Return Value**

Returns STS = 0 when successful. Otherwise, returns an error code.

**Access**

This function requires no special privileges.
Example

Sub ESB_OtlAssociateAttributeDimension()
    ' NOTE: 'Out' is a sub to print the output within quotes to a listbox or text box
    Dim sts As Long
    Dim hOutline As Long
    Dim BaseMbr As Long
    Dim AttrMbr As Long

    hOutline = ESB_OtlOpenOutline
    If hOutline = vbNull Then Out "ESB_OtlOpenOutline() failed: " & sts: Exit Sub
    ' abstract function (using EsbOtlFindMember()) to get member handle, while passing in a prompt string
    BaseMbr = ESB_OtlFindMember("Enter base dimension: ")
    If BaseMbr = vbNull Then
        Out "ESB_OtlFindMember() failed."
        Exit Sub
    End If
    ' abstract function (using EsbOtlFindMember()) to get member handle, while passing in a prompt string
    AttrMbr = ESB_OtlFindMember("Enter attribute dimension: ")
    If AttrMbr = vbNull Then Out "ESB_OtlFindMember() failed."
    Exit Sub

    sts = EsbOtlAssociateAttributeDimension(ghOutline, BaseMbr, AttrMbr)
    ' abstract sub to call EsbOtlVerifyOutline(), ESBOTLWriteOutline(), EsbOtlRestructure(), EsbUnlockObject() and ' EsbOtlCloseOutline() as needed
    tuckinoutline
    If sts <> 0 Then Out "EsbOtlAssociateAttributeDimension failed: " & sts: Exit Sub
End Sub

See Also

- EsbCheckAttributes
- EsbGetAssociatedAttributesInfo
- EsbGetAttributeInfo
- EsbGetAttributeSpecifications
- EsbOtlAssociateAttributeMember
- EsbOtlDisassociateAttributeDimension
- EsbOtlDisassociateAttributeMember
- EsbOtlFindAttributeMembers
- EsbOtlGetAssociatedAttributes
- EsbOtlGetAttributeInfo
- EsbOtlGetAttributeSpecifications
- EsbOtlQueryAttributes
- EsbOtlSetAttributeSpecifications

EsbOtlAssociateAttributeMember

 Associates an attribute member with a standard or base member.
Syntax

EsbOtlAssociateAttributeMember (hOutline, BaseMember, AttributeMember)
ByVal hOutline As Long
ByVal BaseMember As Long
ByVal AttributeMember As Long

Parameter Description
hOutline Handle to the outline
BaseMember Handle to the standard of base member
AttributeMember Handle to the attribute member

Notes

- Before you associate an attribute member with a standard or base member using this function, associate the dimension of the attribute member with the dimension of the standard or base member using EsbOtlAssociateAttributeDimension().
- You cannot associate an attribute member with a base dimension.
- Only a zero-level attribute member can associate with a standard or base member.
- You cannot associate members of a given attribute dimension with base members that are at different levels from each other.
- You cannot associate more than one member of an attribute dimension with a base member.
- You can associate members of more than one attribute dimension with a base member.

Return Value

Returns STS = 0 when successful. Otherwise, returns an error code.

Access

This function requires no special privileges.

Example

Sub ESB_OtlAssociateAttributeMember()
    ' NOTE: 'Out' is a sub to print the output within quotes to a listbox or text box
    Dim BaseMbr As Long
    Dim AttrMbr As Long
    Dim sts as long
    Dim hOutline as long
    hOutline = ESB_OtlOpenOutline
    If hOutline = vbNull Then Out "ESB_OtlOpenOutline() failed: " & sts: Exit Sub
    BaseMbr = ESB_OtlFindMember("Enter base dimension: ")
    If BaseMbr = vbNull Then Out "No valid member found." End If
    Out "ESB_OtlAssociateAttributeDimension() failed." Exit Sub
    AttrMbr = ESB_OtlFindMember("Enter attribute dimension: ")
End Sub
If AttrMbr = vbNull Then
Out "No valid member found."
Out "ESB_OtlAssociateAttributeMember() failed."
Exit Sub
End If

sts = EsbOtlAssociateAttributeMember(hOutline, BaseMbr, AttrMbr)
' abstract sub to call EsbOtlVerifyOutline(), ESBOTLWriteOutline(),
EsbOtlRestructure(), EsbUnlockObject() and
' EsbOtlCloseOutline() as needed

tuckinoutline
If sts <> 0 Then Out "EsbOtlAssociateAttributeMember failed" & sts: Exit Sub
ESB_OtlGetAttributeInfo
End Sub

See Also

- EsbCheckAttributes
- EsbGetAssociatedAttributesInfo
- EsbGetAttributeInfo
- EsbGetAttributeSpecifications
- EsbOtlAssociateAttributeDimension
- EsbOtlDisassociateAttributeDimension
- EsbOtlDisassociateAttributeMember
- EsbOtlFindAttributeMembers
- EsbOtlGetAssociatedAttributes
- EsbOtlGetAttributeInfo
- EsbOtlGetAttributeSpecifications
- EsbOtlQueryAttributes
- EsbOtlSetAttributeSpecifications

EsbOtlClearAliasTable

Clears all entries from an existing alias table without deleting it.

Syntax

EsbOtlClearAliasTable (hOutline, pszAliasTable)
ByVal hOutline As Long
ByVal pszAliasTable As String

Parameter | Description
--- | ---
hOutline | Outline context handle.
pszAliasTable | Name of alias table to clear. Use "" or "Default" for the default table.

Notes

When clearing aliases from an alias table, language codes associated with the alias table are removed.

Return Value

Returns 0 if successful; otherwise:
OTLAPI_BAD_ALIASTABLE

Example

Declare Function EsbOtIClearAliasTable Lib "ESBOTLN"
(ByVal hOutline As Long, ByVal pszAliasTable As String) As Long

Sub ESB_OtIClearAliasTable()
Dim sts As Long
Dim Object As ESB_OBJDEF_T
Dim hOutline As Long
Object.hCtx = hCtx
Object.Type = ESB_OBJTYPE_OUTLINE
Object.AppName = "Sample"
Object.DbName = "Basic"
Object.FileName = "Basic"
sts = EsbOtICreateAliasTable(hCtx, Object,
ESB_YES, ESB_YES, hOutline)
If sts = 0 Then
    sts = EsbOtIClearAliasTable(hOutline, "Default")
End If
End Sub

See Also

- EsbOtICreateAliasTable
- EsbOtICopyAliasTable
- EsbOtIRenameAliasTable
- EsbOtIDeleteAliasTable
- EsbOtISetAliasTableLanguage

EsbOtIClearAliasTableLanguages

Clears the set of language codes associated with the specified alias table.

Syntax

ESB_FUNC_M EsbOtIClearAliasTableLanguages (hOutline, pszAliasTable)
ByVal hOutline As Long
ByVal pszAliasTable As String

Parameter Description

hOutline Handle to the outline.

pszAliasTable Name of the alias table from which to remove all associated language codes.

Return Value

- If successful, returns 0.
- If unsuccessful, returns the error OTLAPI_BAD_ALIASTABLE (invalid alias table).

Access

This function does not require special privileges.
Example

Declare Function EsbOtlGetAliasTableLanguages Lib "esbotln" (ByVal hOutline As Long, ByVal pszAliasTable As String, pulCount As Long) As Long
Declare Function EsbOtlSetAliasTableLanguage Lib "esbotln" (ByVal hOutline As Long, ByVal pszAliasTable As String, ByVal pszLanguageCode As String) As Long
Declare Function EsbOtlClearAliasTableLanguages Lib "esbotln" (ByVal hOutline As Long, ByVal pszAliasTable As String) As Long

Sub ESB_Sub ()
Dim sts As Long
Dim Object As ESB_OBJDEF_T
Dim hOutline As Long
Dim Items As Long
Dim AliasLang As String * ESB_ALIASNAMELEN

Object.hCtx = hCtx
Object.Type = ESB_OBJTYPE_OUTLINE
Object.AppName = "Sample"
Object.DbName = "Basic"
Object.FileName = "Basic"
sts = EsbOtlOpenOutline(hCtx, Object, ESB_YES, ESB_YES, hOutline)
If sts = 0 Then
    sts = EsbOtlCreateAliasTable(hOutline, "French Alias Table")
End If
If sts = 0 Then
    sts = EsbOtlSetAliasTableLanguage(hOutline, "French Alias Table", "fr")
End If
If sts = 0 Then
    sts = EsbOtlSetAliasTableLanguage(hOutline, "French Alias Table", "fr-CA")
End If
If sts = 0 Then
    sts = EsbOtlGetAliasTableLanguages(hOutline, "French Alias Table", Items)
If sts = 0 Then
    For N = 1 To Items
        sts = EsbGetNextItem(hCtx, ESB_ALIASLANG_TYPE, ByVal AliasLang)
        Next
End If
End If
If sts = 0 Then
    sts = EsbOtlClearAliasTableLanguages(hOutline, "French Alias Table")
End If

End Sub

See Also
- EsbOtlGetAliasTableLanguages
- EsbOtlSetAliasTableLanguage
**EsbOtlCloseOutline**

Frees all information associated with the outline.

**Syntax**

```vbscript
EsbOtlCloseOutline (hOutline)
ByVal hOutline As Long
```

**Parameter Description**

hOutline  Outline context handle.

**Notes**

- This function should always be called if `EsbOtlNewOutline()` or `EsbOtlOpenOutline()` is called.
- If the object was locked when it was opened, you should call `EsbUnlockObject()` before making this call.

**Return Value**

Returns 0 if successful.

**Example**

```vbscript
Declare Function EsbOtlCloseOutline Lib "ESBOTLN" (ByVal hOutline As Long) As Long

Sub ESB_OtlCloseOutline()
Dim sts As Long
Dim Object As ESB_OBJDEF_T
Dim hOutline As Long
Object.hCtx = hCtx
Object.Type = ESB_OBJTYPE_OUTLINE
Object.AppName = "Sample"
Object.DbName = "Basic"
Object.FileName = "Basic"
sts = EsbOtlOpenOutline(hCtx, Object,
ESB_YES, ESB_YES, hOutline)
'body of code...
If sts = 0 Then
    sts = ESBOTLNriteOutline(hOutline, Object)
End If
'restructure outline using EsbOtlRestructure()
If sts = 0 Then
    sts = EsbOtlCloseOutline(hOutline)
End If
End Sub
```

**See Also**

- `EsbOtlOpenOutline`
- `EsbOtlWriteOutline`
- `EsbOtlRestructure`
**EsbOtlCopyAliasTable**

Copies an alias table to another alias table.

**Syntax**

```
EsbOtlCopyAliasTable (hOutline, pszSourceAliasTable, pszDestAliasTable, fMerge)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>Outline context handle.</td>
</tr>
<tr>
<td>pszSourceAliasTable</td>
<td>Name of alias table to copy from. If this parameter is &quot;&quot;, the default alias table is used.</td>
</tr>
<tr>
<td>pszDestAliasTable</td>
<td>Name of alias table to copy to.</td>
</tr>
<tr>
<td>fMerge</td>
<td>Set to ESB_YES to merge the source file into the existing destination alias table. Set to ESB_NO to clear the destination alias table before copying.</td>
</tr>
</tbody>
</table>

**Notes**

- If the destination alias table does not exist, it is created. If the destination alias table exists, it is cleared first, unless the fMerge flag is set to ESB_YES.
- The maximum number of alias tables in a single block storage or aggregate storage database outline (including the default table) is 32.
- When copying an alias table, language codes associated with the alias table are removed from the copied alias table.

**Return Value**

Returns 0 if successful; otherwise one of the following:

- OTLAPI_BAD_ALIAS_TABLE
- OTLAPI_ERR_MAX_ALIAS_TABLES
- OTLAPI_ERR_ALIAS_TABLE_NAME

**Example**

```Visual Basic
Declare Function EsbOtlCopyAliasTable Lib "ESBOTLN" (ByVal hOutline As Long, ByVal pszSourceAliasTable As String, ByVal pszDestAliasTable As String, ByVal fMerge As Integer) As Long

Sub ESB_OtlCopyAliasTable()
Dim sts As Long
Dim Object As ESB_OBJDEF_T
Dim hOutline As Long
Object.hCtx = hCtx
Object.Type = ESB_OBJTYPE_OUTLINE
Object.AppName = "Sample"
Object.DbName = "Basic"
```

Visual Basic Outline API Function Reference 1485
Object.FileName = "Basic"
sts = EsbOtlOpenOutline(hCtx, 
Object, ESB_YES, ESB_YES, hOutline)
If sts = 0 Then
    sts = EsbOtlCopyAliasTable 
    (hOutline, "", "Alias Table 2", ESB_YES)
End If
End Sub

See Also

- EsbOtlCreateAliasTable
- EsbOtlClearAliasTable
- EsbOtlRenameAliasTable
- EsbOtlDeleteAliasTable
- EsbOtlSetAliasTableLanguage

**EsbOtlCreateAliasTable**

Creates an empty alias table in the outline.

**Syntax**

```
EsbOtlCreateAliasTable (hOutline, pszAliasTable)
ByVal hOutline As Long
ByVal pszAliasTable As String
```

**Parameter**  
- **hOutline**: Outline context handle.
- **pszAliasTable**: Name of alias table to create.

**Notes**

- An alias table named "Default" cannot be created, since the default alias table always exists.
- The maximum number of alias tables in a single block storage or aggregate storage database outline (including the default table) is 32.
- You can specify multiple language codes for an alias table, using the `EsbOtlSetAliasTableLanguage` API. When you create an alias table, a language code is not specified.

**Return Value**

Returns 0 if successful; otherwise one of the following:

- OTLAPI_ERR_ALIASTABLEEXISTS
- OTLAPI_ERR_MAXALIASTABLES
- OTLAPI_ERR_ALIASTABLENAME
Example

Declare Function EsbOtlCreateAliasTable Lib "ESBOTLN" (ByVal hOutline As Long, ByVal pszAliasTable As String) As Long

Sub ESB_OtlCreateAliasTable()
Dim sts As Long
Dim Object As ESB_OBJDEF_T
Dim hOutline As Long
Object.hCtx = hCtx
Object.Type = ESB_OBJTYPE_OUTLINE
Object.AppName = "Sample"
Object.DbName = "Basic"
Object.FileName = "Basic"
sts = EsbOtlOpenOutline(hCtx, Object, ESB_YES, ESB_YES, hOutline)
If sts = 0 Then
    sts = EsbOtlCreateAliasTable(hOutline, "Alias Table 1")
End If
End Sub

See Also

- EsbOtlCopyAliasTable
- EsbOtlRenameAliasTable
- EsbOtlDeleteAliasTable
- EsbOtlSetAliasTableLanguage

**EsbOtlDeleteAliasCombination**

Deletes an alias combination from a member for a single alias table.

**Syntax**

```vba
EsbOtlDeleteAliasCombination (hOutline, hMember, pszAliasTable, pszAlias)
```

**Parameter**  
**Description**

- hOutline: Outline context handle.
- hMember: Handle of member to remove the alias combination from.
- pszAliasTable: Alias table to remove the combination from. If this parameter is "", the default alias table is used.
- pszAlias: Alias to remove.

**Return Value**

Returns 0 if successful; otherwise one of the following:

- OTLAPI_BAD_ALIASTABLE
Example

Declare Function EsbOtlDeleteAliasCombination Lib "ESBOTLN" (ByVal hOutline As Long, ByVal hMember As Long, ByVal pszAliasTable As String, ByVal pszAlias As String) As Long

Sub ESB_OtlDeleteAliasCombination()
  Dim sts As Long
  Dim Object As ESB_OBJDEF_T
  Dim hOutline As Long
  Dim hMemberJan As Long
  Object.hCtx = hCtx
  Object.Type = ESB_OBJTYPE_OUTLINE
  Object.AppName = "Sample"
  Object.DbName = "Basic"
  Object.FileName = "Basic"
  sts = EsbOtlOpenOutline(hCtx, Object, ESB_YES, ESB_YES, hOutline)
  If sts = 0 Then
    sts = EsbOtlFindMember(hOutline, "Jan", hMemberJan)
  End If
  If sts = 0 And hMemberJan <> 0 Then
    sts = EsbOtlDeleteAliasCombination(hOutline, hMemberJan, "Default", "alias combination")
  End If
End Sub

See Also

- EsbOtlAddAliasCombination
- EsbOtlGetNextAliasCombination

EsbOtlDeleteAliasTable

Deletes the specified alias table from the outline, clearing all of its entries.

Syntax

EsbOtlDeleteAliasTable (hOutline, pszAliasTable)
ByVal hOutline As Long
ByVal pszAliasTable As String

Parameter  Description
hOutline    Outline context handle.
pszAliasTable  Name of alias table to delete.

Notes

You cannot delete the default alias table.
**Return Value**

Returns 0 if successful; otherwise one of the following:

- OTLAPI_BAD_ALIAS_TABLE
- OTLAPI_ERR_DELETEDEFALIAS

**Example**

```vbnet
Declare Function EsbOtlDeleteAliasTable Lib "ESBOTLN" (ByVal hOutline As Long, ByVal pszAliasTable As String) As Long

Sub ESB_OtlDeleteAliasTable()
    Dim sts As Long
    Dim Object As ESB_OBJDEF_T
    Dim hOutline As Long
    Object.hCtx = hCtx
    Object.Type = ESB_OBJTYPE_OUTLINE
    Object.AppName = "Sample"
    Object.DbName = "Basic"
    Object.FileName = "Basic"
    sts = EsbOtlOpenOutline(hCtx, Object, ESB_YES, ESB_YES, hOutline)
    If sts = 0 Then
        sts = EsbOtlDeleteAliasTable(hOutline, "Alias Table 1")
    End If
End Sub
```

**See Also**

- EsbOtlCreateAliasTable
- EsbOtlCopyAliasTable
- EsbOtlRenameAliasTable
- EsbOtlClearAliasTable

---

**EsbOtlDeleteDimension**

Deletes a dimension from the outline. The call also specifies a member of the dimension being deleted from which to keep data when the outline is restructured.

**Syntax**

```vbnet
EsbOtlDeleteDimension (hOutline, hMember, pszDataMbr)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>Outline context handle.</td>
</tr>
<tr>
<td>hMember</td>
<td>Handle of member to delete.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
</tr>
<tr>
<td>pszDataMbr</td>
<td>Member name in the dimension to be deleted from which data will be saved when the outline is restructured. If this field is &quot;&quot;, the dimension is used.</td>
</tr>
</tbody>
</table>

**Notes**

- All shared members of the dimension and its descendants are deleted.
- All members of the dimension are deleted.
- To delete a dimension, you can use this call or `EsbOtlDeleteMember()`. `EsbOtlDeleteDimension()` gives you the benefit of selecting a member of the deleted dimension whose data values will be used as the data values for the other dimensions when the database is restructured. If `EsbOtlDeleteMember()` is used, the data values of the top member (dimension) of the deleted dimension are used.
- In order for the `pszDataMbr` field to take effect, the outline must have been opened with `EsbOtlOpenOutline()` with the `fKeepTrans` flag set to ESB_YES.

**Return Value**

Returns 0 if successful; otherwise one of the following:

OTLAPI_ERR_NOTIMEDIM

**Example**

```visualbasic
Declare Function EsbOtlDeleteDimension Lib "ESBOTLN" (ByVal hOutline As Long, ByVal hMember As Long, ByVal pszDataMbr As String) As Long

Sub ESB_OtlDeleteDimension()
    Dim sts As Long
    Dim Object As ESB_OBJDEF_T
    Dim hOutline As Long
    Dim hMemberScenario As Long
    Object.hCtx = hCtx
    Object.Type = ESB_OBJTYPE_OUTLINE
    Object.AppName = "Sample"
    Object.DbName = "Basic"
    Object.FileName = "Basic"
    sts = EsbOtlOpenOutline(hCtx, Object, ESB_YES, ESB_YES, hOutline)
    If sts = 0 Then
        sts = EsbOtlFindMember(hOutline, "Scenario", hMemberScenario)
    End If
    If sts = 0 And hScenario <> 0 Then
        sts = EsbOtlDeleteDimension(hOutline, hMemberScenario, "Actual")
    End If
End Sub
```

**See Also**

- `EsbOtlDeleteMember`
- `EsbOtlAddDimension`
EsbOtlDeleteDTSMemberAlias

Deletes an alias name for a DTS member.

Syntax

EsbOtlDeleteDTSMemberAlias (hOutline, pszDTSMember, pszAliasTable)
ByVal hOutline As Long
ByVal pszDTSMember As String
ByVal pszAliasTable As String

Parameter Description
hOutline Esbbase outline handle returned from the EsbOtlOpenOutlineQuery call.
pszDTSMember Name of the DTS member which provides the alias.
pszAliasTable Name of the alias table which provides the alias. If NULL, use the default alias table.

Notes

This function only clears the alias name. It does not disable the DTS member (see EsbOtlEnableDTSMember).

Return Value

If successful the return value is zero. Otherwise, one of the following is returned:

- OTLAPI_ERR_DTSMBRNOTDEFINED
- OTLAPI_BAD_ALIASTABLE
- OTLAPI_ERR_NOALIAS

Example

Public Sub ESB_OtlDeleteDTSMemberAlias()
    Dim DTSMember As String * ESB_MBRNAMELEN
    Dim AliasTable As String * ESB_ALIASNAMELEN

    DTSMember = "H-T-D"
    AliasTable = "default"

    sts = EsbOtlDeleteDTSMemberAlias(hOutline, _
    DTSMember, AliasTable)
End Sub

See Also

- EsbOtlEnableDTSMember
- EsbOtlGetEnabledDTSMembers
- EsbOtlGetDTSMemberAlias
- EsbOtlSetDTSMemberAlias
**EsbOtlDeleteGenName**

Deletes the name of a specific generation within a dimension. Generation names are explicitly added to the outline with **EsbOtlSetGenName**.

**Syntax**

```plaintext
EsbOtlDeleteGenName (hOutline, pszDimension, usGen)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>Outline context handle.</td>
</tr>
<tr>
<td>pszDimension</td>
<td>Name of the dimension that contains the generation.</td>
</tr>
<tr>
<td>usGen</td>
<td>Number of generation for which to delete name. Leaf members are level 0.</td>
</tr>
</tbody>
</table>

**Return Value**

Returns 0 if successful; otherwise one of the following:

- **OTLAPI_NO_GENLEVELNAME**
- **OTLAPI_ERR_NOTADIM**

**Example**

```visualbasic
Declare Function EsbOtlDeleteGenName Lib "ESBOTLN" (ByVal hOutline As Long, ByVal pszDimension As String, ByVal usGen As Integer) As Long

Sub ESB_OtlDeleteGenName()
    Dim sts As Long
    Dim Dimension As String
    Dim GenNum As Integer
    Dim Object As ESB_OBJDEF_T
    Object.hCtx = hCtx
    Object.Type = ESB_OBJTYPE_OUTLINE
    Object.AppName = "Sample"
    Object.DbName = "Basic"
    Object.FileName = "Basic"
    sts = EsbOtlOpenOutline(hCtx, Object, ESB_YES, ESB_YES, hOutline)
    '**********************************
    '******* Delete Generation Name ***
    '**********************************
    Dimension = "Year"
    GenNum = 2
    GenName = "Qtr1 Qtr2 Qtr3 Qtr4"
    If sts = 0 Then
        sts = EsbOtlDeleteGenName(hOutline, Dimension, GenNum)
    End If
End Sub
```

1492 Visual Basic Outline API Functions
EsbOtlDeleteLevelName

Deletes the name for a specific level within a dimension. Level names are explicitly added to the outline with EsbOtlSetLevelName.

Syntax

EsbOtlDeleteLevelName (hOutline, pszDimension, usLevel)

ByVal hOutline As Long
ByVal pszDimension As String
ByVal usLevel As Integer

Parameter Description

hOutline Outline context handle.

pszDimension Name of dimension that contains the level name.

usLevel Number of level for which to delete name. Leaf members are level 0.

Return Value

Returns 0 if successful; otherwise one of the following:

- OTLAPI_NO_GENLEVELNAME
- OTLAPI_ERR_NOTADIM

Example

Declare Function EsbOtlDeleteLevelName Lib "ESBOTLN" (ByVal hOutline As Long, ByVal pszDimension As String, ByVal usLevel As Integer) As Long

Sub ESB_OtlDeleteLevelName()
Dim sts As Long
Dim Dimension As String
Dim LevelNum As Integer
Dim Object As ESB_OBJDEF_T
Dim hOutline As Long
Object.hCtx = hCtx
Object.Type = ESB_OBJTYPE_OUTLINE
Object.AppName = "Sample"
Object.DbName = "Basic"
Object.FileName = "Basic"
sts = EsbOtlOpenOutline(hCtx, Object, ESB_YES, ESB_YES, hOutline)
'****************************
'***** Delete Level Name *****
'****************************
Dimension = "Year"
LevelNum = 1
LevelName = "Month"
If sts = 0 Then
    sts = EsbOtlDeleteLevelName(hOutline, Dimension, LevelNum)
End If
End Sub

See Also
- EsbOtlGetLevelName
- EsbOtlGetLevelNames
- EsbOtlSetLevelName

EsbOtlDeleteMember

Deletes a member from the outline.

Syntax

EsbOtlDeleteMember (hOutline, hMember)
ByVal hOutline As Long
ByVal hMember As Long

Parameter Description

hOutline  Outline context handle.

hMember  Handle of member to delete.

Notes

- All descendants of the member are deleted.
- All shared members of this member and its descendants are deleted.
- If a shared member, only the specified member is deleted.
- To delete a dimension, you can use this call or EsbOtlDeleteDimension().
  EsbOtlDeleteDimension() gives you the benefit of selecting a member of the deleted
  dimension whose data values will be used as the data values for the other dimensions when
  the database is restructured. If EsbOtlDeleteMember() is used, the data values of the top
  member (dimension) of the deleted dimension are used.

Return Value

Returns 0 if successful; otherwise one of the following:

- OTLAPI_ERR_LEAFLABEL
- OTLAPI_ERR_NOTIMEDIM

Example

Declare Function EsbOtlDeleteMember Lib "ESBOTLN" (ByVal hOutline As Long, ByVal
hMember As Long) As Long
Sub ESB_OtlDeleteMember()
Dim sts As Long
Dim Object As ESB_OBJDEF_T
Dim hOutline As Long
Dim hCOGS As Long
Object.hCtx = hCtx
Object.Type = ESB_OBJTYPE_OUTLINE
Object.AppName = "Sample"
Object.DbName = "Basic"
Object.FileName = "Basic"
sts = EsbOtlOpenOutline(hCtx, Object,
ESB_YES, ESB_YES, hOutline)
If sts = 0 Then
    sts = EsbOtlFindMember(hOutline, "COGS", hCOGS)
End If
If sts = 0 And hCOGS <> 0 Then
    sts = EsbOtlDeleteMember(hOutline, hCOGS)
End If
End Sub

See Also
- EsbOtlDeleteDimension
- EsbOtlAddMember
- EsbOtlAddDimension
- EsbOtlFindMember
- EsbOtlGetMemberInfo

**EsbOtlDeleteMemberAlias**

Deletes the default member alias for a specified member in a specified alias table.

**Syntax**

```vbnet
EsbOtlDeleteMemberAlias (hOutline, hMember, pszAliasTable)
ByVal hOutline As Long
ByVal hMember As Long
ByVal pszAliasTable As String
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>Outline context handle.</td>
</tr>
<tr>
<td>hMember</td>
<td>Handle of member to delete the alias from.</td>
</tr>
<tr>
<td>pszAliasTable</td>
<td>Alias table to delete the alias from. If this parameter is &quot;&quot;, the default table is used.</td>
</tr>
</tbody>
</table>

**Return Value**

Returns 0 if successful; otherwise one of the following:

OTLAPI_ERR_NOALIAS
Example

Declare Function EsbOtlDeleteMemberAlias Lib "ESBOTLN" (ByVal hOutline As Long, ByVal hMember As Long, ByVal pszAliasTable As String) As Long

Sub ESB_OtlDeleteMemberAlias()
    Dim sts As Long
    Dim Object As ESB_OBJDEF_T
    Dim hOutline As Long
    Dim hMemberYear As Long
    Object.hCtx = hCtx
    Object.Type = ESB_OBJTYPE_OUTLINE
    Object.AppName = "Sample"
    Object.DbName = "Basic"
    Object.FileName = "Basic"
    sts = EsbOtlOpenOutline(hCtx, Object, ESB_YES, ESB_YES, hOutline)
    If sts = 0 Then
        sts = EsbOtlFindMember(hOutline, "Year", hMemberYear)
    End If
    If sts = 0 And hMemberYear <> 0 Then
        sts = EsbOtlDeleteMember Alias(hOutline, hMemberYear,"")
    End If
End Sub

See Also

- EsbOtlGetMemberAlias
- EsbOtlSetMemberAlias

EsbOtlDeleteMemberFormula

Deletes the formula for the specified member.

Syntax

EsbOtlDeleteMemberFormula (hOutline, hMember)

ByVal hOutline As Long
ByVal hMember As Long

Parameter Description

- hOutline: Outline context handle.
- hMember: Member handle.

Return Value

Returns 0 if successful; otherwise one of the following:

OTLAPI_ERR_NOFORMULA

Example

Declare Function EsbOtlDeleteMemberFormula Lib "ESBOTLN" (ByVal hOutline As Long,
Sub ESB_OtlDeleteMemberFormula()
    Dim sts As Long
    Dim Object As ESB_OBJDEF_T
    Dim hOutline As Long
    Dim hMemberProfit As Long
    Object.hCtx = hCtx
    Object.Type = ESB_OBJTYPE_OUTLINE
    Object.AppName = "Sample"
    Object.DbName = "Basic"
    Object.FileName = "Basic"
    sts = EsbOtlOpenOutline(hCtx, Object, ESB_YES, ESB_YES, hOutline)
    If sts = 0 Then
        sts = EsbOtlFindMember(hOutline, "Profit", hMemberProfit)
    End If
    If sts = 0 And hMemberProfit <> 0 Then
        sts = EsbOtlDeleteMemberFormula(hOutline, hMemberProfit)
    End If
End Sub

See Also

- EsbOtlSetMemberFormula
- EsbOtlDeleteMemberFormula

**EsbOtlDeleteUserAttribute**

Deletes a user-defined attribute for a member.

**Syntax**

```
EsbOtlDeleteUserAttribute (hOutline, hMember, pszString)
```

**Parameter Description**

- hOutline: Outline context handle.
- hMember: Handle of member whose attribute you are deleting.
- pszString: User attribute string.

**Notes**

The caller passes in a string to identify the attribute.

**Return Value**

Returns 0 if successful; otherwise: OTLAPI_NO_USERATTR.
Example

Declare Function EsbOtlDeleteUserAttribute Lib "ESBOTLN" (ByVal hOutline As Long, ByVal hMember As Long, ByVal pszString As String) As Long

Sub ESB_OtlDeleteUserAttribute()
    Dim sts As Long
    Dim Object As ESB_OBJDEF_T
    Dim hOutline As Long
    Dim hMember As Long
    Dim AttributeList As String
    Object.hCtx = hCtx
    Object.Type = ESB_OBJTYPE_OUTLINE
    Object.AppName = "Sample"
    Object.DbName = "Basic"
    Object.FileName = "Basic"
    AttributeList = "Read Write"
    sts = EsbOtlOpenOutline(hCtx, Object, ESB_YES, ESB_YES, hOutline)
    If sts = 0 Then
        sts = EsbOtlFindMember(hOutline, "Jan", hMember)
    End If
    If sts = 0 And hMember <> 0 Then
        '********************
        ' Delete User Attributes
        '********************
        sts = EsbOtlDeleteUserAttribute(hOutline, hMember, AttributeList)
    End If
End Sub

See Also

- EsbOtlGetUserAttributes
- EsbOtlSetUserAttribute

**EsbOtlDisassociateAttributeDimension**

Disassociates an attribute dimension from a base dimension.

Syntax

```
EsbOtlDisassociateAttributeDimension (hOutline, BaseDimension, AttributeDimension)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>Handle to the outline</td>
</tr>
<tr>
<td>BaseDimension</td>
<td>Handle to the base dimension</td>
</tr>
<tr>
<td>AttributeDimension</td>
<td>Handle to the attribute dimension</td>
</tr>
</tbody>
</table>
Notes

- When you disassociate an attribute dimension from a base dimension, you disassociate all members of the attribute dimension from members of the base dimension.

- A disassociated attribute dimension may not remain in the outline when being verified and written to disk. A suggested method for dealing with this situation is to delete the now disassociated dimension from the outline.

Return Value

Returns STS = 0 when successful. Otherwise, returns an error code.

Access

This function requires no special privileges.

Example

```visualbasic
Sub ESB_OtlDisAssociateAttributeDimension()
    ' NOTE: 'Out' is a sub to print the output within quotes to a listbox or text box
    Dim sts as long
    Dim hOutline as long
    Dim BaseMbr As Long
    Dim AttrMbr As Long
    hOutline = ESB_OtlOpenOutline
    If hOutline = vbNull Then Out "ESB_OtlOpenOutline() failed: "; Exit Sub
    BaseMbr = ESB_OtlFindMember("Enter base dimension: ")
    If BaseMbr = vbNull Then MsgBox "ESB_OtlDisAssociateAttributeDimension() failed.": Exit Sub
    AttrMbr = ESB_OtlFindMember("Enter attribute dimension: ")
    If AttrMbr = vbNull Then MsgBox "ESB_OtlDisAssociateAttributeDimension() failed.": Exit Sub
    sts = EsbOtlDisassociateAttributeDimension(hOutline, BaseMbr, AttrMbr)
    If sts <> 0 Then
        Out "EsbOtlDeleteDimension failed" & sts: Exit Sub
    Else
        Out "EsbOtlDeleteDimension succeeded: "; Exit Sub
    End If
    ' abstract sub to call EsbOtlVerifyOutline(), ESBOLTWriteOutline(),
    ' EsbOtlRestructure(),EsbUnlockObject() and
tuckinoutline
    If sts <> 0 Then Out "EsbOtlDisassociateAttributeDimension failed: "; Exit Sub
End Sub
```

See Also

- `EsbCheckAttributes`
- `EsbGetAssociatedAttributesInfo`
- `EsbGetAttributeInfo`
- `EsbGetAttributeSpecifications`
- `EsbOtlAssociateAttributeDimension`
- `EsbOtlAssociateAttributeMember`
EsbOtlDisassociateAttributeMember

Disassociates an attribute member from a base member.

Syntax

```vba
EsbOtlDisassociateAttributeMember (hOutline, BaseMember, AttributeMember)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>Handle to the outline</td>
</tr>
<tr>
<td>BaseMember</td>
<td>Handle to the base member</td>
</tr>
<tr>
<td>AttributeMember</td>
<td>Handle to the attribute member</td>
</tr>
</tbody>
</table>

Notes

When you disassociate an attribute dimension from a base dimension, you disassociate all members of the attribute dimension from members of the base dimension.

Return Value

Returns STS = 0 when successful. Otherwise, returns an error code.

Access

This function requires no special privileges.

Example

```vba
Sub ESB_OtlDisassociateAttributeMember()
    Dim BaseMbr As Long
    Dim AttributeMbr As Long
    Dim sts as long
    Dim hOutline as long
    hOutline = ESB_OtlOpenOutline
    If hOutline = vbNull Then Out "ESB_OtlOpenOutline() failed: " & sts:
    Exit Sub
    BaseMbr = ESB_OtlFindMember("Enter base member: ")
    If BaseMbr = vbNull Then
        Out "ESB_OtlGetMemberInfo() failed in ESB_OtlFindMember. " & sts:
        Exit Sub
    AttributeMbr = ESB_OtlFindMember("Enter attribute member: ")
    If AttributeMbr = vbNull Then
        Out "ESB_OtlGetMemberInfo() failed in ESB_OtlFindMember. "
```

1500 Visual Basic Outline API Functions
& sts: Exit Sub
    sts = EsbOtlDisassociateAttributeMember(hOutline, BaseMbr, AttributeMbr)
If sts = 0 Then Out "EsbOtlDisassociateAttributeMember failed " & sts: Exit Sub
    sts = EsbOtlDeleteMember(ghOutline, AttrMbr)
If sts <> 0 Then Out "EsbOtlDeleteMember failed" & sts: Exit Sub
' abstract sub to call EsbOtlVerifyOutline(), ESBOTLWriteOutline(),
EsbOtlRestructure(),EsbUnlockObject() and
' EsbOtlCloseOutline() as needed
End Sub

See Also
- EsbCheckAttributes
- EsbGetAssociatedAttributesInfo
- EsbGetAttributeInfo
- EsbGetAttributeSpecifications
- EsbOtlAssociateAttributeDimension
- EsbOtlAssociateAttributeMember
- EsbOtlDisassociateAttributeDimension
- EsbOtlFindAttributeMembers
- EsbOtlGetAssociatedAttributes
- EsbOtlGetAttributeInfo
- EsbOtlGetAttributeSpecifications
- EsbOtlQueryAttributes
- EsbOtlSetAttributeSpecifications

EsbOtlEnableDTSMember

Enables a new DTS member for the outline.

Syntax

EsbOtlEnableDTSMember (hOutline, pszDTSMember, usGen, bEnable)
ByVal hOutline As Long
ByVal pszDTSMember As String
ByVal usGen As Integer
ByVal bEnable As Integer

Parameter | Description
----------|----------------------
hOutline | Essbase outline handle returned from the EsbOtlOpenOutline function.
pszDTSMember | A string containing the name of the DTS member to enable.
usGen | The generation number at which to enable the DTS member.
bEnable | A flag. True means enable the member, false means disable the member.

Notes

This function also fills in the ESB_DTSMBRNAME_T structure passed to it.
Return Value

If successful the return value is zero. Otherwise, returns the status of the EsbOtlQueryMembers() call.

Example

Public Sub ESB_OtlEnableDTSMember()
    Dim DTSMember As String
    Dim GenNum As Integer
    Dim Enable As Integer

    DTSMember = "H-T-D"
    GenNum = 1
    Enable = ESB_TRUE

    sts = EsbOtlEnableDTSMember(hOutline, DTSMember, _
                                GenNum, Enable)
End Sub

See Also

- EsbOtlDeleteDTSMemberAlias
- EsbOtlGetEnabledDTSMembers
- EsbOtlGetDTSMemberAlias
- EsbOtlSetDTSMemberAlias

EsbOtlFindAlias

Finds a member with the specified alias name and returns a handle to the member.

Syntax

EsbOtlFindAlias (hOutline, pszAlias, pszAliasTable, phMember)
ByVal hOutline As Long
ByVal pszAlias As String
ByVal pszAliasTable As String
    phMember As Long

Parameter Description

hOutline Outline context handle.
pszAlias Alias name to find.
pszAliasTable Alias table to search in. Use "" to search all alias tables. Use "Default" to search the default alias table.
phMember Return variable for the member handle. ESB_NULL if the member is not found.

Notes

- Aliases used in alias combinations are also searched.
- If no member is found, phMember is set to "" and the call returns 0.
Return Value

Returns 0 if successful.

Example

Declare Function EsbOtlFindAlias Lib "ESBOTLN" (ByVal hOutline As Long, ByVal pszAlias As String, ByVal pszAliasTable As String, phMember As Long) As Long

Sub ESB_OtlFindAlias()
Dim sts As Long
Dim Object As ESB_OBJDEF_T
Dim hOutline As Long
Dim hMemberAlias As Long
Object.hCtx = hCtx
Object.Type = ESB_OBJTYPE_OUTLINE
Object.AppName = "Sample"
Object.DbName = "Basic"
Object.FileName = "Basic"
sts = EsbOtlOpenOutline(hCtx, Object, ESB_YES, ESB_YES, hOutline)
If sts = 0 Then
    sts = EsbOtlFindAlias(hOutline, "Root Beer", "", hMemberAlias)
End If
End Sub

See Also

- EsbOtlGetOutlineInfo
- EsbOtlGetMemberAlias

EsbOtlFindAttributeMembers

Returns all attribute members having the specified short name.

Syntax

EsbOtlFindAttributeMembers (hOutline, MemberName, DimensionName, Count, MemberArray)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>Handle to the outline</td>
</tr>
<tr>
<td>MemberName</td>
<td>Attribute short name</td>
</tr>
<tr>
<td>DimensionName</td>
<td>Attribute dimension name (optional)</td>
</tr>
<tr>
<td>Count</td>
<td>Number of members returned</td>
</tr>
<tr>
<td>MemberArray</td>
<td>Array of base member handles</td>
</tr>
</tbody>
</table>
Notes

- MemberName must be a short name.
- DimensionName is optional. You may enter NULL.

Return Value

Returns STS = 0 when successful. Otherwise, returns an error code.

Access

This function requires no special privileges.

Example

Sub ESB_OtlFindAttributeMembers()
    Dim MemberName As String
    Dim DimensionName As String
    Dim hMember() As Long
    Dim Count As Integer
    Dim MbrArr As Variant
    Dim MbrInfo As ESB_MBRINFO_T
    Dim index As Integer
    ghOutline = ESB_OtlOpenOutline
    If ghOutline = vbNull Then Out "ESB_OtlOpenOutline() failed: " & sts: Exit Sub
    ' expecting return of handle to "caffeinated_true"
    MemberName = "true"
    ' "null" by default - dimension name is optional
    DimensionName = ""
    sts = EsbOtlFindAttributeMembers(ghOutline, MemberName, DimensionName, Count, MbrArr)
    ' sts = EsbOtlFindAttributeMembers(ghOutline, MemberName, Count, MbrArr)
    If sts = 0 Then
        Out "EsbOtlFindAttributeMembers passed " & sts
        Out "Count is : " & Count
        For index = 0 To Count - 1
            sts = EsbOtlGetMemberInfo(ghOutline, MbrArr(index), MbrInfo)
            Out "Member Name : " & MbrInfo.szMember
            Next index
    Else
        Out "EsbOtlFindAttributeMembers failed " & sts
        Exit Sub
    End If
End Sub

See Also

- EsbCheckAttributes
- EsbGetAssociatedAttributesInfo
- EsbGetAttributeInfo
- EsbGetAttributeSpecifications
- EsbOtlAssociateAttributeDimension
- EsbOtlAssociateAttributeMember
- EsbOtlDisassociateAttributeDimension
- EsbOtlDisassociateAttributeMember
- EsbOtlGetAssociatedAttributes
EsbOtlFindMember

Finds a member with the specified name and returns a handle to the member.

Syntax

EsbOtlFindMember (hOutline, pszMember, phMember)

Parameter Description

hOutline  Outline context handle.

pszMember  Member name to find.

phMember  Return variable for the member handle. ESB_NULL if the member is not found.

Notes

- If the member being sought has shared members, only the handle to the actual member is returned. Once you have the handle, use EsbOtlGetNextSharedMember() to get shared member information.

- If no member is found, phMember is set to ESB_NULL and the call returns 0.

Return Value

Returns 0 if successful.

Example

Declare Function EsbOtlFindMember Lib "ESBOTLN" (ByVal hOutline As Long,
ByVal pszMember As String, phMember As Long) As Long
Sub ESB_OtlFindMember()
Dim sts As Long
Dim Object As ESB_OBJDEF_T
Dim hOutline As Long
Dim MbrInfo As ESB_MBRINFO_T
Dim hMemberProfit As Long
Dim hNewMember As Long
Object.hCtx = hCtx
Object.Type = ESB_OBJTYPE_OUTLINE
Object.AppName = "Sample"
Object.DbName = "Basic"
Object.FileName = "Basic"
sts = EsbOtlOpenOutline(hCtx, Object,
ESB_YES, ESB_YES, hOutline)
If sts = 0 Then
sts = EsbOtlFindMember(hOutline, "Profit", hMemberProfit)
End If
End Sub

See Also

- EsbOtlMoveMember
- EsbOtlRenameMember
- EsbOtlAddMember
- EsbOtlDeleteMember
- EsbOtlGetNextSharedMember

EsbOtlFreeMember

Frees any member returned from EsbOtlQueryMembers when EsbGetNextItem is called.

Syntax

EsbOtlFreeMember (hOutline, hMember)
ByVal hOutline As Long
ByVal hMember As Long

Parameter Description

hOutline    Essbase outline handle, returned from EsbOtlOpenOutlineQuery().

hMember    The member handle defining the member to free.

Notes

The results from EsbOtlQueryMembers() returns one member at a time via the
EsbGetNextItem() call. When each of these items is done being used, the programmer should
call EsbOtlFreeMember().

Return Value

Returns zero if successful.

Example

Declare Function EsbOtlFreeMember Lib "ESBOTLN"
(ByVal hOutline As Long, ByVal hMember As Long) As Long
Declare Function EsbOtlQueryMembers Lib "ESBOTLN"
(ByVal hOutline As Long, ByVal hMember As Long,
pPredicate As ESB_PREDICATE_T, pCounts As ESB_MBRCOUNTS_T) As Long

Sub ESB_OtlQueryMembers()
    Dim sts As Long
    Dim hOutline As Long
    Dim hMember As Long
    Dim ihMember As Long
    Dim Object As ESB_OBJDEF_T
    Dim MbrInfo As ESB_MBRINFO_T
    Dim Predicate As ESB_PREDICATE_T
    Dim Counts As ESB_MBRCOUNTS_T

1506 Visual Basic Outline API Functions
Dim Access As Integer
Dim AppName As String
Dim DbName As String

AppName = "Sample"
DbName = "Basic"
sts = EsbOtlOpenOutlineQuery(hCtx, Object, hOutline)
If sts = 0 Then
    sts = EsbOtlOpenOutlineQuery(hCtx, Object, hOutline)
Predicate.ulQuery = ESB_CHILDREN
Predicate.pszDimension = "Year"
Counts.ulStart = 0
Counts.ulMaxCount = 10
If sts = 0 Then
    sts = EsbOtlQueryMembers(hOutline, hMember, Predicate, Counts)
    If sts = 0 And Counts.ulReturnCount <> 0 Then
        For n% = 1 To Counts.ulReturnCount
            if sts = 0 And ihMember <> 0 Then
                sts = EsbOtlFreeMember(hOutline, ihMember)
        Next
    End If
    Next
End If
End If
End Sub

See Also

- EsbOtlOpenOutlineQuery
- EsbOtlQueryMembers
- EsbOtlQueryMembersByName

**EsbOtlGenerateCurrencyOutline**

Generates a currency outline based on the existing outline.

**Syntax**

```vba
EsbOtlGenerateCurrencyOutline (hOutline, phCurOutline)
```

**Parameter**

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
</tr>
<tr>
<td>phCurOutline</td>
</tr>
</tbody>
</table>

**Notes**

- There must be a Time, Accounts, and Country dimension in the source outline.
- The Time dimension and all descendants are copied directly from the source outline to a Time dimension in the new outline.
A dimension named CurCategory (Dense, Category = Accounts) is created in the new outline. All currency categories in the source Accounts dimension become children of the CurCategory dimension in the new outline.

A dimension named CurName (Dense, Category = Country) is created in the new outline. All currency names from the source Country dimension become children of the CurName dimension in the new outline.

A dimension named CurType (Sparse, Category = Type) is created with no children in the new outline.

The currency outline must be saved by calling ESBOTLNWriteOutline() followed by EsbOtlRestructure() and closed by calling EsbOtlCloseOutline().

The new outline has the following attributes:
- Auto-configure is set to ESB_TRUE
- Case-sensitivity is set to be the same as the original outline

**Return Value**

Returns 0 if successful; otherwise one of the following:

- OTLAPI_ERR_ALREADYCURRENCY
- OTLAPI_CUR_NOACCOUNTS
- OTLAPI_CUR_NOTIME
- OTLAPI_CUR_NOCOUNTRY

**Example**

Declare Function EsbOtlGenerateCurrencyOutline Lib "ESBOTLN" (ByVal hOutline As Long, phCurOutline As Long) As Long
Sub ESB_OtlGenerateCurrencyOutline()
Dim sts As Long
Dim Object As ESB_OBJDEF_T
Dim hOutline As Long
Dim hCurOutline As Long
Object.hCtx = hCtx
Object.Type = ESB_OBJTYPE_OUTLINE
Object.AppName = "Sample"
Object.DbName = "Interntl"
Object.FileName = " Interntl ">
sts = EsbOtlOpenOutline(hCtx, Object, ESB_YES, ESB_YES, hOutline)
If sts = 0 Then
    sts = EsbOtlGenerateCurrencyOutline(hOutline, hCurOutline)
End If
End Sub

**See Also**

- EsbOtlOpenOutline
- EsbOtlWriteOutline
- EsbOtlRestructure
**EsbOtlGetAliasTableLanguages**

Returns the number of the number of language codes associated with the specified alias table, and generates a list of alias table strings accessible through `EsbGetNextItem()`.

**Syntax**

```vba
EsbOtlGetAliasTableLanguages Lib "esbotln" (hOutline, pszAliasTable, pItems)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>Handle to the outline.</td>
</tr>
<tr>
<td>pszAliasTable</td>
<td>Name of the alias table for which to get an associated language code.</td>
</tr>
<tr>
<td>pItems</td>
<td>Address of a variable in which to return the number of language codes associated with the alias table.</td>
</tr>
</tbody>
</table>

**Return Value**

- If successful, returns the number of alias table languages in `pItems`, and generates a list of alias table strings accessible through `EsbGetNextItem()`.
- If unsuccessful, returns the error OTLAPI_BAD_ALIASSTABLE (invalid alias table).

**Access**

This function does not require special privileges.

**Example**

```vba
Declare Function EsbOtlGetAliasTableLanguages Lib "esbotln" (ByVal hOutline As Long, ByVal pszAliasTable As String, pItems As Long) As Long
Declare Function EsbOtlSetAliasTableLanguage Lib "esbotln" (ByVal hOutline As Long, ByVal pszAliasTable As String, ByVal pszLanguageCode As String) As Long
Declare Function EsbOtlClearAliasTableLanguages Lib "esbotln" (ByVal hOutline As Long, ByVal pszAliasTable As String) As Long

Sub ESB_Sub ()
    Dim sts As Long
    Dim Object As ESB_OBJDEF_T
    Dim hOutline As Long
    Dim Items As Long
    Dim AliasLang As String * ESB_ALIASNAMELEN

    Object.hCtx = hCtx
    Object.Type = ESB_OBJTYPE_OUTLINE
    Object.AppName = "Sample"
    Object.DbName = "Basic"
    Object.FileName = "Basic"
    sts = EsbOtlOpenOutline(hCtx, Object, ESB_YES, ESB_YES, hOutline)
    If sts = 0 Then
        sts = EsbOtlCreateAliasTable(hOutline, "French Alias Table")
    End If
```

Visual Basic Outline API Function Reference 1509
If $\text{sts} = 0$ Then
    $\text{sts} = \text{EsbOtlSetAliasTableLanguage}(\text{hOutline},$ 
    "French Alias Table", "fr")
End If
If $\text{sts} = 0$ Then
    $\text{sts} = \text{EsbOtlSetAliasTableLanguage}(\text{hOutline},$ 
    "French Alias Table", "fr-CA")
End If
If $\text{sts} = 0$ Then
    $\text{sts} = \text{EsbOtlGetAliasTableLanguages}(\text{hOutline},$ 
    "French Alias Table", Items)
    If $\text{sts} = 0$ Then
        For $N = 1$ To Items
            $\text{sts} = \text{EsbGetNextItem}(\text{hCtx}, \text{ESB_ALIASLANG_TYPE}, \text{ByVal AliasLang})$
        Next
    End If
End If
If $\text{sts} = 0$ Then
    $\text{sts} = \text{EsbOtlClearAliasTableLanguages}(\text{hOutline},$ 
    "French Alias Table")
End If
End Sub

See Also

- `EsbOtlClearAliasTableLanguages`
- `EsbOtlSetAliasTableLanguage`

**EsbOtlGetAssociatedAttributes**

Returns all attribute members that are associated with a base member or dimension.

**Syntax**

```
EsbOtlGetAssociatedAttributes (hOutline, Member, Count, MemberArray)
```

**Parameter** | **Description**
--- | ---
$h\text{Outline}$ | Handle to the outline
$\text{Member}$ | Handle to the base member or base dimension
$\text{Count}$ | Number of attribute members returned
$\text{MemberArray}$ | Array of attribute member handles

**Return Value**

Returns $\text{STS} = 0$ when successful. Otherwise, returns an error code.
Access

This function requires no special privileges.

Example

Sub ESB_OtlGetAssociatedAttributes()
    Dim hMember As Long
    Dim Count As Integer               '*** Return of number of attributes
    Dim MbrArr As Variant              '*** Returns member array in this
    Dim MbrInfo As ESB_MBRINFO_T       '*** Returned MbrInfo structure
    Dim index As Integer
    eraser
    hMember = ESB_OtlFindMember("Enter target member: ")
    If hMember = vbNull Then Out "ESB_OtlGetAssociatedAttributes() failed.": Exit Sub
    sts = EsbOtlGetAssociatedAttributes(ghOutline, hMember, Count, MbrArr)
    If sts = 0 Then
        Out "Count is : " & Count
        For index = 0 To (Count - 1)
            sts = EsbOtlGetMemberInfo(ghOutline, MbrArr(index), MbrInfo)
            Out "Member Name : " & MbrInfo.szMember
        Next index
    Else
        Out "EsbOtlGetAttributeInfo failed" & sts: Exit Sub
    End If
End Sub

See Also

- EsbCheckAttributes
- EsbGetAssociatedAttributesInfo
- EsbGetAttributeInfo
- EsbGetAttributeSpecifications
- EsbOtlAssociateAttributeDimension
- EsbOtlAssociateAttributeMember
- EsbOtlDisassociateAttributeDimension
- EsbOtlDisassociateAttributeMember
- EsbOtlFindAttributeMembers
- EsbOtlGetAttributeInfo
- EsbOtlGetAttributeSpecifications
- EsbOtlQueryAttributes
- EsbOtlSetAttributeSpecifications

**EsbOtlGetAttributeInfo**

Returns attribute information for a given attribute member or dimension.

Syntax

**EsbOtlGetAttributeInfo** (hOutline, Member, AttrInfo)

ByVal hOutline As Long
ByVal Member As Long
    AttrInfo As ESB_ATTRIBUTEINFO_T
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>Handle to the outline</td>
</tr>
<tr>
<td>Member</td>
<td>Handle to the attribute member or dimension</td>
</tr>
<tr>
<td>AttrInfo</td>
<td>Attribute information</td>
</tr>
</tbody>
</table>

**Notes**

This function is similar to `EsbGetAttributeInfo()`.

**Return Value**

Returns STS = 0 when successful. Otherwise, returns an error code.

**Access**

This function requires no special privileges.

**Example**

```visual_basic
Sub ESB_OtlGetAttributeInfo()
    ' NOTE: 'Out' is a sub to print the output within quotes to a listbox or text box
    Dim sts As Long
    Dim OutAttrInfo As ESB_ATTRIBUTEINFO_T
    Dim MbrName As String
    Dim hCtx As Long
    MbrName = InputBox("Enter Member Name")
    sts = EsbGetAttributeInfo(hCtx, MbrName, OutAttrInfo)
    If sts = 0 Then
        Select Case VarType(OutAttrInfo.Attribute)
            Case vbDouble
                Out "Data Type : Numeric(Double)"
                Out "Data Value : " & OutAttrInfo.Attribute
                Out ""
            Case vbBoolean
                Out "Data Type : Boolean"
                Out "Data Value : " & OutAttrInfo.Attribute
                Out ""
            Case vbDate
                Out "Data Type : Date"
                Out "Data Value : " & OutAttrInfo.DimName
                Out ""
            Case vbString
                Out "Data Type : String"
                Out "Data Value : " & OutAttrInfo.Attribute
                Out ""
        End Select
    Else
        Out "ESB_OtlGetAttributeInfo failed" & sts
        Exit Sub
    End If
End Sub
```
See Also

- EsbCheckAttributes
- EsbGetAssociatedAttributesInfo
- EsbGetAttributeInfo
- EsbGetAttributeSpecifications
- EsbOtlAssociateAttributeDimension
- EsbOtlAssociateAttributeMember
- EsbOtlDisassociateAttributeDimension
- EsbOtlDisassociateAttributeMember
- EsbOtlFindAttributeMembers
- EsbOtlGetAssociatedAttributes
- EsbOtlGetAttributeSpecifications
- EsbOtlQueryAttributes
- EsbOtlSetAttributeSpecifications

**EsbOtlGetAttributeSpecifications**

Retrieves attribute specifications for the outline.

**Syntax**

```
EsbOtlGetAttributeSpecifications (hOutline, AttrSpecs)
```

ByVal hOutline As Long
AttrSpecs As ESB_ATTRSPECS_T

**Parameter Description**

- hOutline: Handle to the outline
- AttrSpecs: Attribute specifications

**Notes**

- This function is similar to `EsbGetAttributeSpecifications()`, except that it returns information from the opened outline.
- Set attribute specifications for the outline using `EsbOtlSetAttributeSpecifications()`.
- Attribute specifications are used to do the following:
  - Generate a long name
  - Indicate the format of a datetime attribute
  - Indicate a numeric attribute's bucketing type
  - Provide the name of the attribute calculations dimension and the names for the values used with it

**Return Value**

Returns STS = 0 when successful. Otherwise, returns an error code.
Access

This function requires no special privileges.

Example

Sub ESB_OtlGetAttributeSpecifications()
    Dim OutAttrSpecs As ESB_ATTRSPECS_T
    Dim test As String
    Dim sts as long
    hOutline = ESB_OtlOpenOutline
    If hOutline = vbNull Then Out "ESB_OtlOpenOutline() failed: " & sts: Exit Sub
    sts = ESB_OtlGetAttributeSpecifications(hOutline, OutAttrSpecs)
    If sts <> 0 Then Out "ESB_OtlGetAttributeSpecifications failed" & sts: Exit Sub
    Out "ESB_OtlGetAttributeSpecifications passed: " & sts
    Out "DefaultTrueString : " & OutAttrSpecs.DefaultTrueString
    Out "DefaultFalseString : " & OutAttrSpecs.DefaultFalseString
    Out "DefaultAttrCalcDimName : " & OutAttrSpecs.DefaultAttrCalcDimName
    Out "DefaultSumMbrName : " & OutAttrSpecs.DefaultSumMbrName
    Out "DefaultCountMbrName : " & OutAttrSpecs.DefaultCountMbrName
    Out "DefaultAverageMbrName : " & OutAttrSpecs.DefaultAverageMbrName
    Out "DefaultMinMbrName : " & OutAttrSpecs.DefaultMinMbrName
    Out "DefaultMaxMbrName : " & OutAttrSpecs.DefaultMaxMbrName
    test = OutAttrSpecs.GenNameBy
    Select Case test
        Case ESB_GENNAMEBY_PREFIX
            Out "GenNameBy : ESB_GENNAMEBY_PREFIX"
        Case ESB_GENNAMEBY_SUFFIX
            Out "GenNameBy : ESB_GENNAMEBY_SUFFIX"
        Case Else
            Out "GenNameBy : invalid"
    End Select
    test = OutAttrSpecs.UseNameOf
    Select Case test
        Case ESB_USENAMEOF_NONE
            Out "UseNameOf : ESB_USENAMEOF_NONE"
        Case ESB_USENAMEOF_PARENT
            Out "UseNameOf : ESB_USENAMEOF_PARENT"
        Case ESB_USENAMEOF_GRANDPARENTANDPARENT
            Out "UseNameOf : ESB_USENAMEOF_GRANDPARENTANDPARENT"
        Case ESB_USENAMEOF_ALLANCESTORS
            Out "UseNameOf : ESB_USENAMEOF_ALLANCESTORS"
        Case ESB_USENAMEOF_DIMENSION
            Out "UseNameOf : ESB_USENAMEOF_DIMENSION"
        Case Else
            Out "UseNameOf : invalid"
    End Select
    test = OutAttrSpecs.Delimiter
    Select Case test
        Case ESB_DELIMITER_UNDERSCORE
            Out "Delimiter : ESB_DELIMITER_UNDERSCORE"
        Case ESB_DELIMITER_PIPE
            Out "Delimiter : ESB_DELIMITER_PIPE"
        Case ESB_DELIMITER_CARET
            Out "Delimiter : ESB_DELIMITER_CARET"
        Case Else
            Out "Delimiter : invalid"
    End Select
End Sub
test = OutAttrSpecs.DateFormat
Select Case test
    Case ESB_DATEFORMAT_MMDDYYYY
        Out "DateFormat : ESB_DATEFORMAT_MMDDYYYY"
    Case ESB_DATEFORMAT_DDMMYYYY
        Out "DateFormat : ESB_DATEFORMAT_DDMMYYYY"
    Case Else
        Out "Delimiter : invalid"
End Select

test = OutAttrSpecs.BucketingType
Select Case test
    Case ESB_UPPERBOUNDINCLUSIVE
        Out "BucketingType : ESB_UPPERBOUNDINCLUSIVE"
    Case ESB_LOWERBOUNDINCLUSIVE
        Out "BucketingType : ESB_LOWERBOUNDINCLUSIVE"
    Case ESB_UPPERBOUNDNONINCLUSIVE
        Out "BucketingType : ESB_UPPERBOUNDNONINCLUSIVE"
    Case ESB_LOWERBOUNDNONINCLUSIVE
        Out "BucketingType : ESB_LOWERBOUNDNONINCLUSIVE"
    Case Else
        Out "BucketingType : invalid"
End Select

End Sub

See Also

- EsbCheckAttributes
- EsbGetAssociatedAttributesInfo
- EsbGetAttributeInfo
- EsbGetAttributeSpecifications
- EsbOtlAssociateAttributeDimension
- EsbOtlAssociateAttributeMember
- EsbOtlDisassociateAttributeDimension
- EsbOtlDisassociateAttributeMember
- EsbOtlFindAttributeMembers
- EsbOtlGetAssociatedAttributes
- EsbOtlGetAttributeInfo
- EsbOtlQueryAttributes
- EsbOtlSetAttributeSpecifications

**EsbOtlGetChild**

Returns the child of a member.

**Syntax**

```plaintext
EsbOtlGetChild (hOutline, hMember, phMember)
ByVal hOutline As Long
ByVal hMember As Long
    phMember As Long
```
### Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>Outline context handle.</td>
</tr>
<tr>
<td>hMember</td>
<td>Handle of member to retrieve the child of.</td>
</tr>
<tr>
<td>phMember</td>
<td>Return variable for the handle of the child of the hMember parameter.</td>
</tr>
</tbody>
</table>

### Notes
- If there is no child, *phMember* is set to ESB_NULL and the call returns 0.

### Return Value
- Returns 0 if successful.

### Example
```visualbasic
declare function EsbOtlGetChild Lib "ESBOTLN" (byval hOutline as long, byval hMember as long, phMember as long) as long
sub ESB_OtlGetChild()
    dim sts as long
    dim object as ESB_OBJDEF_T
    dim hOutline as long
    dim hMemberParent as long
    dim hMemberChild as long
    object.hctx = hCtx
    object.type = ESB_OBJTYPE_OUTLINE
    object.appName = "Sample"
    object.dbName = "Basic"
    object.fileName = "Basic"
    sts = EsbOtlOpenOutline(hCtx, object, ESB_YES, ESB_YES, hOutline)
    if sts = 0 then
        sts = EsbOtlFindMember(hOutline, "Year", hMemberParent)
    end if
    if sts = 0 and hMemberParent <> 0 then
        sts = EsbOtlGetChild(hOutline, hMemberParent, hMemberChild)
    end if
end sub
```

### See Also
- EsbOtlGetParent
- EsbOtlGetNextSibling
- EsbOtlGetPrevSibling
- EsbOtlGetFirstMember

### EsbOtlGetDimensionUserAttributes
- Returns the user defined attributes used in the specified dimension.

---

1516 Visual Basic Outline API Functions
Syntx

EsbOtlGetDimensionUserAttributes (hOutline, pPredicate, pCounts)
ByVal hOutline    As Long
    pPredicate As ESB_PREDICATE_T
    pCounts   As ESB_MBRCOUNTS_T

Parameter Description

hOutline   Essbase outline handle. This must have been returned from EsbOtlOpenOutlineQuery().
pPredicate Structure defining the query. The fields of this structure are used as follows:
    ● ulQuery—Value defining the operation to perform. The only valid value is ESB_DIMUSERATTRIBUTES.
    ● szDimension—Dimension to limit the scope of the query. Specify a valid dimension name.
pCounts Structure defining information about counts It contains the following fields:
    ● ulStart—Starting number to return
    ● ulMaxCount—Maximum number of member names to return.
    ● ulTotalCount—Total number of members that are defined in the results of the query.
    ● pulReturnCount—Number of member names returned in this query.

Notes

This function is used only to get the user's defined attributes on a specific dimension. Therefore, the only valid value for Predicate is ESB_DIMUSERATTRIBUTES_T.

Return Value

The return value is zero if the function was successful.

Example

Declare Function EsbOtlGetDimensionUserAttributes Lib "ESBOTLN"
    (ByVal hOutline As Long, pPredicate As ESB_PREDICATE_T,
    pCounts As ESB_MBRCOUNTS_T) As Long

Sub ESB_OtlQueryMembers()
    Dim sts As Long
    Dim hOutline As Long
    Dim AttrName As String * ESB_MBRNAMELEN
    Dim Predicate As ESB.Predicate_T
    Dim Counts As ESB_MBRCOUNTS_T
    Dim Access As Integer
    Dim AppName As String
    Dim DbName As String

    AppName = "Sample"
    DbName = "Basic"
    sts = EsbOtlOpenOutlineQuery(hCtx, Object, hOutline)
    If sts = 0 Then
        sts = EsbOtlOpenOutlineQuery(hCtx, Object, hOutline)
    Predicate.ulQuery = ESB_DIMUSERATTRIBUTES_T
    Predicate.szDimension = "Product"
    Counts.ulStart = 0
    Counts.ulMaxCount = 10
If sts = 0 Then
    sts = EsbOtlGetDimensionUserAttributes(hOutline, Predicate, Counts)
If sts = 0 And Counts.ulReturnCount <> 0 Then
    For n% = 1 To Counts.ulReturnCount
        sts = EsbGetNextItem(hCtx, ESB_MBRNAME_TYPE, ByVal AttrName)
        MsgBox AttrName
    Next
End If
End If
End Sub

See Also
- EsbGetNextItem
- EsbOtlOpenOutlineQuery
- EsbOtlQueryMembers
- EsbOtlQueryMembersByName

**EsbOtlGetDTSMemberAlias**

Gets an alias name for a DTS member.

**Syntax**

```vbnet
EsbOtlGetDTSMemberAlias (hOutline, pszDTSMember, pszAliasTable, ppszAlias)
```

**Parameter Description**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>Essbase outline handle returned from the EsbOtlOpenOutlineQuery call.</td>
</tr>
<tr>
<td>pszDTSMember</td>
<td>Name of the DTS member which provides the alias.</td>
</tr>
<tr>
<td>pszAliasTable</td>
<td>Name of the alias table which provides the alias. If NULL, the default alias table is used.</td>
</tr>
<tr>
<td>ppszAlias</td>
<td>Pointer to a pointer to a C string containing the alias name for the DTS member.</td>
</tr>
</tbody>
</table>

**Notes**

The fixed length of ESB_ALIASNAMELEN sets the string length for a variable alias.

**Return Value**

If successful the return value is zero. Otherwise, one of the following is returned:

- OTLAPI_ERR_DTSMBRNOREDEFINED
- OTLAPI_BAD_ALIASTABLE

**Example**

```vbnet
Public Sub ESB_OtlGetDTSMemberAlias()
    Dim DTSMember As String * ESB_MBRNAMELEN
```
Dim AliasTable As String * ESB_ALIASNAMELEN
Dim Alias As String * ESB_ALIASNAMELEN

DTSMember = "H-T-D"
AliasTable = "Default"

sts = EsbOtlGetDTSMemberAlias(hOutline, DTSMember, _
    AliasTable, Alias)
MsgBox Alias

End Sub

See Also
- EsbOtlDeleteDTSMemberAlias
- EsbOtlEnableDTSMember
- EsbOtlGetEnabledDTSMembers
- EsbOtlSetDTSMemberAlias

EsbOtlGetEnabledDTSMembers

Gets the defined DTS members for the outline.

Syntax

EsbOtlGetEnabledDTSMembers (hOutline, pusCount)
ByVal hOutline As Long
    pusCount As Integer

Parameter Description

hOutline Essbase outline handle returned from the EsbOtlOpenOutlineQuery() call.
pusCount The number of defined DTS members.

Notes

Upon successful call, a call to EsbGetNextItem must be called for each enabled DTS member determined by the value returned to Count.

Return Value

If successful the return value is zero. Otherwise it returns the status of the EsbOtlQueryMembers() call.

Example

Public Sub ESB_OtlGetEnabledDTSMembers()
    Dim Count As Integer
    Dim DTSMbr As String * ESB_MBRNAMELEN
    Dim i As Integer

    sts = EsbOtlGetEnabledDTSMembers(hOutline, Count)
    If sts = 0 Then
        For i = 1 To Count
            sts = EsbGetNextItem(hCtx, ESB_DTS_TYPE, ByVal DTSMbr)
    End If
End Sub
MsgBox "DTSMbr"
Next i
End If

End Sub

See Also
- EsbOtlDeleteDTSMemberAlias
- EsbOtlEnableDTSMember
- EsbOtlGetDTSMemberAlias
- EsbOtlSetDTSMemberAlias

**EsbOtlGetFirstMember**

Returns a member handle to the first member in the outline. The first member is the first dimension defined in the outline.

**Syntax**

```vbnet
EsbOtlGetFirstMember (hOutline, phMember)
```

ByVal hOutline As Long
 phMember As Long

**Parameter**  **Description**

hOutline  Outline context handle.

phMember  Variable for the handle of the first member in the outline. This parameter is passed to subsequent calls for traversing the outline.

**Return Value**

Returns 0 if successful.

**Example**

Declare Function EsbOtlGetFirstMember Lib "ESBOTLN" (ByVal hOutline As Long, phMember As Long) As Long

Sub ESB_OtlGetFirstMember()
 Dim sts As Long
 Dim Object As ESB_OBJDEF_T
 Dim hOutline As Long
 Dim hMemberFirst As Long
 Object.hCtx = hCtx
 Object.Type = ESB_OBJTYPE_OUTLINE
 ObjectAppName = "Sample"
 Object.DbName = "Basic"
 Object.FileName = "Basic"
 sts = EsbOtlOpenOutline(hCtx, Object, ESB_YES, ESB_YES, hOutline)
 If sts = 0 Then
  sts = EsbOtlGetFirstMember (hOutline, hMemberFirst)
EsbOtlGetGenName

Retrieves the name for a specific generation within a dimension. Generation names are explicitly added to the outline with `EsbOtlSetGenName`.

Syntax

```vba
EsbOtlGetGenName (hOutline, pszDimension, usGen, pszName)
```

Parameter Description

- `hOutline`: Outline context handle.
- `pszDimension`: Name of dimension that contains the generation name.
- `usGen`: Number of generation for which to get a name. The dimension is generation 1.
- `pszName`: Buffer for return of generation name, allocated by the caller. The buffer must be large enough to hold a valid member name (ESB_MBRNAMELEN).

Notes

- The generation name follows the same rules as a member name and must be unique across the entire member name space. It cannot duplicate any other generation, level, member name, or alias. Attempting to add a duplicate name generates an error.
- Generation names are not automatically assigned. For this function to return the name, a name must have been assigned. The name can be assigned with `EsbOtlSetGenName`.

Return Value

Returns 0 if successful; otherwise:

- OTLAPI_NO_GENLEVELNAME
- OTLAPI_ERR_NOTADIM

Example

```vba
Declare Function EsbOtlGetGenName Lib "ESBOTLN" (ByVal hOutline As Long, ByVal pszDimension As String, ByVal usGen As Integer, ByVal pszName As String) As Long
```
Sub ESB_OtlGetGenName()
Dim sts As Long
Dim Object As ESB_OBJDEF_T
Dim hOutline As Long
Dim Dimension As String
Dim GenNum As Integer
Dim GenName As String * ESB_MBRNAMELEN
Object.hCtx = hCtx
Object.Type = ESB_OBJTYPE_OUTLINE
Object.AppName = "Sample"
Object.DbName = "Basic"
Object.FileName = "Basic"
sts = EsbOtlOpenOutline(hCtx, Object, ESB_YES, ESB_YES, hOutline)
'**********************************************
'************** Get Gen Name ****************
'**********************************************
Dimension = "Year"
GenNum = 2
If Not sts Then
  sts = EsbOtlGetGenName(hOutline, Dimension, GenNum, GenName)
End If
End Sub

See Also
- EsbOtlGetGenNames
- EsbOtlDeleteGenName
- EsbOtlSetGenName

EsbOtlGetGenNames
Retrieves all generation names specified for a particular dimension. Generation names are explicitly added to the outline with EsbOtlSetGenName.

Syntax
EsbOtlGetGenNames (hOutline, pszDimension, ulOptions, pulCount)
ByVal hOutline As Long
ByVal pszDimension As String
ByVal ulOptions As Long
  pulCount As Long

ParameterDescription
hOutline Esbase outline handle.
pszDimension The dimension to retrieve generation names for.
### Parameter Description

**ulOptions**

This can be one of the following values:

- ESB_GENLEV_ALL - return default and actual generation names
- ESB_GENLEV_ACTUAL - return only generation names that are actually defined
- ESB_GENLEV_DEFAULT - return all default generation names. This includes the default names for generations that have an actual name.
- ESB_GENLEV_NOACTUAL - return default generation names. This includes only the generations that don’t have an actual generation name

**pulCount**

Return of the number of elements in the pNameArray. It is the number of generation names for the specified member.

**pNameArray**

An array of generation name structures for the specified dimension.

### Notes

- The caller should free the pNameArray structure after use by calling `EsbFree()`.
- The programmer should call `EsbGetNextItem()` once for each generation name structure returned.
- This call will work for both `EsbOtlOpenOutline()` and `EsbOtlOpenOutlineQuery()`. The information will exist locally for both, since it is returned from the server during the `EsbOtlOpenOutlineQuery()` call.

### Return Value

The return value is zero if the function was successful.

### Example

```visualbasic
Declare Function EsbOtlGetGenNames Lib "ESBOTLN"
(ByVal hOutline As Long, ByVal pszDimension As String, ByVal ulOptions As Long, pulCount As Long) As Long

Sub ESB_OtlGetNames()
    Dim sts As Long
    Dim hOutline As Long
    Dim Object As ESB_OBJDEF_T
    Dim Dimension As String
    Dim GenOpt As Long
    Dim Count As Long
    Dim pGenName As ESB_GENLEVELNAME_T
    Dim Access As Integer
    Dim AppName As String
    Dim DbName As String
    AppName = "Sample"
    DbName = "Basic"
    sts = EsbSetActive(hCtx, AppName, DbName, Access)
    If sts=0 Then
        sts = EsbOtlOpenOutlineQuery(hCtx, Object, hOutline)
        '************** Get Gen Names ***************
        Dimension = "Year"
        GenOpt = ESB_GENLEV_DEFAULT
```

Visual Basic Outline API Function Reference 1523
If sts = 0 Then
sts = EsbOtlGetGenNames(hOutline, Dimension, GenOpt, Count)
If sts = 0 And Count <> 0 Then
For n% = 1 To Count
sts = EsbGetNextItem(hCtx, ESB_GENLEVELNAME_TYPE, pGenName)
Next
End If
End If
End If
End Sub

See Also

- EsbGetNextItem
- EsbOtlGetGenName
- EsbOtlGetLevelName
- EsbOtlGetLevelNames
- EsbOtlOpenOutline
- EsbOtlOpenOutlineQuery

**EsbOtlGetLevelName**

Gets the name for a specific level within a dimension. Level names are explicitly added to the outline with `EsbOtlSetLevelName`.

**Syntax**

```vbnet
EsbOtlGetLevelName (hOutline, pszDimension, usLevel, pszName)
```

- **hOutline** As Long
- **pszDimension** As String
- **usLevel** As Integer
- **pszName** As String

**Parameter** | **Description**
---|---
hOutline | Outline context handle.
pszDimension | Name of dimension that contains the generation.
usLevel | Number of level number for which to get a name. Leaf members are level 0.
pszName | Buffer for return of the level of the specified dimension, allocated by the caller. The buffer must be large enough to hold a valid member name (ESB_MBRNAMELEN).

**Notes**

- The generation name follows the same rules as a member name and must be unique across the entire member name space. It cannot duplicate any other generation, level, member name, or alias. Attempting to add a duplicate name generates an error.
- Level names are not automatically assigned. For this function to return the name, a name must have been assigned. The name can be assigned with `EsbOtlSetLevelName`. 

1524 Visual Basic Outline API Functions
**Return Value**

The return value is zero if the function was successful. Otherwise, the command returns either of the following:

- OTLAPI_NO_GENLEVELNAME
- OTLAPI_ERR_NOTADIM

**Example**

Declare Function EsbOtlGetLevelName Lib "ESBOTLN" (ByVal hOutline As Long, ByVal pszDimension As String, ByVal usLevel As Integer, ByVal pszName As String) As Long

Sub ESB_OtlGetLevelName()
  Dim sts As Long
  Dim Object As ESB_OBJDEF_T
  Dim hOutline As Long
  Dim Dimension As String
  Dim LevelNum As Integer
  Dim LevelName As String * ESB_MBRNAMELEN
  Object.hCtx = hCtx
  Object.Type = ESB_OBJTYPE_OUTLINE
  Object.AppName = "Sample"
  Object.DbName = "Basic"
  Object.FileName = "Basic"
  sts = EsbOtlOpenOutline(hCtx, Object, ESB_YES, ESB_YES, hOutline)
  '*****************************************
  '********** Get Level Name **************
  '*****************************************
  Dimension = "Year"
  LevelNum = 2
  If Not sts Then
    sts = EsbOtlGetLevelName(hOutline, Dimension, LevelNum, LevelName)
  End If
End Sub

**See Also**

- EsbOtlGetLevelNames
- EsbOtlDeleteLevelName
- EsbOtlSetLevelName

**EsbOtlGetLevelNames**

Retrieves all level names specified for a particular dimension. Level names are explicitly added to the outline with **EsbOtlSetLevelName**.

**Syntax**

```plaintext```
EsbOtlGetLevelNames (hOutline, pszDimension, ulOptions, pulCount)
ByVal hOutline         As Long
ByVal pszDimension    As String
```
ByVal ulOptions As Long
pulCount As Long

Parameter    Description
hOutline     Esbbase outline handle.
pszDimension The dimension to retrieve level names for.
ulOptions    This can be one of the following values:
        • ESB_GENLEV_ALL—Return default and actual level names.
        • ESB_GENLEV_ACTUAL—Return only level names that are actually defined.
        • ESB_GENLEV_DEFAULT—Return all default level names. This includes the default names for
          levels that have an actual name.
        • ESB_GENLEV_NOACTUAL—Return default level names. This includes only the levels that don’t
          have an actual level name.
pulCount     Return of the number of elements in the pNameArray. It is the number of level names for the specified
                  member.
pulCount     An array of level name structures for the specified dimension.

Notes
        • The caller should free the pNameArray structure after use by calling EsbFree().
        • The programmer should call EsbGetNextItem() once for each level name structure returned.
        • This call will work for both EsbOtlOpenOutline() and EsbOtlOpenOutlineQuery(). The
          information exists locally for both, since it is returned from the server during the
          EsbOtlOpenOutlineQuery() call.

Return Value

The return value is zero if the function was successful.

Example

Declare Function EsbOtlGetLevelNames Lib "ESBOTLN"
(ByVal hOutline As Long, ByVal pszDimension As String, ByVal ulOptions
As Long, pulCount As Long) As Long
Sub ESB_OtlGetLevelNames()
    Dim sts As Long
    Dim hOutline As Long
    Dim Object As ESB_OBJDEF_T
    Dim Dimension As String
    Dim LevOpt As Long
    Dim Count As Long
    Dim pLevName As ESB_GENLEVELNAME_T
    Dim Access As Integer
    Dim AppName As String
    Dim DbName As String

    AppName = "Sample"
    DbName = "Basic"
    sts = EsbSetActive(hCtx, AppName, DbName, Access)
If sts = 0 Then
sts = EsbOtlOpenOutlineQuery(hCtx, Object, hOutline)
'************** Get Level Names ***************
Dimension = "Year"
LevOpt = ESB_GENLEV_DEFAULT
If sts = 0 Then
    sts = EsbOtlGetLevelNames(hOutline, Dimension,
        LevOpt, Count)
    If sts = 0 And pCount <> 0 Then
        For n = 1 To Count
            sts = EsbGetNextItem(hCtx, ESB_GENLEVELNAME_TYPE, pLevName)
            Next
        End If
    End If
End If
End Sub

See Also
- EsbOtlGetGenName
- EsbOtlGetGenNames
- EsbOtlGetLevelName
- EsbOtlOpenOutline
- EsbOtlOpenOutlineQuery

**EsbOtlGetMemberAlias**

Gets the default member alias for a specified member in a specified alias table.

**Syntax**

```
EsbOtlGetMemberAlias (hOutline, hMember, pszAliasTable, pszAlias)
ByVal hOutline As Long
ByVal hMember As Long
ByVal pszAliasTable As String
ByVal pszAlias As String
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>Outline context handle.</td>
</tr>
<tr>
<td>hMember</td>
<td>Handle of member to get the alias for.</td>
</tr>
<tr>
<td>pszAliasTable</td>
<td>Alias table to get the alias from. If this parameter is &quot;&quot;, the default alias table is used.</td>
</tr>
<tr>
<td>pszAlias</td>
<td>Buffer for the return of the alias. The buffer is allocated by the caller.</td>
</tr>
</tbody>
</table>

**Notes**

The `pszAlias` parameter should be a buffer allocated by the caller of at least ESB_MBRNAMELEN bytes.

**Return Value**

Returns 0 if successful; otherwise:
Example

Declare Function EsbOtlGetMemberAlias Lib "ESBOTLN" (ByVal hOutline As Long, ByVal hMember As Long, ByVal pszAliasTable As String, ByVal pszAlias As String) As Long

Sub ESB_OtlGetMemberAlias()
    Dim sts As Long
    Dim Object As ESB_OBJDEF_T
    Dim hOutline As Long
    Dim hMemberProfit As Long
    Dim szAlias As String * ESB_MBRNAMELEN
    Object.hCtx = hCtx
    Object.Type = ESB_OBJTYPE_OUTLINE
    Object.AppName = "Sample"
    Object.DbName = "Basic"
    Object.FileName = "Basic"
    sts = EsbOtlOpenOutline(hCtx, Object, ESB_YES, ESB_YES, hOutline)
    If sts = 0 Then
        sts = EsbOtlFindMember(hOutline, "Profit", hMemberProfit)
    End If
    If sts = 0 And hMemberProfit <> 0 Then
        sts = EsbOtlGetMemberAlias(hOutline, hMemberProfit, "Default", szAlias)
    End If
End Sub

See Also

- EsbOtlSetMemberAlias
- EsbOtlDeleteMemberAlias

EsbOtlGetMemberFormula

Gets the formula for the specified member.

Syntax

EsbOtlGetMemberFormula (hOutline, hMember, pszFormula, usBufSize)

ByVal hOutline As Long
ByVal hMember As Long
ByVal pszFormula As String
ByVal usBufSize As Integer

Parameter Description

hOutline Outline context handle.

hMember Member Handle.
Parameter | Description
--- | ---
pszFormula | Return variable for the member formula. The buffer is allocated by the caller, and the length is specified in the usBufSize parameter.
usBufSize | Size of the pszFormula buffer.

Return Value

Returns 0 if successful.

Example

Declare Function EsbOtlGetMemberFormula Lib "ESBOTLN" (ByVal hOutline As Long, ByVal hMember As Long, ByVal pszFormula As String, ByVal usBufSize As Integer) As Long

Sub ESB_OtlGetMemberFormula()
    Dim sts As Long
    Dim Object As ESB_OBJDEF_T
    Dim hOutline As Long
    Dim hMemberProfit As Long
    Dim szFormula As String * 100
    Object.hCtx = hCtx
    Object.Type = ESB_OBJTYPE_OUTLINE
    Object.AppName = "Sample"
    Object.DbName = "Basic"
    Object.FileName = "Basic"
    sts = EsbOtlOpenOutline(hCtx, Object, ESB_YES, ESB_YES, hOutline)
    If sts = 0 Then
        sts = EsbOtlFindMember(hOutline, "Profit", hMemberProfit)
    End If
    If sts = 0 And hMemberProfit <> 0 Then
        sts = EsbOtlGetMemberFormula(hOutline, hMemberProfit, szFormula, 100)
    End If
End Sub

See Also

※ EsbOtlSetMemberFormula
※ EsbOtlDeleteMemberFormula

EsbOtlGetMemberInfo

Gets member information for the specified member.

Syntax

EsbOtlGetMemberInfo (hOutline, hMember, pInfo)
ByVal hOutline As Long
ByVal hMember As Long
    pInfo As ESB_MBRINFO_T
Parameter | Description
--- | ---
hOutline | Outline context handle.
hMember | Member handle.
pInfo | Return variable for the member information structure. This structure is allocated by the caller.

Notes
- The member handle can be retrieved by calling `EsbOtlFindMember()`.
- Two fields of the “ESB_MBRINFO_T” on page 1460 structure are for attributes only:
  - Attribute
  - IsAttributed

Return Value
Returns 0 if successful.

Example

```visualbasic
Declare Function EsbOtlGetMemberInfo Lib "ESBOTLN" (ByVal hOutline As Long, ByVal hMember As Long, pInfo As ESB_MBRINFO_T) As Long

Sub ESB_OtlGetMemberInfo()
    Dim sts As Long
    Dim Object As ESB_OBJDEF_T
    Dim hOutline As Long
    Dim MbrInfo As ESB_MBRINFO_T
    Dim hMemberProfit As Long
    Object.hCtx = hCtx
    Object.Type = ESB_OBJTYPE_OUTLINE
    Object.AppName = "Sample"
    Object.DbName = "Basic"
    Object.FileName = "Basic"
    sts = EsbOtlOpenOutline(hCtx, Object, ESB_YES, ESB_YES, hOutline)
    If sts = 0 Then
        sts = EsbOtlFindMember(hOutline, "Profit", hMemberProfit)
    End If
    If sts = 0 And hMemberProfit <> 0 Then
        sts = EsbOtlGetMemberInfo(hOutline, hMemberProfit, MbrInfo)
    End If
End Sub
```

See Also
- `EsbOtlFindMember`
- `EsbOtlGetFirstMember`
**EsbOtlGetMemberLastFormula**

Returns the last formula used to calculate the member.

**Syntax**

```
EsbOtlGetMemberLastFormula (hOutline, hMember, pszFormula, usBufSize)
```

**Parameter**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>Outline context handle</td>
</tr>
<tr>
<td>hMember</td>
<td>Member handle.</td>
</tr>
<tr>
<td>pszFormula</td>
<td>Return variable for the member formula. The buffer is allocated by the caller, and the length is specified in the usBufSize parameter.</td>
</tr>
<tr>
<td>usBufSize</td>
<td>Size of the pszFormula buffer.</td>
</tr>
</tbody>
</table>

**Notes**

- Use `EsbFree()` to free the formula buffer.
- This call will work for both `EsbOtlOpenOutline()` and `EsbOtlOpenOutlineQuery()`.
- `EsbOtlGetMemberLastFormula()` returns the last formula applied to the selected member, which might differ from the Database Outline formula associated with that member.
- The last formula is derived from the last calculation (either from the outline or calc scripts) done on the member.

**Return Value**

The return value is zero if the function was successful.

**Example**

```vbnet
Declare Function EsbOtlGetMemberLastFormula Lib "ESBOTLN"
(ByVal hOutline As Long, ByVal hMember As Long, ByVal pszFormula As String,
ByVal usBufSize As Integer) As Long

Sub ESB_OtlGetMemberLastFormula()
    Dim sts As Long
    Dim Object As ESB_OBJDEF_T
    Dim hOutline As Long
    Dim hMember As Long
    Dim szFormula As String * 100
    Object.hCtx = hCtx
    Object.Type = ESB_OBJTYPE_OUTLINE
    Object.AppName = "Sample"
    Object.DbName = "Basic"
    Object.FileName = "Basic"
    sts = EsbOtlOpenOutline(hCtx, Object, ESB_YES, ESB_YES, hOutline)
    If sts = 0 Then
        sts = EsbOtlFindMember(hOutline, "Margin", hMember)
```
End If
If sts = 0 And hMember <> 0 Then
    sts = EsbOtlGetMemberLastFormula(hOutline, hMember, szFormula, 100)
End If
End Sub

See Also

- EsbOtlDeleteMemberFormula
- EsbOtlGetMemberFormula
- EsbOtlOpenOutline
- EsbOtlOpenOutlineQuery
- EsbOtlSetMemberFormula

EsbOtlGetNextAliasCombination

Returns the alias combinations for the specified member in the specified alias table. The alias is returned in the `pszAlias` parameter, and the member combination is returned in `pszCombination`.

Syntax

```vbnet
EsbOtlGetNextAliasCombination (hOutline, hMember, pszAliasTable, pszAlias, pszCombination, usBufSize)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>Outline context handle.</td>
</tr>
<tr>
<td>hMember</td>
<td>Handle of member to retrieve the alias combination from.</td>
</tr>
<tr>
<td>pszAliasTable</td>
<td>Alias table to retrieve the alias combination from. If this parameter is &quot;&quot;, the default alias table is used.</td>
</tr>
<tr>
<td>pszAlias</td>
<td>Buffer for the return of the next alias. The next alias is determined by what is specified in this parameter. If a zero-length string, the first alias is returned. If the parameter is a valid alias combination, the next alias is returned.</td>
</tr>
<tr>
<td>pszCombination</td>
<td>Member combination of the returned alias. This buffer is allocated by the caller.</td>
</tr>
<tr>
<td>usBufSize</td>
<td>Size of the <code>pszCombination</code> buffer.</td>
</tr>
</tbody>
</table>

Notes

- You should allocate space of size ESB_MBRNAMELINE for `pszAlias` before calling.
- You should allocate the space for `pszCombination`. The caller should set the length of this buffer in the `usBufSize` parameter.
The *pszAlias* parameter is used to find the next combination. See the description of this parameter for details on how to retrieve the next combination.

If there are no (more) alias combinations, *pszCombination* is set to ESB_NULL and the call returns 0.

**Return Value**

Returns 0 if successful; otherwise: OTLAPI_BAD_ALIASTABLE

**Example**

```vbnet
Declare Function EsbOtlGetNextAliasCombination Lib "ESBOTLN" (ByVal hOutline As Long, ByVal hMember As Long, ByVal pszAliasTable As String, ByVal pszAlias As String, ByVal pszCombination As String, ByVal usBufSize As Integer) As Long

Sub ESB_OtlGetNextAliasCombination()
    Dim sts As Long
    Dim Object As ESB_OBJDEF_T
    Dim hOutline As Long
    Dim hMemberJan As Long
    Dim szAlias As String * ESB_MBRNAMELEN
    Dim szCombination As String * 100
    Object.hCtx = hCtx
    Object.Type = ESB_OBJTYPE_OUTLINE
    Object.AppName = "Sample"
    Object.DbName = "Basic"
    Object.FileName = "Basic"
    sts = EsbOtlOpenOutline(hCtx, Object, ESB_YES, ESB_YES, hOutline)
    If sts = 0 Then
        sts = EsbOtlFindMember(hOutline, "Jan", hMemberJan)
    End If
    If sts = 0 And hMemberJan <> 0 Then
        szCombination = "xxx"
        Do While sts = 0 And Left$(szCombination, 1) <> Chr$(0)
            sts = EsbOtlGetNextAliasCombination
                      (hOutline, hMemberJan, "Default", szAlias, szCombination, 100)
        Loop
    End If
End Sub
```

**See Also**

- EsbOtlAddAliasCombination
- EsbOtlDeleteAliasCombination

**EsbOtlGetNextSharedMember**

Returns the member handle to the next shared member of the specified member.
**Syntax**

```plaintext
EsbOtlGetNextSharedMember (hOutline, hMember, phMember)
```

**Parameter**  
**Description**

- **hOutline**  
  Outline context handle.

- **hMember**  
  Member to find the next shared member for.

- **phMember**  
  Return variable for the next shared member in the outline. This parameter is ESB_NULL if there are no more shared members.

**Notes**

- If `hMember` is the actual member, the first shared member is returned in the `phMember` parameter. If `hMember` is a shared member, the next shared member is returned in the `phMember` parameter.

- If there are no (more) shared members, `phMember` is set to ESB_NULL and the call returns 0.

**Return Value**

Returns 0 if successful.

**Example**

```vbnet
Declare Function EsbOtlGetNextSharedMember Lib "ESBOTLN" (ByVal hOutline As Long, ByVal hMember As Long, phMember As Long) As Long

Sub ESB_OtlGetNextSharedMember()
    Dim sts As Long
    Dim Object As ESB_OBJDEF_T
    Dim hOutline As Long
    Dim hMargin As Long
    Dim hShared As Long
    Object.hCtx = hCtx
    Object.Type = ESB_OBJTYPE_OUTLINE
    Object.AppName = "Sample"
    Object.DbName = "Basic"
    Object.FileName = "Basic"
    sts = EsbOtlOpenOutline(hCtx, Object, ESB_YES, ESB_YES, hOutline)
    If sts = 0 Then
        sts = EsbOtlFindMember(hOutline, "Margin", hMargin)
    End If
    If sts = 0 Then
        Do While sts = 0 And hMargin <> 0
            sts = EsbOtlGetNextSharedMember(hOutline, hMargin, hShared)
            hMargin = hShared
            hShared = ESB_NULL
        Loop
    End If
End Sub
```

1534 Visual Basic Outline API Functions
Returns the next sibling of a member.

Syntax

EsbOtlGetNextSibling (hOutline, hMember, phMember)
ByVal hOutline As Long
ByVal hMember As Long
    phMember As Long

Parameter | Description
----------|-----------------
hOutline   | Outline context handle.
hMember    | Handle of member to retrieve the sibling of.
phMember   | Return variable for the handle of the sibling of the hMember parameter.

Notes

If there is no next sibling, phMember is set to ESB_NULL and the call returns 0.

Return Value

Returns 0 if successful.

Example

 Declare Function EsbOtlGetNextSibling Lib "ESBOTLN" (ByVal hOutline As Long, ByVal hMember As Long, phMember As Long) As Long

Sub ESB_OtlGetNextSibling()
    Dim sts As Long
    Dim Object As ESB_OBJDEF_T
    Dim hOutline As Long
    Dim hChild As Long
    Dim hNextSibling As Long
    Object.hCtx = hCtx
    Object.Type = ESB_OBJTYPE_OUTLINE
    Object.AppName = "Sample"
    Object.DbName = "Basic"
    Object.FileName = "Basic"
    sts = EsbOtlOpenOutline(hCtx, Object, ESB_YES, ESB_YES, hOutline)
    If sts = 0 Then
        sts = EsbOtlFindMember(hOutline, "Profit", hChild)
    End If
    If sts = 0 And hChild <> 0 Then

See Also

• EsbOtlFindMember
sts = EsbOtlGetNextSibling(hOutline, hChild, hNextSibling)
End If
End Sub

See Also

- EsbOtlGetPrevSibling
- EsbOtlGetParent
- EsbOtlGetChild
- EsbOtlGetFirstChild

**EsbOtlGetOutlineInfo**

Returns information about the outline file.

**Syntax**

```vba
EsbOtlGetOutlineInfo (hOutline, pInfo, pusCount)
```

**Parameter Description**

- **hOutline**  Outline context handle.
- **pInfo**  Return variable for the information structure. The ESB_OUTLINEINFO_T structure should be allocated by the caller.
- **pusCount**  Return variable for the number of alias tables in the outline.

**Notes**

- The caller should call EsbGetNextItem() once for each alias table (returned in the `pusCount` variable).

**Return Value**

Returns 0 if successful.

**Example**

```vba
Declare Function EsbOtlGetOutlineInfo Lib "ESBOTLN" (ByVal hOutline As Long, pInfo As ESB_OUTLINEINFO_T, pusCount As Integer) As Long
Sub ESB_OtlGetOutlineInfo()
    Dim sts As Long
    Dim Object As ESB_OBJDEF_T
    Dim hOutline As Long
    Dim Info As ESB_OUTLINEINFO_T
    Dim szAliasTable As String * ESB_ALIASNAMELEN
    Dim usCount As Integer
    Object.hCtx = hCtx
    Object.Type = ESB_OBJTYPE_OUTLINE
    Object.AppName = "Sample"
    Object.DbName = "Basic"
End Sub
```

1536 Visual Basic Outline API Functions
sts = EsbOtlOpenOutline(hCtx, Object, ESB_YES, ESB_YES, hOutline)
If sts = 0 Then
    sts = EsbOtlGetOutlineInfo(hOutline, Info, usCount)
    Do While sts = 0 And usCount > 0
        sts = EsbGetNextItem(hCtx, ESB_OUTLINEINFO_TYPE,
            ByVal szAliasTable)
        usCount = usCount - 1
    Loop
End If
End Sub

See Also
- EsbOtlSetOutlineInfo

EsbOtlGetParent

Retrieves the parent of a member.

Syntax

EsbOtlGetParent (hOutline, hMember, phMember)
ByVal hOutline As Long
ByVal hMember As Long
    phMember As Long

Parameter Description

hOutline  Outline context handle.
hMember   Handle of member to retrieve the parent of.
phMember  Return variable for the member handle of the parent of the hMember parameter.

Notes
- If there is no parent, phMember is set to ESB_NULL and the call returns 0. (hMember is a dimension.)

Return Value

Returns 0 if successful.

Example

Declare Function EsbOtlGetParent Lib "ESBOTLN" (ByVal hOutline As Long, ByVal hMember As Long, phMember As Long) As Long

Sub ESB_OtlGetParent()
    Dim sts As Long
    Dim Object As ESB_OBJDEF_T
    Dim hOutline As Long
    Dim hMemberProfit As Long
    Dim hParent As Long
Object.hCtx = hCtx
Object.Type = ESB_OBJTYPE_OUTLINE
Object.AppName = "Sample"
Object.DbName = "Basic"
Object.FileName = "Basic"
sts = EsbOtlOpenOutline(hCtx, Object, ESB_YES, ESB_YES, hOutline)
If sts = 0 Then
    sts = EsbOtlFindMember(hOutline, "Profit", hMemberProfit)
End If
If sts = 0 And hMemberProfit <> 0 Then
    sts = EsbOtlGetParent(hOutline, hMemberProfit, hParent)
End If
End Sub

See Also

- EsbOtlGetChild
- EsbOtlGetNextSibling
- EsbOtlGetPrevSibling
- EsbOtlGetFirstMember

**EsbOtlGetPrevSibling**

Returns the previous sibling of a member.

**Syntax**

```visualbasic
EsbOtlGetPrevSibling (hOutline, hMember, phMember)
```

**Parameter**

- `hOutline`  
  Outline context handle.
- `hMember`  
  Handle of member to retrieve the previous sibling of.
- `phMember`  
  Return variable for the handle of the previous sibling of the `hMember` parameter.

**Notes**

- If there is no previous sibling, `phMember` is set to ESB_NULL and the call returns 0.

**Return Value**

Returns 0 if successful.

**Example**

```visualbasic
Declare Function EsbOtlGetPrevSibling Lib "ESBOTLN" (ByVal hOutline As Long, ByVal hMember As Long, ByVal phMember As Long) As Long
```
Sub ESB_OtlGetPrevSibling()
Dim sts As Long
Dim Object As ESB_OBJDEF_T
Dim hOutline As Long
Dim hChild As Long
Dim hPrevSibling As Long
Object.hCtx = hCtx
Object.Type = ESB_OBJTYPE_OUTLINE
Object.AppName = "Sample"
Object.DbName = "Basic"
Object.FileName = "Basic"
sts = EsbOtlOpenOutline(hCtx, Object, ESB_YES, ESB_YES, hOutline)
If sts = 0 Then
    sts = EsbOtlFindMember(hOutline, "Profit", hChild)
End If
If sts = 0 And hChild <> 0 Then
    sts = EsbOtlGetPrevSibling(hOutline, hChild, hPrevSibling)
End If
End Sub

See Also
- EsbOtlGetNextSibling
- EsbOtlGetParent
- EsbOtlGetChild
- EsbOtlGetFirstMember

EsbOtlGetUpdateTime

Returns a timestamp for the specified outline.

Syntax
EsbOtlGetUpdateTime (hOutline, TimeStamp)
ByVal hOutline As Long
    TimeStamp As Long

Parameter Description
hOutline      The outline handle
TimeStamp    The timestamp for the outline

Notes
- The value for time (of type Long) is represented by the number of seconds since 00:00:00 1/1/1970 GMT.
- The value for time is not persistent; that is, the value for time is reset whenever the server loads the database.

Return Value
Returns the timestamp for the specified outline.
See Also

- EsbOtlGetOutlineInfo
- EsbOtlSetOutlineInfo
- EsbOtlVerifyOutline
- EsbOtlSortChildren
- EsbOtlGenerateCurrencyOutline

EsbOtlGetUserAttributes

Retrieves all user-defined attributes for a member.

Syntax

```
EsbOtlGetUserAttributes (hOutline, hMember, pusCount)
```

ByVal hOutline As Long
ByVal hMember As Long
    pusCount As Integer

Parameter Description

- **hOutline**  Outline context handle.
- **hMember**  Handle of member for which to get the user-defined attribute.
- **pusCount**  Count of user attributes returned; defines the number of elements in the `ppAttributeList` array.

Notes

- Call EsbGetNextItem() once for each user-defined attribute (*pusCount attributes).
- A caller can set any number of user-defined attributes for a member using EsbOtlSetUserAttribute(). Each attribute is defined as a unique string that follows the same conventions as member names.
- A user attribute can be the same as any member name, alias, or generation or level name.

Return Value

Returns 0 if successful.

Example

```
Declare Function EsbOtlGetUserAttributes Lib "ESBOTLN" (ByVal hOutline As Long, ByVal hMember As Long, 
pusCount As Integer) As Long

Sub ESB_OtlGetUserAttributes()
    Dim sts As Long
    Dim Object As ESB_OBJDEF_T
    Dim hOutline As Long
    Dim hMember As Long
    Dim AttributeList As String * ESB_MBRNAMELEN
    Dim n As Integer
    Dim Count As Integer
    Object.hCtx = hCtx
    Object.Type = ESB_OBJTYPE_OUTLINE
```

1540 Visual Basic Outline API Functions
Object.AppName = "Sample"
Object.DbName = "Basic"
Object.FileName = "Basic"

sts = EsbOtlOpenOutline(hCtx, Object, ESB_YES, ESB_YES, hOutline)
If sts = 0 Then
    sts = EsbOtlFindMember(hOutline, "Jan", hMember)
End If
If sts = 0 And hMember <> 0 Then
    '********************
    ' Get User Attributes
    '********************
    sts = EsbOtlGetUserAttributes(hOutline, hMember, Count)
End If
If sts = 0 And Count <> 0 Then
    For n = 1 To Count
        '********************************
        ' Get next User Attribute String
        ' from the list
        '********************************
        sts = EsbGetNextItem(hCtx, ESB_OTLUSERATTR_TYPE, ByVal AttributeList)
    Next
End If
End Sub

See Also

- EsbOtlDeleteUserAttribute
- EsbOtlSetUserAttribute

**EsbOtlMoveMember**

Moves a member.

**Syntax**

```
EsbOtlMoveMember (hOutline, hMember, hNewParent, hNewPrevSibling)
```

**Parameter** | **Description**
---|---
hOutline | Outline context handle.
hMember | Handle of member to move.
hNewParent | Handle of new parent. This field is only used if the `hNewPrevSibling` field is ESB_NULL.
hNewPrevSibling | Handle of new previous sibling.
Notes

- The moved member is inserted following the $hPrevSibling$ member. If this field is ESB_NULL, the moved member becomes the first child of the parent specified by $hParent$.
- If both $hParent$ and $hPrevSibling$ are ESB_NULL, the moved member becomes the first dimension in the outline.
- Moving a zero-level (leaf node) attribute member that is not of type ESB_ATTRMBRDT_STRING resets the member's long name, using the specifications for the outline in the "ESB_ATTRSPECS_T" on page 1169 structure.
- Moving an ancestor may affect the long name of a zero-level attribute member.

Return Value

Returns 0 if successful; otherwise:

OTLAPI_BAD_MOVE

Example

Declare Function EsbOtlMoveMember Lib "ESBOTLN" (ByVal hOutline As Long, ByVal hMember As Long, ByVal hNewParent As Long, ByVal hNewPrevSibling As Long) As Long

Sub ESB_OtlMoveMember()
  Dim sts As Long
  Dim Object As ESB_OBJDEF_T
  Dim hOutline As Long
  Dim hMemberProfit As Long
  Dim hQ As Long
  Object.hCtx = hCtx
  Object.Type = ESB_OBJTYPE_OUTLINE
  Object.AppName = "Sample"
  Object.DbName = "Basic"
  Object.FileName = "Basic"
  sts = EsbOtlOpenOutline(hCtx, Object, ESB.YES, ESB.YES, hOutline)
  If sts = 0 Then
    sts = EsbOtlFindMember(hOutline, "First Q", hQ)
  End If
  If sts = 0 Then
    sts = EsbOtlFindMember(hOutline, "Profit", hMemberProfit)
  End If
  If sts = 0 And hQ And hMemberProfit Then
    sts = EsbOtlMoveMember(hOutline, hQ, hMemberProfit, ESB_NULL)
  End If
End Sub

See Also

- EsbOtlFindMember
- EsbOtlRenameMember
- EsbOtlAddMember
- EsbOtlDeleteMember

1542 Visual Basic Outline API Functions
**EsbOtlNewOutline**

Creates an outline without creating a file. Used as an alternative to **EsbOtlOpenOutline()**.

**Syntax**

```vba
EsbOtlNewOutline (hCtx, pNewInfo, phOutline)
```

<table>
<thead>
<tr>
<th>ByVal hCtx As Long</th>
</tr>
</thead>
<tbody>
<tr>
<td>pNewInfo As ESB_OUTLINEINFO_T</td>
</tr>
<tr>
<td>phOutline As Long</td>
</tr>
</tbody>
</table>

**Parameter Description**

- **hCtx**  Essbase Context handle.
- **pNewInfo**  Structure describing the new outline.
- **phOutline**  Return variable for the ESB_HOUTLINE_T value. This handle is set by the API and should be passed to subsequent Outline API functions.

**Notes**

- This function creates an empty outline in memory.
- No transactions are kept when this call is used. See **EsbOtlOpenOutline()** for more information on keeping transactions.

**Return Value**

Returns 0 if successful.

**Example**

```vba
Declare Function EsbOtlNewOutline Lib "ESBOTLN.DLL" (ByVal hCtx As Long, pNewInfo As ESB_OUTLINEINFO_T, phOutline As Long) As Long

Sub ESB_OtlNewOutline()
 Dim sts As Long
 Dim NewInfo As ESB_OUTLINEINFO_T
 Dim hOutline As Long
 NewInfo.usOutlineType = ESB_DBTYPE_NORMAL
 NewInfo.fCaseSensitive = ESB_FALSE
 NewInfo.fAutoConfigure = ESB_TRUE
 sts = EsbOtlNewOutline(hCtx, NewInfo, hOutline)
 End Sub
```

**See Also**

- **EsbOtlOpenOutline**
- **EsbOtlWriteOutline**
- **EsbOtlRestructure**
- **EsbOtlCloseOutline**
- **EsbOtlVerifyOutline**
EsbOtlOpenOutline

Opens and reads in an existing outline. You must call this function (or EsbOtlNewOutline()) before any operations on the outline can take place.

Syntax

EsbOtlOpenOutline (hCtx, pObject, fLock, fKeepTrans, phOutline)

ByVal hCtx As Long
pObject As ESB_OBJDEF_T
ByVal fLock As Integer
ByVal fKeepTrans As Integer
phOutline As Long

Parameter Description

hCtx Essbase Context handle.
pObject Outline object to open.
fLock Flag to determine if the outline should be locked when it is opened. This is valid only for server outlines.
fKeepTrans Flag to determine whether to keep transactions.
    If you are opening an existing outline to make changes, and you intend to restructure the database and keep data, we recommend that you set this flag to ESB_YES. When ESB_YES, a log is kept of activities done to the outline.
    If you are starting from an empty outline or are not planning on saving data when you restructure, we recommend that you set this field to ESB_NO. When ESB_NO, no log is kept, saving time and memory.
phOutline Return variable for ESB_HOUTLINE_T variable. This handle is set by the API and should be passed to subsequent Outline API functions.

Notes

- If the outline file exists on the server, this call copies the file locally for client access.

Return Value

Returns 0 if successful; otherwise one of the following:

- OTLAPI_BAD_OBJTYPE
- OTLAPI_ERR_FILEOPEN
- OTLAPI_ERR_FILEIO

Access

This function requires you to have the appropriate level of access to the specified application and/or database to contain the outline object. To lock the outline object (lock flag is ESB_YES), you must have Application Designer or Database Designer privilege (ESB_PRIV_APPDESIGN or ESB_PRIV_DBDESIGN) for the specified application or database containing the outline.

Example

Declare Function EsbOtlOpenOutline Lib "ESBOTLN.DLL" (ByVal hCtx As Long, pObject As ESB_OBJDEF_T, ByVal fLock As Integer, ByVal fKeepTrans As Integer,
Sub ESB_OtlOpenOutline()
    Dim sts As Long
    Dim Object As ESB_OBJDEF_T
    Dim hOutline As Long
    Object.hCtx = hCtx
    Object.Type = ESB_OBJTYPE_OUTLINE
    Object.AppName = "Sample"
    Object.DbName = "Basic"
    Object.FileName = "Basic"
    sts = EsbOtlOpenOutline(hCtx, Object, ESB_YES, ESB_YES, hOutline)
End Sub

See Also
- EsbOtlNewOutline
- EsbOtlWriteOutline
- EsbOtlRestructure
- EsbOtlCloseOutline
- EsbOtlVerifyOutline

EsbOtlOpenOutlineQuery
Opens an existing outline.

Syntax

EsbOtlOpenOutlineQuery (hCtx, pObject, phOutline)
ByVal hCtx As Long
    pObject As ESB_OBJDEF_T
    phOutline As Long

Parameter Description

hCtx Outline context handle. This must be a valid server login context.

pObject Pointer to object structure defining the outline object to open. Currently this is ignored. You should call EsbSetActive() for the database you are accessing.

phOutline Pointer to an ESB_HOUTLINE_T variable. This will be set by the API and should be passed in to subsequent API functions.

Notes
- Use this function to access an outline using EsbOtlQueryMembers().
- The call will not download the outline and load the entire file into memory.
- Therefore, many of the outline API calls will not work with hOutline that is passed back from this call.
- The following calls are accessible after this call is made. All other Outline API calls will return an error.
  - EsbOtlCloseOutline
Return Value

The return value is zero if the function was successful.

- OTLAPI_BAD_OBJTYPE
- OTLAPI_ERR_FILEOPEN
- OTLAPI_ERR_FILEIO

Example

Declare Function EsbOtlOpenOutlineQuery Lib "ESBOTLN.DLL"
(ByVal hCtx As Long, pObject As ESB_OBJDEF_T, phOutline As Long) As Long

Sub ESB_OtlOpenOutlineQuery()
    Dim sts As Long
    Dim hOutline As Long
    Dim Object As ESB_OBJDEF_T
    Dim Access As Integer
    DimAppName As String
    DimDbName As String
    AppName = "Sample"
    DbName = "Basic"
    sts = EsbSetActive(hCtx, AppName, DbName, Access)
    If sts = 0 Then
        sts = EsbOtlOpenOutlineQuery(hCtx, Object, hOutline)
    End If
End Sub

See Also

- EsbOtlCloseOutline
- EsbOtlOpenOutline
- EsbOtlQueryMembers
- EsbOtlQueryMembersByName
- EsbSetActive
**EsbOtlQueryAttributes**

Queries member information for a given attribute member or dimension.

**Syntax**

```vbscript
EsbOtlQueryAttributes (hOutline, AttrQuery, Count, MemberArray)
```

**Parameter Description**

- **hOutline** Handle to the outline
- **AttrQuery** Structure that defines the query
- **Count** Number of member handles returned
- **MemberArray** Array of member handles returned

**Notes**

Before you call this function, call `EsbOpenOutlineQuery()` to open the outline in query mode.

**Access**

This function requires no special privileges.

**Example**

```vbscript
Sub ESB_OtlQueryAttributes()
    Dim OutAttrInfo As ESB_ATTRIBUTEINFO_T
    Dim InAttrQuery As ESB_ATTRIBUTEQUERY_T
    Dim MbrInfo As ESB_MBRINFO_T
    Dim index As Integer
    Dim test As Integer
    Dim Count As Long
    Dim sts As Long
    Dim Dummy As String
    Dim MbrName As String
    Dim attribdtvar As Variant
    Dim OutMemberArray As Variant
    InAttrQuery.InputMember = "Product"
    InAttrQuery.InputMemberType = ESB_STANDARD_DIMENSION
    InAttrQuery.OutputMemberType = ESB_ATTRIBUTE_DIMENSION
    InAttrQuery.Operation = ESB_ALL
    InAttrQuery.Attribute = ""
    sts = EsbOtlQueryAttributes(ghOutline, InAttrQuery, Count, OutMemberArray)
    If sts = 0 Then
        Out "attribute query Count is : " & Count
        Out "EsbOtlGetMemberInfo passed"
        For index = 0 To Count - 1
            sts = EsbOtlGetMemberInfo(ghOutline, OutMemberArray(index), MbrInfo)
            If sts = 0 Then
                Out "MbrName : " & MbrInfo.szMember
        Next index
    End If
End Sub
```
Else
    Out "EsbOtlGetMemberInfo Failed: " & sts
End If

Else
    Next index
    Out "EsbOtlQueryAttributes failed: " & sts
End If
End Sub

See Also

- EsbCheckAttributes
- EsbGetAssociatedAttributesInfo
- EsbGetAttributeInfo
- EsbGetAttributeSpecifications
- EsbOtlAssociateAttributeDimension
- EsbOtlAssociateAttributeMember
- EsbOtlDisassociateAttributeDimension
- EsbOtlDisassociateAttributeMember
- EsbOtlFindAttributeMembers
- EsbOtlGetAssociatedAttributes
- EsbOtlGetAttributeInfo
- EsbOtlGetAttributeSpecifications
- EsbOtlGetAttributeSpecifications

**EsbOtlQueryMembers**

Queries the outline.

**Syntax**

EsbOtlQueryMembers (hOutline, hMember, pPredicate, pMbrCounts, pulCount)

ByVal hOutline As Long
ByVal hMember As Long
    pPredicate As ESB_PREDICATE_T
    pMbrCounts As ESB_MBRCOUNTS_T

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>Essbase outline handle. This must have been returned from EsbOtlOpenOutlineQuery().</td>
</tr>
<tr>
<td>hMember</td>
<td>The member outline handle of the member to do the operation on. If this value is NULL, it is assumed to be the very top of the outline, representing the logical parent of the dimensions. This value will be ignored for the following options:</td>
</tr>
<tr>
<td></td>
<td>- ESB_NAMEDGENERATION</td>
</tr>
<tr>
<td></td>
<td>- ESB_NAMEDLEVEL</td>
</tr>
<tr>
<td></td>
<td>- ESB_USERATTRIBUTE</td>
</tr>
<tr>
<td></td>
<td>- ESB_SEARCH</td>
</tr>
<tr>
<td></td>
<td>- ESB_WILDSEARCH</td>
</tr>
<tr>
<td>pPredicate</td>
<td>Structure defining the query. The fields of this structure are described in Notes.</td>
</tr>
</tbody>
</table>
Parameter | Description
--- | ---
pMbrCounts | Structure defining information about counts. It contains the following fields:
  | ulStart — Starting number to return.
  | ulMaxCount — Maximum number of member handles to return.
  | ulTotalCount — Total number of members that are defined in the results of the query.
  | pulReturnCount — Number of member handles returned in this query.

phMemberArray | An array of member handles returned from the query.

Notes

- The call takes a member handle to operate on and returns an array of member handles satisfying the criteria specified by the option value.
- The caller should call `EsbOtlFreeMember()` when the returned phMembers member array is no longer needed.
- Each hMember element in the array can only be used in calls that are listed in `EsbOtlOpenOutlineQuery()`. For example, a returned member handle cannot be used to call `EsbOtlGetSibling()`.
- The programmer should call `EsbGetNextItem()` once for each member handle returned.
- The fields of the `pPredicate` structure are used as follows:
  - `ulQuery` — Value defining the operation to perform. It can be one of the following:
    - ESB_CHILDREN
    - ESB_DESCENDANTS
    - ESB_BOTTOMLEVEL
    - ESB_SIBLINGS
    - ESB_SAMELEVEL
    - ESB_SAMEGENERATION
    - ESB_PARENT
    - ESB_DIMENSION
    - ESB_NAMEDGENERATION
    - ESB_NAMEDLEVEL
    - ESB_SEARCH
    - ESB_WILDSEARCH
    - ESB_USERATTRIBUTE
    - ESB_ANCESTORS
    - ESB_DTSMEMBERS
    - ESB_DIMUSERATTRIBUTES
  - `ulOptions` — Value defining any options. It is used with the following query options:
- ESB_SEARCH, ESB_WILDSEARCH—One of the following values:
  - ESB_MEMBERSONLY
  - ESB_ALIASESONLY
  - ESB_MEMBERSANDALIASES
- All Options—ESB_COUNTONLY: Returns no member handles, but only fills in the pTotalCount field in the pCounts structure.
- `szDimension`—Dimension to limit the scope of the query. It is used with the following query options and ignored otherwise:
  - ESB_NAMEDGENERATION
  - ESB_NAMEDLEVEL
  - ESB_USERATTRIBUTE
  - ESB_SEARCH—Set to NULL to search through all dimensions
  - ESB_WILDSEARCH—Set to NULL to search through all dimensions
- `pszString1`—Input string that is determined by the option. It is used with the following query options and ignored otherwise:
  - ESB_NAMEDGENERATION—Name of the generation.
  - ESB_NAMEDLEVEL—Name of the level.
  - ESB_SEARCH—String to search for. The string is defined as an exact.
  - ESB_WILDSEARCH—String to search for. The string is defined as an exact search string with an optional '*' at the end to mean any set of characters.
  - ESB_USERATTRIBUTE—User defined attribute.
- `pszString2`—Input string that is determined by the option. It is used with the following query options and ignored otherwise:
  - ESB_USERATTRIBUTE—User defined attribute.
  - ESB_SEARCH, ESB_WILDSEARCH—If the options are set to look in the alias tables, specifies the alias table to search in. If it is null, all alias tables will be searched.

**Return Value**

The return value is zero if the function was successful.

**Example**

Declare Function EsbOtlQueryMembers Lib "ESBOTLN"
(ByVal hOutline As Long, ByVal hMember As Long, 
pPredicate As ESB_PREDICATE_T, pCounts As ESB_MBRCOUNTS_T) As Long
Declare Function EsbOtlFreeMember Lib "ESBOTLN"
(ByVal hOutline As Long, ByVal hMember As Long) As Long

Sub ESB_OtlQueryMembers()
  Dim sts As Long
  Dim hOutline As Long
  Dim hMember As Long
  Dim ihMember As Long

1550 Visual Basic Outline API Functions
Dim Object As ESB_OBJDEF_T
Dim MbrInfo As ESB_MBRINFO_T
Dim Predicate As ESB_PREDICATE_T
Dim Counts As ESB_MBRCOUNTS_T
Dim Access As Integer
DimAppName As String
Dim DbName As String
AppName = "Sample"
DbName = "Basic"
sts = EsbOtlOpenOutlineQuery(hCtx, Object, hOutline)
If sts = 0 Then
    sts = EsbOtlOpenOutlineQuery(hCtx, Object, hOutline)
Predicate.ulQuery = ESB_CHILDREN
Predicate pszDimension = "Year"
Counts.ulStart = 0
Counts.ulMaxCount = 10
If sts = 0 Then
    sts = EsbOtlQueryMembers(hOutline, hMember, Predicate, Counts)
If sts = 0 And Counts.ulReturnCount <> 0 Then
    For n% = 1 To Counts.ulReturnCount
        sts = EsbGetNextItem(hCtx, ESB_HMEMBER_TYPE, ihMember)
        If sts = 0 And ihMember <> 0 Then
            sts = EsbOtlFreeMember(hOutline, ihMember)
    Next
End If
End If
End If
End Sub

See Also

- EsbGetNextItem
- EsbOtlFreeMember
- EsbOtlGetDimensionUserAttributes
- EsbOtlOpenOutlineQuery
- EsbOtlQueryMembersByName

**EsbOtlQueryMembersByName**

Queries the outline.

**Syntax**

```
EsbOtlQueryMembersByName (hOutline, pszMember, pPredicate, pCounts)
```

**Parameter**   **Description**

- **hOutline** Essbase outline handle. This must have been returned from EsbOtlOpenOutlineQuery().
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pszMember</td>
<td>The member name string of the member to do the operation on. If this value is NULL, it is assumed to be the very top of the outline, representing the logical parent of the dimensions. This value will be ignored for the following options:</td>
</tr>
</tbody>
</table>
|             |   - ESB_NAMEDGENERATION  
|             |   - ESB_NAMEDLEVEL  
|             |   - ESB_USERATTRIBUTE  
|             |   - ESB_SEARCH  
|             |   - ESB_WILDSEARCH  |
| pPredicate  | Structure defining the query. The fields of this structure are described in Notes.                                                                 |
| pCounts     | Structure defining information about counts. It contains the following fields:  
|             |   - ulStart—Starting number to return.  
|             |   - ulMaxCount—Maximum number of member handles to return.  
|             |   - ulTotalCount—Total number of members that are defined in the results of the query.  
|             |   - pulReturnCount—Number of member handles returned in this query. |

**Notes**

- The call takes a member name string to operate on and returns an array of member handles satisfying the criteria specified by the option value.
- The caller should call `EsbOtlFreeMember()` when the returned phMembers member array is no longer needed.
- Each hMember element in the array can only be used in calls that are listed in `EsbOtlOpenOutlineQuery()`. For example, a returned member handle cannot be used to call `EsbOtlGetSibling()`.
- The programmer should call `EsbGetNextItem()` once for each member handle returned.
- The fields of the `pPredicate` structure are used as follows:
  - ulQuery—Value defining the operation to perform. It can be one of the following:
    - ESB_CHILDREN  
    - ESB_DESCENDANTS  
    - ESB_BOTTOMLEVEL  
    - ESB_SIBLINGS  
    - ESB_SAMELEVEL  
    - ESB_SAMEGENERATION  
    - ESB_PARENT  
    - ESB_DIMENSION  
    - ESB_NAMEDGENERATION  
    - ESB_NAMEDLEVEL  
    - ESB_SEARCH
ulOptions—Value defining any options. It is used with the following query options:
- ESB_SEARCH, ESB_WILDSEARCH—One of the following values:
  - ESB_MEMBERVERSONLY
  - ESB_ALIASESONLY
  - ESB_MEMBERSANDALIASES
- All Options—ESB_COUNTONLY: Returns no member handles, but only fills in the pTotalCount field in the pCounts structure

szDimension—Dimension to limit the scope of the query. It is used with the following query options and ignored otherwise:
- ESB_NAMEDGENERATION
- ESB_NAMEDLEVEL
- ESB_USERATTRIBUTE
- ESB_SEARCH—Set to NULL to search through all dimensions
- ESB_WILDSEARCH—Set to NULL to search through all dimensions

pszString1—Input string that is determined by the option. It is used with the following query options and ignored otherwise:
- ESB_NAMEDGENERATION - The name of the generation
- ESB_NAMEDLEVEL - The name of the level
- ESB_SEARCH - The string to search for. The string is defined as an exact search string with an optional '*' at the end to mean any set of characters.
- ESB_WILDSEARCH - The string to search for. The string is defined as an exact search string with an optional '*' at the end to mean any set of characters.
- ESB_USERATTRIBUTE - The user defined attribute

pszString2—Input string that is determined by the option. It is used with the following query options and ignored otherwise:
- ESB_USERATTRIBUTE—User defined attribute.
- ESB_SEARCH, ESB_WILDSEARCH—If the options are set to look in the alias tables, specifies the alias table to search in. If it's null, all alias tables will be searched.

Return Value

The return value is zero if the function was successful.
Example

Declare Function EsbOtlQueryMembersByName Lib "ESBOTLN" (ByVal hOutline As Long, ByVal pszMember As String, pPredicate As ESB_PREDICATE_T, pCounts As ESB_MBRCOUNTS_T) As Long
Declare Function EsbOtlFreeMember Lib "ESBOTLN" (ByVal hOutline As Long, ByVal hMember As Long) As Long

Sub ESB_OtlQueryMembersByName()
    Dim sts As Long
    Dim hOutline As Long
    Dim pszMember As String
    Dim ihMember As Long
    Dim Object As ESB_OBJDEF_T
    Dim MbrInfo As ESB_MBRINFO_T
    Dim Predicate As ESB_PREDICATE_T
    Dim Counts As ESB_MBRCOUNTS_T
    Dim Access As Integer
    Dim AppName As String
    Dim DbName As String

    pszMember = "Qtr1"
    AppName = "Sample"
    DbName = "Basic"
    sts = EsbOtlOpenOutlineQuery(hCtx, Object, hOutline) 'open outline
    If sts = 0 Then 'proceed if open successful
        Predicate.ulQuery = ESB_CHILDREN
        Predicate.pszDimension = "Year"
        Counts.ulStart = 0
        Counts.ulMaxCount = 10
        If sts = 0 Then
            If sts = 0 And Counts.ulReturnCount <> 0 Then
                For n% = 1 To Counts.ulReturnCount
                    sts = EsbGetNextItem(hCtx, ESB_HMEMBER_TYPE, ihMember)
                    If sts = 0 And ihMember <> 0 Then
                        sts = EsbOtlFreeMember(hOutline, ihMember)
                    End If
                Next
            End If
        Else
            msgbox "Outline open failed with error: " & sts
            Endif
    Else
        msgbox "Outline open failed with error: " & sts
    End Sub

See Also

● EsbGetNextItem
● EsbOtlFreeMember
● EsbOtlGetDimensionUserAttributes
● EsbOtlOpenOutlineQuery
● EsbOtlQueryMembers
**EsbOtlRenameAliasTable**

Renames an existing alias table.

**Syntax**

```plaintext
EsbOtlRenameAliasTable (hOutline, pszAliasTable, pszNewAliasTable)
```

**Parameter**

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
</tr>
<tr>
<td>pszAliasTable</td>
</tr>
<tr>
<td>pszNewAliasTable</td>
</tr>
</tbody>
</table>

**Notes**

- The default alias table cannot be renamed from "Default".
- When renaming an alias table, language codes associated with the alias table are preserved in the renamed alias table.

**Return Value**

Returns 0 if successful; otherwise one of the following:

- OTLAPI_BAD_ALIASTABLE
- OTLAPI_ERR_RENAMEDEFALIAS
- OTLAPI_ERR_ALIASTABLENAME
- OTLAPI_ERR_ALIASTABLEEXISTS

**Example**

```vbnet
Declare Function EsbOtlRenameAliasTable Lib "ESBOTLN" (ByVal hOutline As Long, ByVal pszAliasTable As String, ByVal pszNewAliasTable As String) As Long

Sub ESB_OtlRenameAliasTable()
    Dim sts As Long
    Dim Object As ESB_OBJDEF_T
    Dim hOutline As Long
    Object.hCtx = hCtx
    Object.Type = ESB_OBJTYPE_OUTLINE
    Object.AppName = "Sample"
    Object.DbName = "Basic"
    Object.FileName = "Basic"
    sts = EsbOtlOpenOutline(hCtx, Object, ESB_YES, ESB_YES, hOutline)
    If sts = 0 Then
        EsbOtlRenameAliasTable(hOutline, "Alias Table 1", "1st Alias Table")
    End If
End Sub
```
EsbOtlRenameMember

Renames a member.

Syntax

EsbOtlRenameMember (hOutline, hMember, pszNewMember)
ByVal hOutline As Long
ByVal hMember As Long
ByVal pszNewMember As String

Parameter Description

hOutline Outline context handle.

hMember Handle of member to rename.

pszNewMember New member name.

Notes

● All shared members are also renamed.
● This call fails if hMember points to a shared member.
● Renaming a zero-level (leaf node) attribute member that is not of type ESB_ATTRMRBRDT_STRING resets the following:
  ● the attribute value
  ● the member's long name, using the specifications for the outline in the “ESB_ATTRSPECS_T” on page 1169 structure
● Renaming an ancestor may affect the long name of a zero-level attribute member.

Return Value

Returns 0 if successful; otherwise one of the following:

● OTLAPI_BAD_MBRNAME
● TLAPI_BAD_RENAMESHARE
● OTLAPI_ERR_RENAMENAMEUSED

Example

Declare Function EsbOtlRenameMember Lib "ESBOTLN" (ByVal hOutline As Long, ByVal hMember As Long, ByVal pszNewMember As String) As Long
Sub ESB_OtlRenameMember()
Dim sts As Long
Dim Object As ESB_OBJDEF_T
Dim hOutline As Long
Dim hMemProfit As Long
Object.hCtx = hCtx
Object.Type = ESB_OBJTYPE_OUTLINE
Object.AppName = "Sample"
Object.DbName = "Basic"
Object.FileName = "Basic"
sts = EsbOtlOpenOutline(hCtx, Object, ESB_YES, ESB_YES, hOutline)
If sts = 0 Then
    sts = EsbOtlFindMember(hOutline, "Profit", hMemProfit)
End If
If sts = 0 And hMemProfit <> 0 Then
    sts = EsbOtlRenameMember(hOutline, hMemProfit, "Prelim Profit")
End If
End Sub

See Also
- EsbOtlFindMember
- EsbOtlMoveMember
- EsbOtlAddMember
- EsbOtlDeleteMember

**EsbOtlRestructure**

Restructures an outline on the server. This is an asynchronous call.

**Syntax**

```
EsbOtlRestructure (hCtx, usRestructType)
ByVal hCtx As Long
ByVal usRestructType As Integer
```

**Parameter** | **Description**
--- | ---
hCtx | Server login context handle. This must be the server on which the outline was saved using ESBOTLNriteOutline().
usRestructType | Type of restructuring to do. This can be one of the following values:
- ESB_DOR_ALLDATA
- ESB_DOR_INDATA
- ESB_DOR_LOWDATA
- ESB_DOR_NODATA

**Notes**
- The caller must have saved the outline using ESBOTLNriteOutline() before calling this function.
● This call is only valid for outlines saved to the server.

● This call is asynchronous. You should call EsbGetProcessState() after making this call until
EsbGetProcessState() returns a status indicating the restructure operation is complete.

● In order for data to be properly restructured (saving data), the outline must have been
opened using EsbOtlOpenOutline() with the fKeepTrans flag set to ESB_YES.

Return Value

Returns 0 if successful; otherwise:

OTLAPI_BAD_RESTRUCTTYPE

Access

This function requires you to have the appropriate level of access to the specified application
and/or database to contain the outline object. To restructure the outline object, you must have
Application Designer or Database Designer privilege (ESB_PRIV_APPDESIGN or
ESB_PRIV_DBDESIGN) for the specified application or database containing the outline.

Example

Declare Function EsbOtlRestructure Lib
"ESBOTLN.DLL" (ByVal hCtx As Long, ByVal
usRestructType As Integer) As Long

Sub ESB_OtlRestructure()
Dim hCtx As Long
Dim sts As Long
Dim Object As ESB_OBJDEF_T
Dim hOutline As Long
Object.hCtx = hCtx
Object.Type = ESB_OBJTYPE_OUTLINE
Object.AppName = "Sample"
Object.DbName = "Basic"
Object.FileName = "Basic"
sts = EsbOtlOpenOutline(hCtx, Object,
ESB_YES, ESB_YES, hOutline)
'***
'body of code
'write outline to server using
'ESBOTLNWriteOutline()
'***
If sts = 0 Then
sts = EsbOtlRestructure(hCtx, ESB_DOR_ALLDATA)
End If
'***
'need to call EsbGetProcessState()
'to check for completion before proceeding
'***
End Sub

See Also

● EsbOtlOpenOutline
● EsbOtlNewOutline
● EsbOtlWriteOutline

1558 Visual Basic Outline API Functions
EsbOtlVerifyOutline
EsbOtlCloseOutline

**EsbOtlSetAliasTableLanguage**

Sets a language code for the specified alias table.

By setting alias table language codes, when an application running in an ApplCore session accesses an Essbase database, the correct alias table is automatically selected on application selection.

**Syntax**

```visual basic
ESB_FUNC_M EsbOtlSetAliasTableLanguage (hOutline, pszAliasTable, pszLanguageCode)
ByVal hOutline As Long
ByVal pszAliasTable As String
ByVal pszLanguageCode As String
```

**Parameter**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>Outline context handle.</td>
</tr>
<tr>
<td>pszAliasTable</td>
<td>Name of the alias table for which to set a language code.</td>
</tr>
<tr>
<td>pszLanguageCode</td>
<td>A language code to assign to the alias table specified in pszAliasTable.</td>
</tr>
</tbody>
</table>

Notes:

- You cannot set a language code on the default alias table.
- Any number of language codes can be assigned to an alias table. To set multiple language codes, call this function for each language code.
- Setting a new language code does not override language codes currently assigned to the alias table.
- The same language code must not be assigned to another alias table within the same database.

**Return Value**

- If successful, returns 0.
- If unsuccessful, returns one of the following errors:
  - OTLAPI_BAD_ALIASTABLE (invalid alias table)
  - OTLAPI_ERR_DUP_LANGCODE (the language code is assigned to another alias table within the same database)

**Access**

This function does not require special privileges.
Example

Declare Function EsbOtlGetAliasTableLanguages Lib "esbotln" (ByVal hOutline As Long, ByVal pszAliasTable As String, pulCount As Long) As Long
Declare Function EsbOtlSetAliasTableLanguage Lib "esbotln" (ByVal hOutline As Long, ByVal pszAliasTable As String, ByVal pszLanguageCode As String) As Long
Declare Function EsbOtlClearAliasTableLanguages Lib "esbotln" (ByVal hOutline As Long, ByVal pszAliasTable As String) As Long

Sub ESB_Sub ()
Dim sts As Long
Dim Object As ESB_OBJDEF_T
Dim hOutline As Long
Dim Items As Long
Dim AliasLang As String * ESB_ALIASNAMELEN

Object.hCtx = hCtx
Object.Type = ESB_OBJTYPE_OUTLINE
Object.AppName = "Sample"
Object.DbName = "Basic"
Object.FileName = "Basic"
sts = EsbOtlOpenOutline(hCtx, Object, ESB_YES, ESB_YES, hOutline)
If sts = 0 Then
    sts = EsbOtlCreateAliasTable(hOutline, "French Alias Table")
End If
If sts = 0 Then
    sts = EsbOtlSetAliasTableLanguage(hOutline, "French Alias Table", "fr")
End If
If sts = 0 Then
    sts = EsbOtlSetAliasTableLanguage(hOutline, "French Alias Table", "fr-CA")
End If
If sts = 0 Then
    sts = EsbOtlGetAliasTableLanguages(hOutline, "French Alias Table", Items)
If sts = 0 Then
    For N = 1 To Items
        sts = EsbGetNextItem(hCtx, ESB_ALIASLANG_TYPE, ByVal AliasLang)
    Next
End If
End If
If sts = 0 Then
    sts = EsbOtlClearAliasTableLanguages(hOutline, "French Alias Table")
End If
End Sub

See Also

- EsbOtlGetAliasTableLanguages
- EsbOtlClearAliasTableLanguages
EsbOtlSetAttributeSpecifications

Sets attribute specifications for the outline.

Syntax

EsbOtlSetAttributeSpecifications (hOutline, AttrSpecs)

ByVal hOutline As Long
AttrSpecs As ESB_ATTRSPECS_T

Parameter Description

hOutline    Handle to the outline
AttrSpecs   Attribute specifications

Notes

● Attribute specifications are used to do the following:
  ○ Generate a long name
  ○ Indicate the format of a datetime attribute
  ○ Indicate a numeric attribute's bucketing type
  ○ Provide the name of the attribute calculations dimension and the names for the values used with it
● If you do not set attribute specifications, the outline uses the default attribute specifications.
● Changing attribute specifications may cause a restructure.

Access

This function requires no special privileges.

Example

Sub ESB_OtlSetAttributeSpecifications()
  Dim InAttrSpecs As ESB_ATTRSPECS_T
  eraser
  InAttrSpecs.GenNameBy = InputBox("Enter GenNameBy:" & vbCrLf & "0. ESB_GENNAMEBY_PREFIX" & vbCrLf & "1. ESB_GENNAMEBY_SUFF"")
  InAttrSpecs.UseNameOf = InputBox("Enter UseNameOf:" & vbCrLf & "0. ESB_USENAMEOF_NONE" & vbCrLf & "1. ESB_USENAMEOF_PARENT" & vbCrLf & "2. ESB_USENAMEOF_GRANDPARENTANDPARENT" & vbCrLf & "3. ESB_USENAMEOF_ALLANCESTORS" & vbCrLf & "4. ESB_USENAMEOF_DIMENSION")
  InAttrSpecs.Delimiter = InputBox("Enter Delimiter:" & vbCrLf & "0. ESB_DELIMITER_UNDERSCORE" & vbCrLf & "1. ESB_DELIMITER_PIPE" & vbCrLf & "2. ESB_DELIMITER_CARET")
  InAttrSpecs.DateFormat = InputBox("Enter DateFormat:" & vbCrLf & "0. ESB_DATEFORMAT_MMDDYYYY" & vbCrLf & "1. ESB_DATEFORMAT_DDMMYYYY")
  InAttrSpecs.BucketingType = InputBox("Enter BucketingType:" & vbCrLf & "0. ESB_UPPERBOUNDINCLUSIVE")
End Sub
"1. ESB_ESBLOWERBOUNDINCLUSIVE" & vbCrLf &
"2. ESB_UPPERBOUNDNONINCLUSIVE" & vbCrLf &
"3. ESB_LOWERBOUNDNONINCLUSIVE")
InAttrSpecs.DefaultTrueString = InputBox("Enter DefaultTrueString: ",
"ESB_DEFAULT_TRUESTRING")
InAttrSpecs.DefaultFalseString = InputBox("Enter DefaultFalseString: ",
"ESB_DEFAULT_FALSESTRING")
InAttrSpecs.DefaultAttrCalcDimName = InputBox("Enter DefaultAttrCalcDimName: ",
"ESB_DEFAULT_ATTRIBUTECALCULATIONS")
InAttrSpecs.DefaultSumMbrName = InputBox("Enter DefaultSumMbrName: ",
"ESB_DEFAULT_SUM")
InAttrSpecs.DefaultCountMbrName = InputBox("Enter DefaultCountMbrName: ",
"ESB_DEFAULT_COUNT")
InAttrSpecs.DefaultAverageMbrName = InputBox("Enter DefaultAverageMbrName: ",
"ESB_DEFAULT_AVERAGE")
InAttrSpecs.DefaultMinMbrName = InputBox("Enter DefaultMinMbrName: ",
"ESB_DEFAULT_MIN")
InAttrSpecs.DefaultMaxMbrName = InputBox("Enter DefaultMaxMbrName: ",
"ESB_DEFAULT_MAX")
sts = EsbOtlSetAttributeSpecifications(ghOutline, InAttrSpecs)
If sts = 0 Then
  Out "ESB_OtlSetAttributeSpecifications passed: " & sts
  Out "GenNameBy : " & InAttrSpecs.GenNameBy
  Out "UseNameOf : " & InAttrSpecs.UseNameOf
  Out "Delimiter : " & InAttrSpecs.Delimiter
  Out "DateFormat : " & InAttrSpecs.DateFormat
  Out "BucketingType : " & InAttrSpecs.BucketingType
  Out "DefaultTrueString : " & InAttrSpecs.DefaultTrueString
  Out "DefaultFalseString : " & InAttrSpecs.DefaultFalseString
  Out "DefaultAttrCalcDimName : " & InAttrSpecs.DefaultAttrCalcDimName
  Out "DefaultSumMbrName : " & InAttrSpecs.DefaultSumMbrName
  Out "DefaultCountMbrName : " & InAttrSpecs.DefaultCountMbrName
  Out "DefaultAverageMbrName : " & InAttrSpecs.DefaultAverageMbrName
  Out "DefaultMinMbrName : " & InAttrSpecs.DefaultMinMbrName
  Out "DefaultMaxMbrName : " & InAttrSpecs.DefaultMaxMbrName
Else
  Out "ESB_OtlSetAttributeSpecifications failed" & sts
End If
End Sub

See Also

- EsbCheckAttributes
- EsbGetAssociatedAttributesInfo
- EsbGetAttributeInfo
- EsbGetAttributeSpecifications
- EsbOtlAssociateAttributeDimension
- EsbOtlAssociateAttributeMember
- EsbOtlDisassociateAttributeDimension
- EsbOtlDisassociateAttributeMember
- EsbOtlFindAttributeMembers
- EsbOtlGetAssociatedAttributes
- EsbOtlGetAttributeInfo
- EsbOtlGetAttributeSpecifications
EsbOtlSetDTSMemberAlias

Sets an alias name for a DTS member.

Syntax

EsbOtlSetDTSMemberAlias (hOutline, pszDTSMember, pszAlias, pszAliasTable)

ByVal hOutline As Long
ByVal pszDTSMember As String
ByVal pszAlias As String
ByVal pszAliasTable As String

Parameter Description

hOutline Essbase outline handle returned from the EsbOtlOpenOutlineQuery call.
pszDTSMember Name of the DTS member which provides the alias.
pszAlias Pointer to a C string containing the alias name for the DTS member.
pszAliasTable Name of the alias table which provides the alias. If NULL, the default alias table is used.

Return Value

If successful the return value is zero. Otherwise, one of the following is returned:

- OTLAPI_ERR_DTSMBRNOTDEFINED
- OTLAPI_BAD_ALIASTABLE
- OTLAPI_ERR_ILLEGALALIASSTRING
- OTLAPI_ERR_DUPLICATEALIAS

Example

Public Sub ESB_OtlSetDTSMemberAlias()
    Dim DTSMember As String * ESB_MBRNAMELEN
    Dim Alias As String * ESB_ALIASNAMELEN
    Dim AliasTable As String * ESB_ALIASNAMELEN

    DTSMember = "Y-T-D"
    Alias = "Year_To_Date"
    AliasTable = "default"

    sts = EsbOtlSetDTSMemberAlias(hOutline, DTSMember, _
                                  Alias, AliasTable)
End Sub

See Also

- EsbOtlDeleteDTSMemberAlias
- EsbOtlEnableDTSMember
- EsbOtlGetEnabledDTSMembers
- EsbOtlGetDTSMemberAlias
**EsbOtlSetGenName**

Sets the name for a specific generation within a dimension.

**Syntax**

```
EsbOtlSetGenName (hOutline, pszDimension, usGen, pszName)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>Outline context handle.</td>
</tr>
<tr>
<td>pszDimension</td>
<td>Name of dimension that contains the generation.</td>
</tr>
<tr>
<td>usGen</td>
<td>Number of generation for which to set a name. The dimension itself is generation 1.</td>
</tr>
<tr>
<td>pszName</td>
<td>Name to give the generation.</td>
</tr>
</tbody>
</table>

**Notes**

- The generation name follows the same rules as a member name and must be unique across the entire member name space. It cannot duplicate any other generation, level, member name, or alias. Attempting to add a duplicate name generates an error.
- Each specific dimension and generation must have only one name.

**Return Value**

Returns 0 if successful; otherwise one of the following:

- OTLAPI_BAD_GENLEVELNAME
- OTLAPI_ERR_GENLEVELNAMEEXISTS
- OTLAPI_ERR_GENLEVELVALUE
- LAPI_ERR_NOTADIM
- OTLAPI_ERR_GENLEVELNAMEMBR

**Example**

```visual-basic
Declare Function EsbOtlSetGenName Lib "ESBOTLN" (ByVal hOutline As Long, ByVal pszDimension As String,
ByVal usGen As Integer, ByVal pszName As String) As Long

Sub ESB_OtlSetGenName()
Dim sts As Long
Dim Object As ESB_OBJDEF_T
Dim hOutline As Long
Dim Dimension As String
Dim GenNum As Integer
Dim GenName As String
Object.hCtx = hCtx
```

1564 Visual Basic Outline API Functions
Object.Type = ESB_OBJTYPE_OUTLINE
Object.AppName = "Sample"
Object.DbName = "Basic"
Object.FileName = "Basic"
sts = EsbOtlOpenOutline(hCtx, Object, ESB_YES,
ESB_YES, hOutline)

'*****************************************
'********** Set Generation Name **********
'*****************************************
Dimension = "Year"
GenNum = 2
GenName = "Qtr1 Qtr2 Qtr3 Qtr4"
If Not sts Then
   sts = EsbOtlSetGenName(hOutline, Dimension,
   GenNum, GenName)
End If
End Sub

See Also
- EsbOtlDeleteGenName
- EsbOtlGetGenNames
- EsbOtlGetGenName

**EsbOtlSetLevelName**

Sets the name for a specific level within a dimension.

**Syntax**

```vbnet
EsbOtlSetLevelName(hOutline, pszDimension, usLevel, pszName)
```

- `hOutline` As Long
- `pszDimension` As String
- `usLevel` As Integer
- `pszName` As String

**Parameter** | **Description**
--- | ---
hOutline | Outline context handle.
pszDimension | Name of dimension that contains the level.
usGen | Number of level for which to set a name. Leaf members are level 0.
pszName | Name to give the level.

**Notes**

- The level name follows the same rules as a member name and must be unique across the entire member name space. It cannot duplicate any other generation, level, member name, or alias. Attempting to add a duplicate name generates an error.
- Each specific dimension and level must have only one name.
Return Value

Returns 0 if successful; otherwise one of the following:

- OTLAPI_BAD_GENLEVELNAME
- OTLAPI_ERR_GENLEVELNAMEEXISTS
- OTLAPI_ERR_GENLEVELNAMEEXISTS
- OTLAPI_ERR_NOTADIM
- OTLAPI_ERR_GENLEVELNAMEMBR

Example

Declare Function EsbOtlSetLevelName Lib "ESBOTLN" (ByVal hOutline As Long, ByVal pszDimension As String, ByVal usLevel As Integer, ByVal pszName As String) As Long

Sub ESB_OtlSetLevelName()
Dim sts As Long
Dim Object As ESB_OBJDEF_T
Dim hOutline As Long
Dim Dimension As String
Dim LevelNum As Integer
Dim LevelName As String
Object.hCtx = hCtx
Object.Type = ESB_OBJTYPE_OUTLINE
Object.AppName = "Sample"
Object.DbName = "Basic"
Object.FileName = "Basic"
sts = EsbOtlOpenOutline(hCtx, Object, ESB_YES, ESB_YES, hOutline)
 '************************************
 '********** Set Level Name **********
 '************************************
Dimension = "Year"
LevelNum = 1
LevelName = "Month"
If Not sts Then
    sts = EsbOtlSetLevelName(hOutline, Dimension, LevelNum, LevelName)
End If
End Sub

See Also

- EsbOtlDeleteLevelName
- EsbOtlGetLevelName
- EsbOtlGetLevelNames

EsbOtlSetMemberAlias

Sets the default member alias for a specified member in a specified alias table.
Syntax

**EsbOtlSetMemberAlias** (hOutline, hMember, pszAliasTable, pszAlias)

*ByVal hOutline As Long*
*ByVal hMember As Long*
*ByVal pszAliasTable As String*
*ByVal pszAlias As String*

**Parameter** | **Description**
--- | ---
 hOutline | Outline context handle.
 hMember | Handle of member to set the alias for.
 pszAliasTable | Alias table to set the alias for. If this parameter is "", the default alias table is used.
 pszAlias | Alias.

**Notes**

- The member handle cannot be a shared member. No aliases are allowed for shared members.
- Use **EsbOtlDeleteMemberAlias()** to remove an alias.

**Return Value**

Returns 0 if successful; otherwise one of the following:

- **OTLAPI_BAD_ALIASTABLE**
- **OTLAPI_ERR_ALIASSHARED**
- **OTLAPI_ERR_ILLEGALDEFAULTALIAS**
- **OTLAPI_ERR_ILLEGALCOMBOALIAS**
- **OTLAPI_ERR_ILLEGALALIASSTRING**
- **OTLAPI_ERR_DUPLICATEALIAS**

**Example**

Declare Function EsbOtlSetMemberAlias Lib "ESBOTLN" (ByVal hOutline As Long, ByVal hMember As Long, ByVal pszAliasTable As String, ByVal pszAlias As String) As Long

Sub ESB_OtlSetMemberAlias()
    Dim sts As Long
    Dim Object As ESB_OBJDEF_T
    Dim hOutline As Long
    Dim hMemberYear As Long
    Dim szAlias As String * ESB_MBRNAMELEN
    Object.hCtx = hCtx
    Object.Type = ESB_OBJTYPE_OUTLINE
    Object.AppName = "Sample"
    Object.DbName = "Basic"
    Object.FileName = "Basic"
    sts = EsbOtlOpenOutline(hCtx, Object, SB_YES, ESB_YES, hOutline)
    If sts = 0 Then
        sts = EsbOtlFindMember(hOutline, "Year", "Year", ESB_MBRNAMELEN, hCtx, Object, Object.hCtx, hCtx, Object.hCtx)
    End If
    If sts = 0 Then
        Object.Name = "Year"
        Object.Type = ESB_OBJTYPE_BASICHIERARCHY
        Object.AppName = "Sample"
        Object.DbName = "Basic"
        Object.FileName = "Basic"
        sts = EsbOtlAddNode(hCtx, Object, Object.hCtx, hCtx, Object.hCtx)
    End If
End Sub
End If
If sts = 0 And hMemberYear <> 0 Then
    szAlias = "Year Dimension"
    sts = EsbOtlSetMemberAlias(hOutline, hMemberYear, ",", szAlias)
End If
End Sub

See Also
l EsbOtlGetMemberAlias
l EsbOtlDeleteMemberAlias

**EsbOtlSetMemberFormula**

Sets the formula for a specified member.

**Syntax**

```
EsbOtlSetMemberFormula (hOutline, hMember, pszFormula)
ByVal hOutline   As Long
ByVal hMember    As Long
ByVal pszFormula As String
```

**Parameter Description**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>Outline context handle.</td>
</tr>
<tr>
<td>hMember</td>
<td>Member handle.</td>
</tr>
<tr>
<td>pszFormula</td>
<td>Buffer containing the member formula.</td>
</tr>
</tbody>
</table>

**Notes**

Use `EsbOtlDeleteMemberFormula()` to remove a member formula.

**Return Value**

Returns 0 if successful; otherwise one of the following:

- `OTLAPI_ERR_SHAREDMEMBERFORMULA`
- `OTLAPI_ERR_MEMBERCALC`

**Example**

Declare Function EsbOtlSetMemberFormula Lib "ESBOTLN" (ByVal hOutline As Long, ByVal hMember As Long, ByVal pszFormula As String) As Long

Sub ESB_OtlSetMemberFormula()
    Dim sts As Long
    Dim Object As ESB_OBJDEF_T
    Dim hOutline As Long
    Dim hMemberProfit As Long
    Dim szFormula as String * 100
    Object.hCtx = hCtx
Object.Type = ESB_OBJTYPE_OUTLINE
Object.AppName = "Sample"
Object.DbName = "Basic"
Object.FileName = "Basic"

sts = EsbOtlOpenOutline(hCtx, Object,
ESB_YES, ESB_YES, hOutline)
If sts = 0 Then
    sts = EsbOtlFindMember(hOutline, "Profit",
hMemberProfit)
End If
If sts = 0 hMemberProfit <> 0 Then
    szFormula = "Profit = Gross / Margin;"
    sts = EsbOtlSetMemberFormula(hOutline,
        hMemberProfit, szFormula)
End If
End Sub

See Also

● EsbOtlGetMemberFormula
● EsbOtlDeleteMemberFormula

EsbOtlSetMemberInfo

Sets member attribute information.

Syntax

EsbOtlSetMemberInfo (hOutline, hMember, pInfo)
ByVal hOutline As Long
ByVal hMember As Long
    pInfo As ESB_MBRINFO_T

Parameter Description

hOutline    Outline context handle.
hMember     Handle to member to set attributes for.
pInfo       Member information structure.

Notes

● EsbOtlGetMemberInfo() should be called to initialize the fields of the ESB_MBRINFO_T structure.
● Attributes:
    ○ Two fields of the ESB_MBRINFO_T structure are for attributes only:
### Data Type | Field | Description
--- | --- | ---
As Variant | Attribute | Attribute value: For an attribute dimension or zero-level (leaf node) attribute member, one of the following data types:
- ESB_ATTRMBRDT_BOOL
- ESB_ATTRMBRDT_DATETIME
- ESB_ATTRMBRDT_DOUBLE
- ESB_ATTRMBRDT_STRING

For any attribute member, but not an attribute dimension:
- ESB_ATTRMBRDT_NONE
- ESB_ATTRMBRDT_AUTO

Use ESB_ATTRMBRDT_AUTO only when adding a member. See Notes on Adding an Attribute Member.

As Integer | IsAttribute | Indicates whether the member has attributes associated with it.

Values for two fields of the ESB_MBRINFO_T structure are for attributes only:

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
</table>
| As Integer | usCategory | One of the following dimension categories:  
- ESB_CAT_ATTRIBUTE  
- ESB_CAT_ATTRCALC (for internal use only) |
| As Integer | usStorageCategory | One of the following dimension storage categories:  
- ESB_STORECAT_ATTRIBUTE  
- ESB_STORECAT_ATTRCALC (for internal use only) |

#### Return Value
Returns 0 if successful; otherwise one of the following:
- OTLAPI_BAD_CONSOL
- OTLAPI_BAD_MBRNAME
- OTLAPI_BAD_MEMBER
- OTLAPI_ERR_ADDNAMEUSED
- OTLAPI_ERR_CURTOOMANYDIMS
- OTLAPI_ERR_BADSHARE
- OTLAPI_ERR_BADSKIP
- OTLAPI_ERR_BADSTORAGE
- OTLAPI_ERR_BADSTORAGECATEGORY
- OTLAPI_ERR_BADTIMEBAL
- OTLAPI_ERR_ILLEGALBOOLEAN
- OTLAPI_ERR_ILLEGALCURRENCY
- OTLAPI_ERR_ILLEGALDATE
- OTLAPI_ERR_ILLEGALNAME
- OTLAPI_ERR_ILLEGALNUMERIC
- OTLAPI_ERR_ILLEGALTAG
- OTLAPI_ERR_LEAFLABEL
- OTLAPI_ERR_NOSHAREPROTO
- OTLAPI_ERR_NOTIMEDIM
- OTLAPI_ERR_SHARENOTLEVEL0

Example

Declare Function EsbOtlSetMemberInfo Lib "ESBOTLN" (ByVal hOutline As Long, ByVal hMember As Long, pInfo As ESB_MBRINFO_T) As Long

Sub ESB_OtlSetMemberInfo()
Dim sts As Long
Dim Object As ESB_OBJDEF_T
Dim hOutline As Long
Dim MbrInfo As ESB_MBRINFO_T
Dim hFeb As Long
Object.hCtx = hCtx
Object.Type = ESB_OBJTYPE_OUTLINE
Object.AppName = "Sample"
Object.DbName = "Basic"
Object.FileName = "Basic"
sts = EsbOtlOpenOutline(hCtx, Object, ESB_YES, ESB_YES, hOutline)
If sts = 0 Then
    sts = EsbOtlFindMember(hOutline, "Feb", hFeb)
End If
If sts = 0 And hFeb <> 0 Then
    MbrInfo.fTwoPass = ESB_TRUE
    MbrInfo.fExpense = ESB_TRUE
    MbrInfo.usTimeBalance = ESB_TIMEBAL_AVG
    MbrInfo.usSkip = ESB_SKIP_ZEROS
    MbrInfo.usConsolidation = ESB_UCALC_MULT
    sts = EsbOtlSetMemberInfo(hOutline, hFeb, MbrInfo)
End If
End Sub

See Also

- EsbOtlGetMemberInfo
- EsbOtlFindMember
**EsbOtlSetOutlineInfo**

Sets outline information.

**Syntax**

```vba
EsbOtlSetOutlineInfo (hOutline, pInfo)
ByVal hOutline As Long
    pInfo   As ESB_OUTLINEINFO_T
```

**Parameter Description**

- **hOutline**
  Outline context handle.

- **pInfo**
  Storage variable for outline information, allocated by the caller.

**Notes**

- Only some of the fields of the ESB_OUTLINEINFO_T structure are used to set information. See the "API Structures" section for more information.
- Call `EsbOtlGetOutlineInfo()` to initialize the fields in the ESB_OUTLINEINFO_T structure.
- If the `fCaseSensitive` flag in the ESB_OUTLINEINFO_T structure is changed from ESB_TRUE to ESB_FALSE, and this causes duplicate member names, the call fails.

**Return Value**

Returns 0 if successful; otherwise one of the following:

- OTLAPI_BAD_OUTLINETYPE
- OTLAPI_ERR_DUPLICATEALIAS
- OTLAPI_ERR_CURTOOMANYDIMS
- OTLAPI_ERR_ILLEGALTAG
- OTLAPI_ERR_DUPLICATENAME

**Example**

```vba
Declare Function EsbOtlSetOutlineInfo Lib "ESBOTLN" (ByVal hOutline As Long,
pInfo As ESB_OUTLINEINFO_T) As Long

Sub ESB_OtlSetOutlineInfo()
    Dim sts As Long
    Dim Object As ESB_OBJDEF_T
    Dim hOutline As Long
    Dim Info As ESB_OUTLINEINFO_T
    Object.hCtx = hCtx
    Object.Type = ESB_OBJTYPE_OUTLINE
    Object.AppName = "Sample"
    Object.DbName = "Basic"
    Object.FileName = "Basic"
    sts = EsbOtlOpenOutline(hCtx, Object, ESB_YES, hOutline)
    'call GetOutlineInfo() to fill structure
    If sts = 0 Then
```
Info.fCaseSensitive = ESB_FALSE
sts = EsbOtlSetOutlineInfo(hOutline, Info)
End If
End Sub

See Also

● EsbOtlGetOutlineInfo

**EsbOtlSetUserAttribute**

Sets a user-defined attribute for a member.

**Syntax**

```plaintext
EsbOtlSetUserAttribute (hOutline, hMember, pszString)
```

**Parameter Description**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hOutline</td>
<td>Outline context handle.</td>
</tr>
<tr>
<td>hMember</td>
<td>Handle of member for which to set the user-defined attribute.</td>
</tr>
<tr>
<td>pszString</td>
<td>User-defined attribute to set.</td>
</tr>
</tbody>
</table>

**Notes**

- A caller can set any number of user-defined attributes for a member. The string passed in uniquely defines each attribute and follows the same conventions as user names. See EsbOtlGetUserAttributes().
- Attempting to set a user attribute for a shared member generates an error.

**Return Value**

Returns 0 if successful; otherwise one of the following:

- OTLAPI_BAD_USERATTR
- OTLAPI_ERR_SHAREUDA

**Example**

```vbnet
Declare Function EsbOtlSetUserAttribute Lib "ESBOTLN" (ByVal hOutline As Long, ByVal hMember As Long, ByVal pszString As String) As Long

Sub ESB_OtlSetUserAttribute()
    Dim sts As Long
    Dim Object As ESB_OBJDEF_T
    Dim hOutline As Long
    Dim hMember As Long
    Dim AttributeList As String
    Object.hCtx = hCtx
    Object.Type = ESB_OBJTYPE_OUTLINE
```

Visual Basic Outline API Function Reference 1573
Object.AppName = "Sample"
Object.DbName = "Basic"
Object.FileName = "Basic"
AttributeList = "Read Write"
sts = EsbOtlOpenOutline(hCtx, Object, ESB_YES,
ESB_YES, hOutline)
If sts = 0 Then
    sts = EsbOtlFindMember(hOutline, "Jan",
    hMember)
End If
If sts = 0 And hMember <> 0 Then
    '********************
    ' Set User Attributes
    '********************
    sts = EsbOtlSetUserAttribute(hOutline,
    hMember, AttributeList)
End If
End Sub

See Also

- EsbOtlDeleteUserAttribute
- EsbOtlGetUserAttributes

**EsbOtlSortChildren**

Sorts the children of an outline member.

**Syntax**

```vbnet
EsbOtlSortChildren (hOutline, hParent, usType)
ByVal hOutline As Long
ByVal hParent As Long
ByVal usType As Integer
```

**Parameter Description**

- **hOutline** Outline context handle.
- **hParent** Handle of parent of the children to sort. If this is ESB_NULL, the dimensions are sorted.
- **usType** Sort type. This can be one of the following:
  - ESB_SORT_ASCENDING
  - ESB_SORT_DESCENDING

**Return Value**

Returns 0 if successful; otherwise one of the following:

- OTLAPI_BAD_SORTTYPE
- OTLAPI_BAD_SORTCOMPAREFUNC
- OTLAPI_SORT_TOOMANY
Example

Declare Function EsbOtlSortChildren Lib "ESBOTLW" (ByVal hOutline As Long, ByVal hParent As Long, ByVal usType As Integer) As Long
Sub ESB_OtlSortChildren()
  Dim sts As Long
  Dim Object As ESB_OBJDEF_T
  Dim hOutline As Long
  Dim hParent As Long
  Object.hCtx = hCtx
  Object.Type = ESB_OBJTYPE_OUTLINE
  Object.AppName = "Sample"
  Object.DbName = "Basic"
  Object.FileName = "Basic"
  sts = EsbOtlOpenOutline(hCtx, Object, ESB_YES, ESB_YES, hOutline)
  If sts = 0 Then
    sts = EsbOtlFindMember(hOutline, "Profit", hParent)
  End If
  If sts = 0 And hParent <> 0 Then
    sts = EsbOtlSortChildren(hOutline, hParent, ESB_SORT_DESCENDING)
  End If
End Sub

See Also

- EsbOtlFindMember

EsbOtlVerifyOutline

Verifies that an outline is correct. Returns both global outline errors and errors for each incorrect member.

Syntax

EsbOtlVerifyOutline (hOutline, pulErrors, pulCount)

ByVal hOutline As Long
  pulErrors As Long
  pulCount As Long

Parameter Description

hOutline  Outline context handle.

pulErrors  Return variable representing the bitmask for return of global outline errors. Currently, this field has only one value: ESB_OUTERROR_CURTOOMANYDIMS

pulCount  Count of members with errors.

Notes

- This function checks for duplicate user attributes in shared members and duplicate level or generation names or aliases.
Saving the outline to the server succeeds only when the outline is free of errors (\textquote{pulErrors} == 0 and \textquote{pulCount} == 0).

To retrieve error values:

1. Allocate an \texttt{ESB\_OUTERROR\_T} structure.
2. Call \texttt{EsbGetNextItem()} once for each member error (returned in the \textquote{pulCount} variable).
   
   Each call to \texttt{EsbGetNextItem()} returns the error information for a member in an \texttt{ESB\_OUTERROR\_T} structure.

Return Value

Returns 0 if successful; otherwise one of the following:

- \texttt{ESB\_OUTERROR\_SHAREUDA}
- \texttt{ESB\_OUTERROR\_DUPGENLEVNAME}

Example

Declare Function EsbOt1VerifyOutline Lib "ESBOTLN" (ByVal hOutline As Long, pulErrors As Long, pulCount As Long) As Long

Sub ESB\_Ot1VerifyOutline()
  Dim sts As Long
  Dim Object As ESB\_OBJDEF\_T
  Dim hOutline As Long
  Dim ulErrors As Long
  Dim ulCount As Long
  Dim pOutError As ESB\_OUTERROR\_T
  Object.hCtx = hCtx
  Object.Type = ESB\_OBJTYPE\_OUTLINE
  Object.AppName = "Sample"
  Object.DbName = "Basic"
  Object.FileName = "Basic"
  sts = EsbOt1OpenOutline(hCtx, Object, ESB\_YES, ESB\_YES, hOutline)
  'body of code
  If sts = 0 Then
    sts = EsbOt1VerifyOutline(hOutline, ulErrors, ulCount)
    Do While sts = 0 And ulCount > 0
      sts = EsbGetNextItem(hCtx, ESB\_OUTERROR\_TYPE, pOutError)
      ulCount = ulCount - 1
      'do something with the error value
    Loop
  End If
End Sub

See Also

- \texttt{EsbOt1NewOutline}
- \texttt{EsbOt1OpenOutline}
- \texttt{EsbOt1WriteOutline}
EsbOtlWriteOutline

Writes the existing outline information to disk.

Syntax

EsbOtlWriteOutline (hOutline, pObject)
(ByVal hOutline As Long
    pObject As ESB_OBJDEF_T)

Parameter Description

hOutline   Outline context handle.
pObject    Outline object to write.

Notes

- If you are saving the outline as a server object, it is initially saved as an .OTN file. You should then call EsbOtlRestructure() to create the actual .OTL file.
- If you are saving the outline as a server object, the object name must be the same as the database name.
- The database must already exist if you are saving a server outline object, or a client outline object to a local database.
- This call fails if the outline is not currently locked by the specified user (hCtx parameter in the ESB_OBJDEF_T structure).

Return Value

Returns 0 if successful; otherwise one of the following:

- OTLAPI_BAD_OBJTYPE
- OTLAPI_ERR_NOTVERIFIED

Access

This function requires you to have the appropriate level of access to the specified application and/or database to contain the outline object. To write the outline object, you must have Application Designer or Database Designer privilege (ESB_PRIV_APPDESIGN or ESB_PRIV_DBDESIGN) for the specified application or database containing the outline.

Example

Declare Function EsbOtlWriteOutline Lib "ESBOTLN" (ByVal hOutline As Long,
pObject As ESB_OBJDEF_T) As Long

Sub ESB_OtlWriteOutline()
    Dim sts As Long
    Dim Object As ESB_OBJDEF_T
    Dim hOutline As Long
    Object.hCtx = hCtx
    Object.Type = ESB_OBJTYPE_OUTLINE
    Object.AppName = "Sample"
    Object.DbName = "Basic"
Object.FileName = "Basic"
sts = EsbOtlOpenOutline(hCtx, Object, ESB_YES, ESB_YES, hOutline)
'body of code
If sts = 0 Then
  sts = EsbOtlWriteOutline(hOutline, Object)
End If
'restructure db using EsbOtlRestructure()
End Sub

See Also

- EsbOtlOpenOutline
- EsbOtlNewOutline
- EsbOtlVerifyOutline
- EsbOtlRestructure
- EsbOtlCloseOutline
This example demonstrates the use of the outline tree. TraverseTree is a recursive algorithm that traverses the outline tree to provide access to all outline members. It selects each member in turn, allowing processing on each, until it reaches the last member. A comment in the code notes the opportunity for added processing.

This algorithm incorporates several VB Outline API commands.

- **EsbOtlGetFirstMember()** returns a member handle to the first member (the first dimension defined) in the outline.
- **EsbOtlGetMemberInfo()** gets information for the specified member.
- **EsbOtlGetChild()** returns the child of a member.
- **EsbOtlGetNextSibling()** returns the next sibling of a member.

Before executing this code, initialize the API and open the outline. Following this code, close the outline and terminate the API.

```c
TraverseTree (ESB_HOUTLINE_T)
{
    ESB_HMEMBER_T hMember;
    ESB_STS_T  sts = 0;

    sts = EsbOtlGetFirstMember(hOutline, &hMember);
    if (!sts && hMember)
        sts = TraverseTreeRecurse(hOutline, hMember);
}

TraverseTreeRecurse(ESB_HOUTLINE_T hOutline, ESB_HMEMBER_T hMember)
{
    ESB_MEMBERINFO_T MbrInfo;
    ESB_HMEMBER_T, hChild;
    ESB_STS_T  sts = 0;

    while (!sts && hMember)
    {
        sts = EsbOtlGetMemberInfo (hOutline, hMember, &MbrInfo);

        /* ADD THE PROCESSING FOR EACH MEMBER HERE. */

        if (!sts)
        {
            sts = EsbOtlGetChild(hOutline, hMember, &hChild);
            if (!sts && hChild)
            {
```
sts = TraverseTreeRecurse(hOutline, hChild);
}
}
sts = EsbOtlGetNextSibling(hOutline, hMember, &hMember);
}
return (sts);
In Other APIs:

- Java API Reference
- MDX Provider API
- Welcome to XMLA Reference
- Working with XMLA
Java API Reference

The Java API documentation is included as Javadocs in the Oracle Hyperion Provider Services installation. Please refer to the Javadocs in ESSBASEPATH\aps\docs.
In This Chapter

MDX Provider API General Information............................................................... 1585
MDX Provider API Reference.......................................................................... 1587
MDX Sample Client Program.......................................................................... 1606

MDX Provider API General Information

MDX queries that follow the grammar given in the MDX functional specification can be submitted to the server through the MDX-API. The query results can then be retrieved by the client using this API.

The grammar of MDX statements is covered in the MDX section of the Oracle Essbase Technical Reference.

A few basic MDX concepts and terminology are reviewed here. An MDX query consists of several axis specifications, and an optional slicer specification. Each axis specifies a set-valued expression. A set is an ordered collection of tuples, with tuples being a sequence of members from one or more dimensions. Tuples in a set are homogeneous in dimensionality (each tuple has members from the same dimensions in the same order).

An example of a set expression based on the Sample Basic database is:

```mdx
Union(
    CrossJoin({[Sales], [Profit]}, {[Actual], [Budget]}),
    Union(
        CrossJoin([Total Expenses].Children, {[Actual]}),
        {([Opening Inventory], [Variance]), ([Additions], [Variance %])}
    )
)
```

This expression uses several MDX functions: Union, CrossJoin, Children. The value of this expression is the set:

```
{([Sales], [Actual]),
([Sales], [Budget]),
([Profit], [Actual]),
([Profit], [Budget]),
([Marketing], [Actual]),
([Payroll], [Actual]),
([Misc], [Actual]),
([Opening Inventory], [Variance]),
```
Note that in the result of the CrossJoin, the tuples are ordered so that the first dimension changes slowest. The tuples in this set have the dimensionality: ([Measures],[Scenario]). The dimensionality of tuples across axis sets must not overlap.

In addition to the set expression, each axis specifies the name of the axis (COLUMNS, ROWS, PAGES, etc.) or the axis number (AXIS(0), AXIS(1), etc.). The cube consisting of all possible combinations of tuples, one from each axis, constitutes the result of the query. Dimensions that are not present in any axis and in the slicer default to having their root member included in defining the result cube. The slicer, if present, specifies a set, with a single tuple, which identifies the members of interest along the respective dimensions. This makes the final result a slice of the cube created from the axes. The result of an MDX query contains the metadata about each axis and the slicer, as well as the data values in the cells in the result cube.

Here is a complete MDX query:

```mdx
SELECT
  UNION(
    CROSSJOIN({[Sales], [Profit]}, {[Actual], [Budget]}),
    UNION(
      CROSSJOIN([Total Expenses].Children, {[Actual]}),
      {[Opening Inventory], [Variance]}, {[Additions], [Variance %]})
) ON COLUMNS,
  CROSSJOIN([200].Children, {[East], [West]}) ON ROWS
FROM Sample.Basic
WHERE {[Jan]}
```

The result of this query has 9 tuples on the column axis and 8 tuples on the row axis, which means there are 72 cells in all. Each cell has an ordinal, or offset, which depends on the position of its tuples along each axis. Offsets and positions start at 0. The cells are ordered so that the first axis position changes the fastest.

For example, the cell identified by tuple 3 in the column axis and tuple 4 in the row axis is at offset 3 + 9*4 = 39.

- Tuple 3 in the column axis is ([Profit], [Budget]).
- Tuple 4 in the row axis is ([200-30], [East]).
- Cell 39 is therefore ([Profit], [Budget], [200-30], [East], [Jan]).

The concept of clusters is needed for reasons of efficiency. A set can be considered to be an ordered collection of tuples, or it can be considered to be an ordered collection of clusters. A cluster is a collection of tuples that involve all possible combinations of certain members from each of the set's dimensions. The tuples need to ordered in the same manner as in the output of the CrossJoin function (the first dimension changes the slowest). Use of the CrossJoin function
causes clusters to be created, but the server may determine clusters from the results of other functions as well.

**MDX Provider API Reference**

The C API for MDX query processing is designed to fit in with the existing Essbase APIs. Client programs are given handles to various structures internal to the API, and use methods to access their components. The number of functions is kept small by judicious combining of output results normally needed together. Except where noted, memory allocated by the API for its internal structures is freed when the client invokes the query free function. ESS_MDX is the prefix used for the handle types, and EssMdx is the prefix used for the functions introduced by the MDX-API.

**MDX Provider Declarations**

The type definitions are as follows:

```c
typedef void *ESS_MDX_QRYHDL_T; /* MDX query handle */
typedef unsigned long ESS_MDX_MEMBERIDTYPE_T; /* MDX mbr id type */
typedef void *ESS_MDX_AXISHDL_T; /* MDX axis handle */
typedef void *ESS_MDX_DIMHDL_T; /* MDX dim handle */
typedef unsigned long ESS_MDX_PROPTYPE_T; /* MDX property type */
typedef void *ESS_MDX_PROPHDL_T; /* MDX property handle */
typedef void *ESS_MDX_CLUSTERHDL_T; /* MDX cluster handle */
typedef void *ESS_MDX_MBRHDL_T; /* MDX mbr handle */
typedef void *ESS_MDX_CELLHDL_T; /* MDX cell handle */
typedef unsigned long ESS_MDX_CELLSTATUS_T; /* MDX cell status */
```

The constant definitions are as follows:

```c
/* MDX member identifier types (ESS_MDX_MEMBERIDTYPE_T) */
#define ESS_MDX_MEMBERIDTYPE_NAME 8
#define ESS_MDX_MEMBERIDTYPE_ALIAS 16

/* MDX property value types (ESS_MDX_PROPTYPE_T) */
#define ESS_MDX_PROPTYPE_BOOL ESS_DT_BOOL
#define ESS_MDX_PROPTYPE_DOUBLE ESS_DT_DOUBLE
#define ESS_MDX_PROPTYPE_DATETIME ESS_DT_DATETIME
#define ESS_MDX_PROPTYPE_STRING ESS_DT_STRING
#define ESS_MDX_PROPTYPE_ULONG ESS_DT_ULONG
#define ESS_MDX_PROPTYPE_NONE 0

/* MDX cell status bitmasks (ESS_MDX_CELLSTATUS_T) */
#define ESS_MDX_CELLSTATUS_LINKEDOBJS 0x00000001
#define ESS_MDX_CELLSTATUS_DYNCALC 0x00000002
#define ESS_MDX_CELLSTATUS_CALCEDMBR 0x00000004
#define ESS_MDX_CELLSTATUS_READONLY 0x00000008

/* MDX cell property bitmasks (ESS_MDX_CELLPROP_T) */
#define ESS_MDX_CELLPROP_GLDRILLTHRU 0x00000008
```

ESS_MDX_PROPVALUE_T
typedef struct ess_mdx_propvalue_t
{
    ESS_MDX_PROPTYPE_T ulPropType;  /* ESS_MDX_PROPTYPE_XXXX */
    union
    {
        ESS_BOOL_T bData;       /* Boolean value */
        ESS_ULONG_T ulData;     /* Ulong value */
        ESS_STR_T strData;      /* String value */
        ESS_DATETIME_T dtData;  /* Datetime value */
        ESS_DOUBLE_T dblData;   /* Double value */
    } value;
} ESS_MDX_PROPVALUE_T;

ESS_MDX_CELLVALUE_T
typedef struct mdxcellvalue
{
    ESS_DOUBLE_T       dblVal;
    ESS_STR_T          fmtVal;
    ESS_STR_T          fmtStr;
    ESS_USHORT_T       smId;
    ESS_USHORT_T       type;
    ESS_ULONG_T        flags;     // captures drill through property.
} ESS_MDX_CELLVALUE_T;

EssMdxExecuteQuery

Executes the specified query on the currently connected database.

Syntax

ESS_FUNC_M EssMdxExecuteQuery(
    ESS_MDX_QRYHDL_T   hQry);

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hQry</td>
<td>input</td>
<td>Query handle</td>
</tr>
</tbody>
</table>

Notes

Before calling this function, you must first create an MDX query by calling EssMDXNewQuery.

EssMdxFreeQuery

Frees memory used for the specified query.

Syntax

ESS_FUNC_M EssMdxFreeQuery(
    ESS_MDX_QRYHDL_T   hQry);
Parameters | Type | Description
--- | --- | ---
hQry | input | Query handle

**EssMdxGetAxes**

Returns information about the axes in the query.

To obtain information on the nature of the axes in the submitted query, use the following APIs after calling “EssMdxExecuteQuery” on page 1588:

- “EssMdxGetAxes” on page 1589
- “EssMdxGetAxisInfo” on page 1589
- “EssMdxGetDimInfo” on page 1595

**Syntax**

```c
ESS_FUNC_M EssMdxGetAxes(
    ESS_MDX_QRYHDL_T hQry,
    ESS_PULONG_T pulNAxes,
    ESS_MDX_PPAXISHDL_T pphAxes,
    ESS_MDX_PAXISHDL_T phSlicer);
```

Parameters | Type | Description
--- | --- | ---
hQry | input | Query handle
pulNAxes | output | Number of axes
pphAxes | output | Array of axis handles
phSlicer | output | Slicer axis handle

**EssMdxGetAxisInfo**

Returns information about the specified axis.

To obtain information on the nature of the axes in the submitted query, use the following APIs after calling “EssMdxExecuteQuery” on page 1588:

- “EssMdxGetAxes” on page 1589
- “EssMdxGetAxisInfo” on page 1589
- “EssMdxGetDimInfo” on page 1595

**Syntax**

```c
ESS_FUNC_M EssMdxGetAxisInfo(
    ESS_MDX_AXISHDL_T hAxis,
    ESS_PULONG_T pulSize,
    ESS_PULONG_T pulNDims,
    ESS_MDX_PPDIMHDL_T pphDims);
```
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hAxis</td>
<td>input</td>
<td>Axis handle</td>
</tr>
<tr>
<td>pulSize</td>
<td>output</td>
<td>Number of tuples in axis</td>
</tr>
<tr>
<td>pulNDims</td>
<td>output</td>
<td>Number of dimensions in axis</td>
</tr>
<tr>
<td>pphDims</td>
<td>output</td>
<td>Array of dimension handles</td>
</tr>
</tbody>
</table>

**EssMdxGetAxisMembers**

Returns the tuple at the specified position in the given axis. Use this function to directly retrieve a particular tuple from an axis.

**Note:** The client should use **EssFree()** when done with pphMbrs.

To obtain information about the contents of an axis set, use the following APIs:

- “EssMdxGetClusters” on page 1594
- “EssMdxGetClusterInfo” on page 1593
- “EssMdxGetClusterMembers” on page 1594
- “EssMdxGetAxisMembers” on page 1590
- “EssMdxGetMbrIdentifier” on page 1597
- “EssMdxGetMbrProperty” on page 1597

**Syntax**

```c
ESS_FUNC_M EssMdxGetAxisMembers(
    ESS_MDX_AXISHDL_T    hAxis,
    ESS_ULONG_T          ulIndex,
    ESS_MDX_PPMBRHDL_T   pphMbrs);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hAxis</td>
<td>input</td>
<td>Axis handle</td>
</tr>
<tr>
<td>ulIndex</td>
<td>input</td>
<td>Tuple position within axis</td>
</tr>
<tr>
<td>pphMbrs</td>
<td>output</td>
<td>Array of member handles for tuple</td>
</tr>
</tbody>
</table>

**EssMdxGetCellAtIndices**

Returns the cell at the intersection of the specified tuple indices.

To obtain the cell values in the cube formed from the axes in the query, use the following APIs:

- “EssMdxGetCellAtOffset” on page 1591
**EssMdxGetCellAtIndices**

Syntax

```c
ESS_FUNC_M EssMdxGetCellAtIndices(
    ESS_MDX_QRYHDL_T hQry,
    ESS_PULONG_T pulIndices,
    ESS_MDX_PCELLHDL_T phCell);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hQry</td>
<td>input</td>
<td>Query handle</td>
</tr>
<tr>
<td>pulIndices</td>
<td>input</td>
<td>Tuple indices, one for each axis</td>
</tr>
<tr>
<td>phCell</td>
<td>output</td>
<td>Cell handle</td>
</tr>
</tbody>
</table>

**EssMdxGetCellAtOffset**

Returns the cell at the specified offset.

To obtain the cell values in the cube formed from the axes in the query, use the following APIs:

- “EssMdxGetCellAtOffset” on page 1591
- “EssMdxGetCellAtIndices” on page 1590
- “EssMdxGetValue” on page 1601

Syntax

```c
ESS_FUNC_M EssMdxGetCellAtOffset(
    ESS_MDX_QRYHDL_T hQry,
    ESS_ULONG_T ulOffset,
    ESS_MDX_PCELLHDL_T phCell);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hQry</td>
<td>input</td>
<td>Query handle</td>
</tr>
<tr>
<td>ulOffset</td>
<td>input</td>
<td>Cell offset (first axis changes fastest)</td>
</tr>
<tr>
<td>phCell</td>
<td>input</td>
<td>Cell handle</td>
</tr>
</tbody>
</table>

**EssMdxGetCellInfo**

Returns the type of the cell corresponding to the input cell handle.

Syntax

```c
ESS_FUNC_M EssMdxGetCellInfo (
    ESS_MDX_CELLHDL_T hCell,
```
ESS_PULONG_T pulType,
ESS_MDX_PCELLINFO_T pulCellInfo,
ESS_MDX_PCELLSTATUS_T pulStatus);

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCell</td>
<td>input</td>
<td>Cell handle</td>
</tr>
<tr>
<td>pulType</td>
<td>output</td>
<td>Cell data type. Values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_MDX_VALTYPE_DOUBLE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Numeric type</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_MDX_VALTYPE_SMARTLIST</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Smartlist type</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_MDX_VALTYPE_DATE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Date type</td>
</tr>
<tr>
<td>pulCellInfo</td>
<td>output</td>
<td>Cell status bit map specified using the following bitmasks:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_MDX_CELLINFO_MISSING</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The cell value is missing</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_MDX_CELLINFO_NOACCESS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The cell value is not accessible to the current user.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_MDX_CELLINFO_MEANINGLESS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The cell value is meaningless in the context of attribute members</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ESS_MDX_CELLINFO_OUTOFRANGE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The cell value is out of range in the context of a smartlist</td>
</tr>
<tr>
<td>pulStatus</td>
<td>output</td>
<td>Cell status information. This is the same information returned by the EssMdxGetCellStatus function; see the function description for more information. The status information is returned only if the function EssMdxSetNeedCellStatus is called.</td>
</tr>
</tbody>
</table>

**EssMdxGetCellStatus**

Returns the status of the cell specified by *hCell*. The status can be tested against the bitmasks in *pulStatus* to determine whether the cell is of the corresponding type. This function should be called only after an earlier call to "EssMdxSetNeedCellStatus” on page 1604.

**Syntax**

```c
ESS_FUNC_M EssMdxGetCellStatus(
    ESS_MDX_QRYHDL_T hQry,
    ESS_MDX_CELLHDL_T hCell,
    ESS_MDX_PCELLSTATUS_T pulStatus);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hQry</td>
<td>input</td>
<td>Query handle</td>
</tr>
<tr>
<td>hCell</td>
<td>input</td>
<td>Cell handle</td>
</tr>
<tr>
<td>Parameter</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>---------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>pulStatus</td>
<td>output</td>
<td>Cell status:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Bitmap with the following masks:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESS_MDX_CELLSTATUS_LINKEDOBS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESS_MDX_CELLSTATUS_DYNCALC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESS_MDX_CELLSTATUS_CALCED</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESS_MDX_CELLSTATUS_READONLYMBR</td>
</tr>
</tbody>
</table>

**EssMdxGetClusterDimMembers**

Returns the member handles for the specified dimension within the given cluster.

**Syntax**

```c
ESS_FUNC_M EssMdxGetClusterDimMembers(
    ESS_MDX_CLUSTERHDL_T   hCluster,
    ESS_ULONG_T            ulIndex,
    ESS_MDX_PPMBRHDL_T     pphMbrs);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCluster</td>
<td>input</td>
<td>Cluster handle</td>
</tr>
<tr>
<td>ulIndex</td>
<td>input</td>
<td>Dimension index within axis containing cluster</td>
</tr>
<tr>
<td>pphMbrs</td>
<td>output</td>
<td>Array of member handles for the specified dimension</td>
</tr>
</tbody>
</table>

**EssMdxGetClusterInfo**

Returns information about the specified cluster.

To obtain information about the contents of an axis set, use the following APIs:

- “EssMdxGetClusters” on page 1594
- “EssMdxGetClusterInfo” on page 1593
- “EssMdxGetClusterMembers” on page 1594
- “EssMdxGetAxisMembers” on page 1590
- “EssMdxGetMbrIdentifier” on page 1597
- “EssMdxGetMbrProperty” on page 1597

**Syntax**

```c
ESS_FUNC_M EssMdxGetClusterInfo(
    ESS_MDX_CLUSTERHDL_T   hCluster,
    ESS_PULONG_T           pulSize,
    ESS_PULONG_T           pulNDims,
    ESS_PPULONG_T          ppulDimSizes);
```
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCluster</td>
<td>input</td>
<td>Cluster handle</td>
</tr>
<tr>
<td>pulSize</td>
<td>output</td>
<td>Number of tuples in cluster</td>
</tr>
<tr>
<td>pulNDims</td>
<td>output</td>
<td>Number of dimensions in cluster (same as that in the axis that this cluster belongs to)</td>
</tr>
<tr>
<td>ppulDimSizes</td>
<td>output</td>
<td>Array of dimension sizes (number of members)</td>
</tr>
</tbody>
</table>

**EssMdxGetClusterMembers**

Returns the tuple at the specified position within the given cluster.

**Note:** The client should use `EssFree()` when done with `pphMbrs`.

To obtain information about the contents of an axis set, use the following APIs:

- “EssMdxGetClusters” on page 1594
- “EssMdxGetClusterInfo” on page 1593
- “EssMdxGetClusterMembers” on page 1594
- “EssMdxGetAxisMembers” on page 1590
- “EssMdxGetMbrIdentifier” on page 1597
- “EssMdxGetMbrProperty” on page 1597

**Syntax**

```c
ESS_FUNC_M EssMdxGetClusterMembers(
    ESS_MDX_CLUSTERTHDL_T hCluster,
    ESS_ULONG_T ulIndex,
    ESS_MDX_PPMBRHDL_T pphMbrs);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCluster</td>
<td>input</td>
<td>Cluster handle</td>
</tr>
<tr>
<td>ulIndex</td>
<td>input</td>
<td>Tuple position within cluster (first dimension changes slowest)</td>
</tr>
<tr>
<td>pphMbrs</td>
<td>output</td>
<td>Array of member handles for the tuple</td>
</tr>
</tbody>
</table>

**EssMdxGetClusters**

Returns the clusters within the specified axis.

To obtain information about the contents of an axis set, use the following APIs:

- “EssMdxGetClusters” on page 1594
EssMdxGetClusterInfo

Returns information about the specified dimension, including the properties available for members in this dimension.

Syntax

```c
ESS_FUNC_M EssMdxGetClusters(
    ESS_MDX_AXISHDL_T        hAxis,
    ESS_PULONG_T             pulNClusters,
    ESS_MDX_PPCLUSTERHDL_T   pphClusters);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hAxis</td>
<td>input</td>
<td>Axis handle</td>
</tr>
<tr>
<td>pulNClusters</td>
<td>output</td>
<td>Number of clusters</td>
</tr>
<tr>
<td>pphClusters</td>
<td>output</td>
<td>Array of cluster handles</td>
</tr>
</tbody>
</table>

EssMdxGetDimInfo

Returns information about the specified dimension, including the properties available for members in this dimension.

Syntax

```c
ESS_FUNC_M EssMdxGetDimInfo(
    ESS_MDX_DIMHDL_T      hDim,
    ESS_PSTR_T            ppszName,
    ESS_PULONG_T          pulNProps,
    ESS_MDX_PPPROPHDL_T   pphProps);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hDim</td>
<td>input</td>
<td>Dimension handle</td>
</tr>
<tr>
<td>ppszDimName</td>
<td>output</td>
<td>Dimension name</td>
</tr>
<tr>
<td>pulNProps</td>
<td>output</td>
<td>Number of properties returned</td>
</tr>
<tr>
<td>pphProps</td>
<td>output</td>
<td>Array of property handles</td>
</tr>
</tbody>
</table>

Notes

- Before calling this query, you should call “EssMdxGetAxisInfo” on page 1589 to get dimensions represented on an axis.
- To get the properties of a dimension:
  1. Call “EssMdxNewQuery” on page 1603 to create a query.
2. Call “EssMdxEvaluateQuery” on page 1588 to execute the query.

3. Call “EssMdGetXAxes” on page 1589 to get the number of axes and the individual axis handles from the result of the query.

4. Call “EssMdGetXAxisInfo” on page 1589 to get information (dimensions/tuples) for an individual axis from an axis handle.

5. Call “EssMdGetXDimInfo” on page 1595 to get information for a dimension (dimension name, number of properties for this dimension, and property handles).

6. Call “EssMdGetXPropertyInfo” on page 1599 to get the dimension properties. To get properties, the MDX query in EssMdxCopy must use the DIMENSION PROPERTIES option.

**EssMdxFacetFormatString**

Returns the formatted value of the given cell.

*Note:* Returns formatted values only if the cell property option ESS_MDX_CELLPROP_FORMAT_STRING is set using EssMdxCopyQueryCellProperties.

**Syntax**

```c
ESS_FUNC_M EssMdxFacetFormatString(
    ESS_MDX_QRYHDL_T hQry,
    ESS_MDX_CELLHDL_T hCell,
    ESS_PSTR_T pFmtStr);
```

**Parameter** | **Type** | **Description**
--- | --- | ---
```
| hQry | input | Query handle |
| hCell | input | Cell handle |
| pFmtStr | output | Format string for the given cell |
```

**See Also**

“EssMdxFacetFormattedValue” on page 1596

**EssMdxFacetFormattedValue**

Returns the formatted value of the given cell.

*Note:* Returns formatted values only if the cell property option ESS_MDX_CELLPROP_FORMATTED_VALUE is set using EssMdxCopyQueryCellProperties.
Syntax

```c
ESS_FUNC_M EssMdxGetFormattedValue(
    ESS_MDX_QRYHDL_T hQry,
    ESS_MDX_CELLHDL_T hCell,
    ESS_PSTR_T pFmtVal);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hQry</td>
<td>input</td>
<td>Input query handle</td>
</tr>
<tr>
<td>hCell</td>
<td>input</td>
<td>Input cell handle</td>
</tr>
<tr>
<td>pFmtVal</td>
<td>output</td>
<td>Formatted value of the cell</td>
</tr>
</tbody>
</table>

See Also

“EssMdxGetFormatString” on page 1596

**EssMdxGetMbrIdentifier**

Returns the identifier for the specified member.

Syntax

```c
ESS_FUNC_M EssMdxGetMbrIdentifier(
    ESS_MDX_MBRHDL_T hMbr,
    ESS_PSTR_T ppszIdentifier);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hMbr</td>
<td>input</td>
<td>Member handle</td>
</tr>
<tr>
<td>ppszIdentifier</td>
<td>output</td>
<td>Member identifier (name or alias)</td>
</tr>
</tbody>
</table>

**EssMdxGetMbrProperty**

Returns the value of the specified property for the specified member. The property value will have a type of ESS_MDX_PROPTYPE_NONE if the property is not applicable to the member.

Syntax

```c
ESS_FUNC_M EssMdxGetMbrProperty(
    ESS_MDX_MBRHDL_T hMbr,
    ESS_MDX_PROPHDL_T hProp,
    ESS_MDX_PROPVALUE_T pPropValue);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hMbr</td>
<td>input</td>
<td>Member handle</td>
</tr>
<tr>
<td>hProp</td>
<td>input</td>
<td>Property handle</td>
</tr>
</tbody>
</table>
### EssMdxGetNamedSets

Returns the named sets in the query.

**Syntax**

```c
ESS_FUNC_M EssMdxGetNamedSets(
    ESS_HCTX_T     hCtx,
    ESS_PULONG_T   pulCount,
    ESS_PPSTR_T    ppNames,
    ESS_PLONG_T    *ppTypes);
```

**Parameter**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>input</td>
<td>Context handle.</td>
</tr>
<tr>
<td>pulCount</td>
<td>output</td>
<td>Count of the named sets returned in the query.</td>
</tr>
<tr>
<td>ppNames</td>
<td>output</td>
<td>An array of named sets.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The memory allocated for <code>ppNames</code> should be freed using <code>EssFree()</code>.</td>
</tr>
<tr>
<td>*ppTypes</td>
<td>output</td>
<td>Pointer to the named set type: ESS_MDX_NAMEDSET_TYPE_SESSION.</td>
</tr>
</tbody>
</table>

**Return Value**

The return values are the number of named sets in `pulCount`, the named sets in `ppNames`, and the type of the named sets in `ppTypes`.

**Access**

This function requires no special privileges.

**Example**

```c
void TestGetNamedSets()
{
    ESS_STS_T sts = ESS_STS_NOERR;
    ESS_STR_T fileNames[2];
    ESS_CHAR_T qry[2][MAXQRYLEN];
    FILE *fileHandle;
    char *s;
    int length, e, i;
    ESS ULONG_T ulCount, j;
    ESS PSTR_T pNames;
    ESS PLONG_T pTypes;

    fileNames[0] = "D:\testarea\MDXAPI\query3.txt";
    fileNames[1] = "D:\testarea\MDXAPI\query4.txt";

    for(i = 0; i < 2; i++)
```

1598 MDX Provider API


defileHandle = fopen(fileNames[i], "r");
if (!fileHandle = fopen(fileNames[i], "r"))
{
    printf("\nUnable to open file: %s\n", fileNames[i]);
    return;
}
else
{
    s = qry[i];
    length = MAXQRYLEN;

    fgets(s, length, fileHandle);
    if ((e = ferror(fileHandle)) != 0)
    {
        printf("fgets error %d\n", e);
        exit((int) e);
    }
    fclose(fileHandle);
    printf("\nThe query[%d]: \n%s\n", i, qry[i]);
}

ulCount = 0;
sts = EssMdxGetNamedSets(hCtx, &ulCount, &pNames, &pTypes);
printf("EssMdxGetNamedSets sts: %ld\n", sts);
for(j = 0; j < ulCount; j++)
{
    printf("\tpNames[%d]: %s\n", j, pNames[j]);
    printf("\tpTypes[%d]: %d\n", j, pTypes[j]);
    printf("\n");
}

sts = EssFree(hInst, (ESS_PVOID_T)pNames);


**EssMdxGetPropertyInfo**

Returns information about the specified property.

**Syntax**

ESS_FUNC_M EssMdxGetPropertyInfo(
    ESS_MDX_PROPHDL_T     hProp,
    ESS_PSTR_T            ppszName,
    ESS_MDX_PPROPTYPE_T   pPropType);

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hProp</td>
<td>input</td>
<td>Property handle</td>
</tr>
<tr>
<td>ppszName</td>
<td>output</td>
<td>Property name</td>
</tr>
<tr>
<td>Parameter</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>-------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>pPropType</td>
<td>output</td>
<td>Property type:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESS_MDX_PROPTYPE_BOOL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESS_MDX_PROPTYPE_DOUBLE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESS_MDX_PROPTYPE_STRING</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESS_MDX_PROPTYPE_DATETIME</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ESS_MDX_PROPTYPE_ULONG</td>
</tr>
</tbody>
</table>

### EssMdxGetQueryCellProperties

Returns the cell properties in effect for this query.

**Syntax**

```c
ESS_FUNC_M    EssMdxGetQueryCellProperties(
    ESS_MDX_QRYHDL_T      hQry,
    ESS_MDX_CELLPROPS_T   pulProp);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hQry</td>
<td>input</td>
<td>Query handle</td>
</tr>
<tr>
<td>pulProp</td>
<td>output</td>
<td>Pointer to bitmask specifying what cell properties are returned</td>
</tr>
</tbody>
</table>

**See Also**

“EssMdxSetQueryCellProperties” on page 1604

### EssMdxGetQueryOptions

Returns the query options in effect for the current query.

**Syntax**

```c
ESS_FUNC_M    EssMdxGetQueryOptions(
    ESS_MDX_QRYHDL_T      hQry,
    ESS_MDX_PQRYOPT_T   pulOpt);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hQry</td>
<td>input</td>
<td>Query handle</td>
</tr>
<tr>
<td>pulOpt</td>
<td>output</td>
<td>Pointer to bitmask specifying current query options in effect</td>
</tr>
</tbody>
</table>

**See Also**

“EssMdxSetQueryOptions” on page 1605
**EssMdxGetSmartlistforCell**

Returns the name of the Smartlist object associated with a cell when the cell type is ESS_MDX_VALTYPE_SMARTLIST. An Essbase database can have multiple Smartlist objects and Smartlist members associated with these objects. This function identifies which Smartlist object a cell is associated with.

**Note:** Returns formatted values only if the cell property option ESS_MDX_CELLPROP_SMLIST_NAME is set using EssMdxSetQueryCellProperties.

**Syntax**

```c
ESS_FUNC_M EssMdxGetSmartlistforCell( 
    ESS_MDX_QRYHDL_T hQry, 
    ESS_MDX_CELLHDL_T hCell, 
    ESS_PSTR_T pSmartlist);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hQry</td>
<td>input</td>
<td>Query handle</td>
</tr>
<tr>
<td>hCell</td>
<td>input</td>
<td>Cell handle</td>
</tr>
<tr>
<td>pSmartlist</td>
<td>output</td>
<td>Name of the Smartlist object a cell is associated with</td>
</tr>
</tbody>
</table>

**EssMdxGetValue**

Returns the specified cell’s value.

To obtain the cell values in the cube formed from the axes in the query, use the following APIs:

- “EssMdxGetCellAtOffset” on page 1591
- “EssMdxGetCellAtIndices” on page 1590
- “EssMdxGetValue” on page 1601

**Syntax**

```c
ESS_FUNC_M EssMdxGetValue( 
    ESS_MDX_CELLHDL_T hCell, 
    ESS_PBOOL_T pbIsMissing, 
    ESS_PBOOL_T pbNoAccess, 
    ESS_PDOUBLE_T pdValue);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCell</td>
<td>input</td>
<td>Cell handle</td>
</tr>
<tr>
<td>pbIsMissing</td>
<td>output</td>
<td>Whether cell value is #Missing</td>
</tr>
<tr>
<td>pbNoAccess</td>
<td>output</td>
<td>Whether cell value is #NoAccess</td>
</tr>
<tr>
<td>Parameter</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>pdValue</td>
<td>output</td>
<td>The cell’s value, if not #Missing</td>
</tr>
</tbody>
</table>

**EssMDXIsCellGLDrillable**

Checks whether the cell is associated with a drill-through URL.

**Syntax**

```c
ESS_FUNC_M EssMdxIsCellGLDrillable (hQry, hCell, pIsDrillable);
```

**Parameter**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hQry</td>
<td>ESS_MDX_QRYHDL_T</td>
<td>Query handle</td>
</tr>
<tr>
<td>hCell</td>
<td>ESS_MDX_CELLHDL_T</td>
<td>Cell handle</td>
</tr>
<tr>
<td>pIsDrillable</td>
<td>ESS_PBOOL_T</td>
<td>True, if the cell is associated with a drill-through URL; False, otherwise</td>
</tr>
</tbody>
</table>

**Return Value**

- If successful, sets `pIsDrillable` based on the cell’s status.
- If unsuccessful, returns an error message.

**Example**

```c
#define ESS_MDX_CELLPROP_GLDRILLTHRU 0x00000008

if (((sts = EssMdxNewQuery(hCtx, qry, &hQry)) != ESS_STS_NOERR))
{
    printf("EssMdxNewQuery failure: %ld\n", sts);
    exit ((int) sts);
}
printf("EssMdxNewQuery sts: %ld\n", sts);
if (((sts = EssMdxSetQueryCellProperties(hQry,
                                   (ESS_MDX_CELLPROP_GLDRILLTHRU
                                   ))) != ESS_STS_NOERR))
{
    printf("EssMdxSetQueryCellProperties failure: %ld\n", sts);
    exit ((int) sts);
}
if (((sts = EssMdxExecuteQuery(hQry)) != ESS_STS_NOERR))
{
    printf("EssMdxExecuteQuery failure: %ld\n", sts);
    exit ((int) sts);
}
printf("EssMdxExecuteQuery sts: %ld\n", sts);

/* To retrieve IsCellGLDrillable property of a cell, use EssMdxIsCellGLDrillable*/
if (((sts = EssMdxIsCellGLDrillable(hQry, hCell, &bIsCellGLDT)))
```

1602 MDX Provider API
printf("EssMdxIsCellGLDrillable failure: %ld\n", sts);
    exit ((int) sts);
}
if (bIsCellGLDT)
    printf(" Is Cell Drillable: TRUE\n");
else
    printf(" Is Cell Drillable: FALSE\n");

**EssMdxNewQuery**

Takes the MDX query specified by `pszQry` and returns a query handle.

**Syntax**

```c
ESS_FUNC_M EssMdxNewQuery(
    ESS_HCTX_T          hCtx,
    ESS_STR_T           pszQry,
    ESS_MDX_PQRYHDL_T   phQry);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCtx</td>
<td>input</td>
<td>API context handle</td>
</tr>
<tr>
<td>pszQry</td>
<td>input</td>
<td>Query text</td>
</tr>
<tr>
<td>phQry</td>
<td>output</td>
<td>Query handle</td>
</tr>
</tbody>
</table>

**Notes**

This function should be called first to create any MDX query. For example, you must call this function before calling `EssMDXExecuteQuery`.

**EssMdxSetDataLess**

Turns on a query execution mode in which cell data are not retrieved. `EssMdxGetCellAtOffset()` and `EssMdxGetCellAtIndices()` should not be called for the query. The default is to retrieve cell data.

**Syntax**

```c
ESS_FUNC_M EssMdxSetDataLess(
    ESS_MDX_QRYHDL_T   hQry);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hQry</td>
<td>input</td>
<td>Query handle</td>
</tr>
</tbody>
</table>
EssMDXSetHideData

Converts #NOACCESS cells to #MISSING.

Syntax

```c
ESS_FUNC_M EssMDXSetHideData(
    ESS_MDX_QRYHDL_T  hQry);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hQry</td>
<td>input</td>
<td>Query handle</td>
</tr>
</tbody>
</table>

EssMdxSetMbrIdType

Sets the type of member identifier desired in the result. Defaults to ESS_MDX_MEMBERIDTYPE_NAME.

Syntax

```c
ESS_FUNC_M EssMdxSetMbrIdType(
    ESS_MDX_QRYHDL_T         hQry,
    ESS_MDX_MEMBERIDTYPE_T   mbrIdType);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hQry</td>
<td>input</td>
<td>Query handle</td>
</tr>
<tr>
<td>mbrIdType</td>
<td>input</td>
<td>Member identifier desired (name/alias):</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESS_MDX_MEMBERIDTYPE_NAME</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ESS_MDX_MEMBERIDTYPE_ALIAS</td>
</tr>
</tbody>
</table>

EssMdxSetNeedCellStatus

Turns on retrieval of cell status information. By default the cell status information is not retrieved.

Syntax

```c
ESS_FUNC_M EssMdxSetNeedCellStatus(
    ESS_MDX_QRYHDL_T  hQry);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hQry</td>
<td>input</td>
<td>Query handle</td>
</tr>
</tbody>
</table>

EssMdxSetQueryCellProperties

Specifies the cell properties to be sent from the server for each cell. By default, only cell value is sent. The options passed in ulProp overwrite the existing query cell properties. In other words,
if EssMdxSetQueryCellProperties is called multiple times, only the ulProp value in the last call is taken into account.

Syntax

```c
ESS_FUNC_M    EssMdxSetQueryCellProperties(
    ESS_MDX_QRYHDL_T      hQry,
    ESS_MDX_CELLPROPS_T   ulProp);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hQry</td>
<td>input</td>
<td>Query handle</td>
</tr>
<tr>
<td>ulProp</td>
<td>input</td>
<td>Bitmask specifying what cell properties should be sent. Values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ESS_MDX_CELLPROP_FORMATTED_VALUE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ESS_MDX_CELLPROP_FORMAT_STRING</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ESS_MDX_CELLPROP_SMLIST_NAME</td>
</tr>
</tbody>
</table>

See Also

“EssMdxGetQueryCellProperties” on page 1600

**EssMdxSetQueryOptions**

Sets query options based on the value of ulOpt. The options passed in ulOpt overwrite the existing query options. In other words, if EssMdxSetQueryOptions is called multiple times, only the ulOpt value in the last call is taken into account.

Syntax

```c
ESS_FUNC_M    EssMdxSetQueryOptions(
    ESS_MDX_QRYHDL_T   hQry,
    ESS_MDX_QRYOPT_T   ulOpt);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hQry</td>
<td>input</td>
<td>Query handle</td>
</tr>
<tr>
<td>ulOpt</td>
<td>input</td>
<td>Query options. Bitmask values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ESS_MDX_QRYOPT_GET_MI CELLS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This option indicates that formatted values should be generated for #Missing cells also. By default #Missing cells are not formatted by server.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ESS_MDX_QRYOPT_GET_ME CELLS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This option tells the server to distinguish #ME (meaningless value) from #Missing values. A #ME is a special case of #Missing value. It indicates that the base member and attribute member combination in the context of that cell is meaningless. By default this option is set to off.</td>
</tr>
</tbody>
</table>

See Also

“EssMdxGetQueryOptions” on page 1600
#if defined _WIN32 || defined _WINDOWS
#include <windows.h>
#endif

#include <string.h>
#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>
#include <assert.h>
#include <time.h>

#if defined _WIN32 || defined _WINDOWS
#pragma pack(push,localid,1)
#endif

#include <essapi.h>
#if defined _WIN32 || defined _WINDOWS
#pragma pack(pop,localid)
#endif

ESS_HINST_T hInst;
ESS_HCTX_T hCtx;
#define MAXQRYLEN 65536
ESS_CHAR_T qry[MAXQRYLEN];
ESS_STR_T AppName = "Sample";
ESS_STR_T DbName = "Basic";

static ESS_CHAR_T *axisnames[] =
{
    "COLUMNS", "ROWS", "PAGES", "CHAPTERS", "SECTIONS"
};

void ESS_Init()
{
    ESS_STS_T sts;
    ESS_INIT_T InitStruct = {ESS_API_VERSION,
                              NULL, 0L,
                              255, NULL,
                              NULL, NULL,
                              NULL, NULL,
                              NULL, NULL,
                              NULL, NULL,
                              NULL, NULL,
                              NULL, NULL,
                              0L
    };
    if ((sts = EssInit(&InitStruct, &hInst)) != ESS_STS_NOERR)
    {
        printf("EssInit failure: %ld\n", sts);
        exit ((int) sts);
    }
    printf("EssInit sts: %ld\n", sts);
}

void ESS_Login ()
{
ESS_STS_T sts = ESS_STS_NOERR;
ESS_USHORT_T Items;
ESS_PAPPDB_T pAppsDbs = NULL;
ESS_CHAR_T SvrName[ESS_SVRNAMELEN];
ESS_CHAR_T UserName[ESS_USERNAMELEN];
ESS_CHAR_T Password[ESS_PASSWORDLEN];

/* Initialize parameters */
strcpy(SvrName,"localhost");
strcpy(UserName,"essexer");
strcpy(Password,"password");
sts = EssLogin(hInst, SvrName, UserName, Password, &Items,
               &pAppsDbs, &hCtx);
if ( (sts != 0) && (sts != 1051093L) && (sts != 1051090L) )
{
    printf("EssLogin failure: %ld\n", sts);
    exit ((int) sts);
}
printf("EssLogin sts: %ld\n", sts);

/* MdxAxis */

void ESS_MdxAxis(ESS_MDX_QRYHDL_T hQry,
                 ESS_MDX_AXISHDL_T hAxis,
                 ESS_STR_T pszAxisName
                 )
{

    ESS_STS_T    sts;
    ESS_ULONG_T ulNAxisDims, ulAxisSize;
    ESS_ULONG_T ulNClusters, ulClusterSize, ulNClusterDims;
    ESS_ULONG_T ulAxisDimCnt, ulIndex, ulPropCnt;
    ESS_ULONG_T ulClusterCnt, ulClusterDimCnt;
    ESS_PULONG_T ulaDimSizes;
    ESS_MDX_PCLUSTERHDL_T haClusters;
    ESS_MDX_CLUSTERHDL_T hCluster;
    ESS_MDX_PDIMHDL_T haDims;
    ESS_STR_T pszDimName, pszMbrIdentifier, pszPropName;
    ESS_MDX_PMBRHDL_T haMbrs;
    ESS_MDX_PPROPHDL_T haaProps = NULL;
    ESS_MDX_PPROPHDL_T haProps = NULL;
    ESS_MDX_PROPTYPE_T propType;
    ESS_MDX_PROPVALUE_T propval;

    if ((sts = EssMdxGetAxisInfo(hAxis, &ulAxisSize, &ulNAxisDims,
                                  &haDims)) != ESS_STS_NOERR)
    {
        printf("EssMdxGetAxisInfo failure: %ld\n", sts);
        exit ((int) sts);
    }
    printf("EssMdxGetAxisInfo sts: %ld\n", sts);
    printf("%s Size %ld Num dims %ld\n", pszAxisName,
           ulAxisSize, ulNAxisDims);
    if (ulAxisSize == 0)
    {
        return;
    }
}

MDX Sample Client Program 1607
if ((sts = EssAlloc(hInst, 
    ulNAxisDims * sizeof(ESS_ULONG_T), 
    (ESS_PPVOID_T) &ulaNProps)) != ESS_STS_NOERR)
{
    printf("EssAlloc failure: %ld\n", sts);
    exit ((int) sts);
}
printf("EssAlloc sts: %ld\n", sts);
if ((sts = EssAlloc(hInst, 
    ulNAxisDims * sizeof(ESS_MDX_PPROPHDL_T), 
    (ESS_PPVOID_T) &haaProps)) != ESS_STS_NOERR)
{
    printf("EssAlloc failure: %ld\n", sts);
    exit ((int) sts);
}
printf("EssAlloc sts: %ld\n", sts);
for (ulAxisDimCnt = 0; ulAxisDimCnt < ulNAxisDims; 
    ulAxisDimCnt++)
{
    if ((sts = EssMdxGetDimInfo(haDims[ulAxisDimCnt], 
        &pszDimName, 
        &ulaNProps[ulAxisDimCnt], 
        &haaProps[ulAxisDimCnt])) != ESS_STS_NOERR)
    {
        printf("EssMdxGetDimInfo failure: %ld\n", sts);
        exit ((int) sts);
    }
    printf("EssMdxGetDimInfo sts: %ld\n", sts);
    printf("Dim %ld name %s #props %ld\n", ulAxisDimCnt, 
        pszDimName, ulaNProps[ulAxisDimCnt]);
    haProps = haaProps[ulAxisDimCnt];
    for (ulPropCnt = 0; ulPropCnt < ulaNProps[ulAxisDimCnt]; ulPropCnt++)
    {
        hProp = haProps[ulPropCnt];
        if ((sts = EssMdxGetPropertyInfo(hProp, &pszPropName, 
            &propType)) != ESS_STS_NOERR)
        {
            printf("EssMdxGetPropertyInfo failure: %ld\n", sts);
            exit ((int) sts);
        }
        printf("EssMdxGetPropertyInfo sts: %ld\n", sts);
        printf("Property %ld type %ld name %s\n", ulPropCnt, 
            propType, pszPropName);
    }
}
if ((sts = EssMdxGetClusters(hAxis, &ulNClusters, 
    &haClusters)) != ESS_STS_NOERR)
{
    printf("EssMdxGetClusters failure: %ld\n", sts);
    exit ((int) sts);
}
printf("EssMdxGetClusters sts: %ld\n", sts);
printf("Num clusters %ld\n", ulNClusters);
for (ulClusterCnt = 0; ulClusterCnt < ulNClusters; 
    ulClusterCnt++)
{
hCluster = haClusters[ulClusterCnt];
if ((sts = EssMdxGetClusterInfo(hCluster, &ulClusterSize,
    &ulNClusterDims,
    &ulaDimSizes)) != ESS_STS_NOERR)
{
    printf("EssMdxGetClusterInfo failure: %ld\n", sts);
    exit ((int) sts);
}
printf("EssMdxGetClusterInfo sts: %ld\n", sts);
printf("Cluster %ld Size %ld\n", ulClusterCnt, ulClusterSize);
for (ulClusterDimCnt = 0; ulClusterDimCnt < ulNClusterDims;
    ulClusterDimCnt++)
{
    printf("Cluster Dim %ld Size %ld\n", ulClusterDimCnt,
        ulaDimSizes[ulClusterDimCnt]);
}
for (ulIndex = 0; ulIndex < ulClusterSize; ulIndex++)
{
    if ((sts = EssMdxGetClusterMembers(hCluster, ulIndex,
        &haMbrs)) != ESS_STS_NOERR)
    {
        printf("EssMdxGetClusterMembers failure: %ld\n", sts);
        exit ((int) sts);
    }
    printf("EssMdxGetClusterMembers sts: %ld\n", sts);
    printf("Cluster %ld Size %ld\n", ulIndex, ulClusterCnt, ulClusterSize);
    for (ulClusterDimCnt = 0; ulClusterDimCnt < ulNClusterDims;
        ulClusterDimCnt++)
    {
        if ((sts = EssMdxGetMbrIdentifier(haMbrs[ulClusterDimCnt],
            &pszMbrIdentifier)) != ESS_STS_NOERR)
        {
            printf("EssMdxGetMbrIdentifier failure: %ld\n", sts);
            exit ((int) sts);
        }
        printf("EssMdxGetMbrIdentifier sts: %ld\n", sts);
        printf("Mbr %ld identifier %s\n", ulClusterDimCnt, 
            pszMbrIdentifier);
        haProps = haaProps[ulClusterDimCnt];
        for (ulPropCnt = 0;
            ulPropCnt < ulaNProps[ulClusterDimCnt];
            ulPropCnt++)
        {
            if ((sts = EssMdxGetMbrProperty(haMbrs[ulClusterDimCnt], 
                haProps[ulPropCnt],
                &propval)) != ESS_STS_NOERR)
            {
                printf("EssMdxGetMbrProperty failure: %ld\n", sts);
                exit ((int) sts);
            }
            printf("EssMdxGetMbrProperty sts: %ld\n", sts);
            printf("Property %ld Type ", ulPropCnt);
            switch (propval.ulPropType)
            {
                case ESS_MDX_PROPTYPE_ULONG:
                    {
                        printf("Ulong Value: %ld\n", 
                            propval.value.ulData);
                    }
                    break;
break;
}
case ESS_MDX_PROPTYPE_STRING:
{
    printf("String Value: %s
", propval.value.strData);
    break;
}
case ESS_MDX_PROPTYPE_BOOL:
{
    printf("Bool Value: %s
", propval.value.bData ? "TRUE" : "FALSE");
    break;
}
case ESS_MDX_PROPTYPE_DOUBLE:
{
    printf("Double Value: %lf
", propval.value.dblData);
    break;
}
case ESS_MDX_PROPTYPE_DATETIME:
{
    ESS_CHAR_T tmpbuf[80];
    struct tm* pTime;
    pTime = gmtime((time_t*)&(propval.value.dtData));
    sprintf(tmpbuf, "%02i-%02i-%04i",
            pTime->tm_mon+1, pTime->tm_mday,pTime->tm_year+1900);
    printf("DateTime Value: %s
", tmpbuf);
    break;
}
case ESS_MDX_PROPTYPE_NONE:
{
    printf("NULL Value\n");
    break;
}
}
}
if ((sts = EssFree(hInst, (ESS_PVOID_T) haMbrs)) != ESS_STS_NOERR)
{
    printf("EssFree failure: %ld\n", sts);
    exit ((int) sts);
}
printf("EssFree sts: %ld\n", sts);
}
for (ulClusterDimCnt = 0; ulClusterDimCnt < ulNClusterDims;
    ulClusterDimCnt++)
{
    if ((sts = EssMdxGetClusterDimMembers(hCluster, ulClusterDimCnt,
        &haMbrs)) != ESS_STS_NOERR)
    {
        printf("EssMdxGetClusterDimMembers failure: %ld\n", sts);
        exit ((int) sts);
    }
    printf("EssMdxGetClusterDimMembers sts: %ld\n", sts);
    for (ulIndex = 0; ulIndex < ulaDimSizes[ulClusterDimCnt];
        ulIndex++)
if ((sts = EssMdxGetMbrIdentifier(haMbrs[ulIndex],
            &pszMbrIdentifier)) != ESS_STS_NOERR)
{
    printf("EssMdxGetMbrIdentifier failure: %ld\n", sts);
    exit ((int) sts);
}
printf("EssMdxGetMbrIdentifier sts: %ld\n", sts);
printf("Dim %ld Mbr %ld identifier %s\n", ulClusterDimCnt,
    ulIndex, pszMbrIdentifier);
}
}
for (ulIndex = 0; ulIndex < ulAxisSize; ulIndex++)
{
    if ((sts = EssMdxGetAxisMembers(hAxis, ulIndex,
                    &haMbrs)) != ESS_STS_NOERR)
    {
        printf("EssMdxGetAxisMembers failure: %ld\n", sts);
        exit ((int) sts);
    }
    printf("EssMdxGetAxisMembers sts: %ld\n", sts);
    for (ulAxisDimCnt = 0; ulAxisDimCnt < ulNAxisDims;
        ulAxisDimCnt++)
    {
        if ((sts = EssMdxGetMbrIdentifier(haMbrs[ulAxisDimCnt],
                        &pszMbrIdentifier)) != ESS_STS_NOERR)
        {
            printf("EssMdxGetMbrIdentifier failure: %ld\n", sts);
            exit ((int) sts);
        }
        printf("EssMdxGetMbrIdentifier sts: %ld\n", sts);
        printf("Mbr %ld identifier %s\n", ulAxisDimCnt, pszMbrIdentifier);
        haProps = haaProps[ulAxisDimCnt];
        for (ulPropCnt = 0;
            ulPropCnt < ulaNProps[ulAxisDimCnt];
            ulPropCnt++)
        {
            hProp = haProps[ulPropCnt];
            if ((sts = EssMdxGetPropertyInfo(hProp, &pszPropName,
                                            &propType)) != ESS_STS_NOERR)
            {
                printf("EssMdxGetPropertyInfo failure: %ld\n", sts);
                exit ((int) sts);
            }
            printf("EssMdxGetPropertyInfo sts: %ld\n", sts);
            if ((sts = EssMdxGetMbrProperty(haMbrs[ulAxisDimCnt],
                                            hProp,
                                            &propval)) != ESS_STS_NOERR)
            {
                printf("EssMdxGetMbrProperty failure: %ld\n", sts);
                exit ((int) sts);
            }
            printf("EssMdxGetMbrProperty sts: %ld\n", sts);
            printf("Property %ld Type ", ulPropCnt);
            switch (propval.ulPropType)
            {
                case ESS_MDX_PROPTYPE_ULONG:
case ESS_MDX_PROPTYPE_STRING:
{
    printf("String Value: %s\n", propval.value.strData);
    break;
}
case ESS_MDX_PROPTYPE_BOOL:
{
    printf("Bool Value: %s\n", propval.value.bData ? "TRUE" : "FALSE");
    break;
}
case ESS_MDX_PROPTYPE_DOUBLE:
{
    printf("Double Value: %lf\n", propval.value.dblData);
    break;
}
case ESS_MDX_PROPTYPE_DATETIME:
{
    ESS_CHAR_T tmpbuf[80];
    struct tm* pTime;
    pTime = gmtime((time_t*)&(propval.value.dtData));
    sprintf(tmpbuf, "%02i-%02i-%04i", pTime->tm_mon+1, pTime->tm_mday,pTime->tm_year+1900);  
    printf("DateTime Value: %s\n", tmpbuf);
    break;
}
case ESS_MDX_PROPTYPE_NONE:
{
    printf("NULL Value\n");
    break;
}
}

if ((sts = EssFree(hInst, (ESS_PVOID_T) haMbrs)) != ESS_STS_NOERR)
{
    printf("EssFree failure: %ld\n", sts);
    exit ((int) sts);
}
printf("EssFree sts: %ld\n", sts);

if ((sts = EssFree(hInst, (ESS_PVOID_T) ulaNProps)) != ESS_STS_NOERR)
{
    printf("EssFree failure: %ld\n", sts);
    exit ((int) sts);
}
printf("EssFree sts: %ld\n", sts);
if ((sts = EssFree(hInst, (ESS_PVOID_T) haaProps)) != ESS_STS_NOERR)
{
    printf("EssFree failure: %ld\n", sts);
}
void ESS_MdxQry()
{
    ESS_STS_T    sts;
    ESS_MDX_QRYHDL_T hQry;
    ESS_ULONG_T ulNAxes, ulAxisDims, ulAxisSize, ulResultSize;
    ESS_ULONG_T ulNCclusters, ulClusterSize, ulNCclusterDims;
    ESS_ULONG_T ulAxisCnt, ulAxisDimCnt, ulIndex, ulPropCnt;
    ESS_ULONG_T ulCellOffset, ulClusterCnt, ulClusterDimCnt;
    ESS_MDX_CELLSTATUS_T ulCellStatus;
    ESS_PULONG_T ulaDimSizes;
    ESS_MDX_PCLUSTERHDL_T haClusters;
    ESS_MDX_CLUSTERHDL_T hCluster;
    ESS_MDX_PAXISHDL_T haAxes;
    ESS_MDX_PDIMHDL_T haDims;
    ESS_STR_T pszDimName, pszMbrIdentifier, pszPropName;
    ESS_MDX_AXISHDL_T hAxis, hSlicerAxis;
    ESS_MDX_PMEMHDL_T haMbrs;
    ESS_MDX_CELLHDL_T hCell;
    ESS_DOUBLE_T dValue;
    ESS_BOOL_T bIsMissing, bNoAccess;
    ESS_PULONG_T ulaNProps;
    ESS_MDX_PPPROPHDL_T haaProps;
    ESS_MDX_PPROPHDL_T haProps;
    ESS_MDX_PROPHDL_T hProp;
    ESS_MDX_PROPTYPE_T propType;
    ESS_MDX_PROPVALUE_T propval;

    if ((sts = EssMdxNewQuery(hCtx, qry, &hQry)) != ESS_STS_NOERR)
    {
        printf("EssMdxNewQuery failure: %ld\n", sts);
        exit ((int) sts);
    }
    printf("EssMdxNewQuery sts: %ld\n", sts);

    if ((sts = EssMdxSetMbrIdType(hQry, ESS_MDX_MEMBERIDTYPE_ALIAS)) !=
        ESS_STS_NOERR)
    {
        printf("EssMdxSetMbrIdType failure: %ld\n", sts);
        exit ((int) sts);
    }
    printf("EssMdxSetMbrIdType sts: %ld\n", sts);

    if ((sts = EssMdxSetNeedCellStatus(hQry)) != ESS_STS_NOERR)
    {
        printf("EssMdxSetNeedCellStatus failure: %ld\n", sts);
        exit ((int) sts);
    }
    printf("EssMdxSetNeedCellStatus sts: %ld\n", sts);

    if ((sts = EssMdxExecuteQuery(hQry)) != ESS_STS_NOERR)
    {
        printf("EssMdxExecuteQuery failure: %ld\n", sts);
    }
exit ((int) sts);
}
printf("EssMdxExecuteQuery sts: %ld\n", sts);
if ((sts = EssMdxGetAxes(hQry, &ulNAxes, &haAxes, &hSlicerAxis)) != ESS_STS_NOERR)
{
    printf("EssMdxGetAxes failure: %ld\n", sts);
    exit ((int) sts);
}
printf("EssMdxGetAxes sts: %ld\n", sts);
printf("Number of axes: %ld", ulNAxes);
ulResultSize = 1;
for (ulAxisCnt = 0; ulAxisCnt < ulNAxes; ulAxisCnt++)
{
    hAxis = haAxes[ulAxisCnt];
    if ((sts = EssMdxGetAxisInfo(hAxis, &ulAxisSize, &ulNAxisDims, &haDims)) != ESS_STS_NOERR)
    {
        printf("EssMdxGetAxisInfo failure: %ld\n", sts);
        exit ((int) sts);
    }
    printf("EssMdxGetAxisInfo sts: %ld\n", sts);
    printf("Axis %ld Size %ld Num dims %ld\n", ulAxisCnt, ulAxisSize, ulNAxisDims);
    ulResultSize *= ulAxisSize;
}
if (hSlicerAxis)
    ESS_MdxAxis(hQry, hSlicerAxis, "SLICER");
else
    printf("Slicer Axis is empty\n");
for (ulAxisCnt = 0; ulAxisCnt < ulNAxes; ulAxisCnt++)
{
    hAxis = haAxes[ulAxisCnt];
    ESS_MdxAxis(hQry, hAxis, axisnames[ulAxisCnt]);
}
for (ulCellOffset = 0; ulCellOffset < ulResultSize; ulCellOffset++)
{
    if ((sts = EssMdxGetCellAtOffset(hQry, ulCellOffset, &hCell)) != ESS_STS_NOERR)
    {
        printf("EssMdxGetCellAtOffset failure: %ld\n", sts);
        exit ((int) sts);
    }
    printf("EssMdxGetCellAtOffset sts: %ld\n", sts);
    if ((sts = EssMdxGetValue(hCell, &bIsMissing, &bNoAccess, &dValue)) != ESS_STS_NOERR)
    {
        printf("EssMdxGetValue failure: %ld\n", sts);
    } else
    {
        printf("EssMdxGetValue ok\n");
    }
}
exit ((int) sts);
}
printf("EssMdxGetValue sts: %ld\n", sts);
if (bIsMissing)
{
    printf("CellOffset %ld Value #Missing\n", ulCellOffset);
} else if (bNoAccess)
{
    printf("CellOffset %ld Value #NoAccess\n", ulCellOffset);
} else
{
    printf("CellOffset %ld Value %lf\n", ulCellOffset, dValue);
}
if (!bNoAccess)
{
    if ((sts = EssMdxGetCellStatus(hQry, hCell, 
        &ulCellStatus)) != ESS_STS_NOERR)
    {
        printf("EssMdxGetCellStatus failure: %ld\n", sts);
        exit ((int) sts);
    }
    printf("EssMdxGetCellStatus sts: %ld\n", sts);
    if (ulCellStatus & ESS_MDX_CELLSTATUS_LINKEDOBJS)
    {
        printf("Cell status: LINKEDOBJS\n");
    }
    if (ulCellStatus & ESS_MDX_CELLSTATUS_DYNCALC)
    {
        printf("Cell status: DYNCALC\n");
    }
    if (ulCellStatus & ESS_MDX_CELLSTATUS_CALCEDMBR)
    {
        printf("Cell status: CALCEDMBR\n");
    }
    if (ulCellStatus & ESS_MDX_CELLSTATUS_READONLY)
    {
        printf("Cell status: READONLY\n");
    }
}
if ((sts = EssMdxFreeQuery(hQry)) != ESS_STS_NOERR)
{
    printf("EssMdxFreeQuery failure: %ld\n", sts);
    exit ((int) sts);
}
printf("EssMdxFreeQuery sts: %ld\n", sts);
}
void ESS_Term()
{
    ESS_STS_T sts = ESS_STS_NOERR;
    if ((sts = EssTerm(hInst)) != ESS_STS_NOERR)
/* error terminating API */
exit((ESS_USHORT_T) sts);

printf("EssTerm sts: %ld\n", sts);
}

void ESS_Logout()
{
    ESS_STS_T sts = ESS_STS_NOERR;
    sts = EssLogout(hCtx);
    printf("\n\nEssLogout sts: %ld\n",sts);
}

void ESS_SetActive()
{
    ESS_STS_T sts = ESS_STS_NOERR;
    ESS_ACCESS_T Access;
    sts = EssSetActive(hCtx, AppName, DbName, &Access);
    printf("EssSetActive sts: %ld\n",sts);
}

int main(int argc, char *argv[])
{
    FILE *f;
    char *s, *sout;
    int n, l, e;

    assert(argc > 1);
    f = fopen(argv[1], "r");
    assert(f != NULL);
    s = qry;
    n = MAXQRYLEN;
    while (n > 0 && !feof(f) && fgets(s, n, f) != NULL)
    {
        l = strlen(s);
        s += l;
        n -= l;
    }
    if ((e = ferror(f)) != 0)
    {
        printf("fgets error %d\n", e);
        exit((int) e);
    }
    fclose(f);
    printf("The query is\n%s\n", qry);
    if (argc > 2)
    {
        AppName = argv[2];
    }
    if (argc > 3)
    {
        DbName = argv[3];
    }
ESS_Init();
ESS_Login();
ESS_SetActive();
ESS_MdxQry();
ESS_Logout();
ESS.Term();
return 0;
}
To use XML for Analysis (XMLA) API, you must install Provider Services. See the Oracle Hyperion Enterprise Performance Management System Installation Start Here and Oracle Hyperion Enterprise Performance Management System Installation and Configuration Guide. This help explains XMLA methods and provides sample code for rowsets. XMLA clients can communicate to Essbase only through Provider Services.

For information, click Contents, Index, or Search in the left frame.
Key Features

XML for Analysis (XMLA) is an open industry-standard Web service interface designed for online analytical processing. XMLA is a set of XML Message Interfaces built on the open standards of HTTP, XML, and Simple Object Access Protocol (SOAP). XMLA, which is not bound to any language, platform, or operating system, provides standardized data access between client applications and any multidimensional data source on the Web.

For more information on XMLA, visit www.xmla.org.

Key XMLA features:

- Support for flattened rowsets
- Support for stateful sessions
- Backward XMLA level representation (level 1 is the top level)
- User authentication through basic HTTP authentication
- XMLA High-Availability functionality through Oracle Hyperion Provider Services
- XMLA administration and monitoring through Oracle Essbase Administration Services

Note: XMLA is available for use with Essbase only.

Methods

The following methods provide a standard way for XML applications to access basic information from the server. Because these methods are invoked using SOAP, they accept input and deliver output in XML. By default, these methods are stateless, so the server context ends at the completion of any command.

The simplified interface model has two methods.
Discover obtains information and metadata from a Web Service. This information can include a list of available data sources and data about a data source provider. Properties define and shape the data obtained. Discover allows you to specify the types of information that the client application needs. The use of generic interface and properties enables extensibility without necessitating rewriting existing functions.

Execute executes Multidimensional Expressions (MDX) or other provider-specific commands against an XMLA data source. The following diagram illustrates a possible implementation of an n-tiered application.

Provided with the URL for a server hosting a Web Service, the client uses SOAP and HTTP protocols to send Discover and Execute calls to the server. The server instantiates the XMLA provider, which handles the calls. The XMLA provider fetches the data, packages it into XML, and sends the data to the client.

The Discover and Execute methods enable users to determine what can be queried on a server and, based on this, submit commands to be executed.

The XML namespace for these methods is “urn:schemas-microsoft-com:xml-analysis”. Connection information is supplied in each method call with the connection properties.

## Discover

The Discover method retrieves information, such as the list of data sources on a server or details about a data source. The data retrieved with the Discover method depends on the values of the parameters passed to it.

**Namespace**

`urn:schemas-microsoft-com:xml-analysis`

**SOAP Action**

"urn:schemas-microsoft-com:xml-analysis:Discover"

**Syntax**

```
Discover (  
    [in] RequestType As EnumString,  
    [in] Restrictions As Restrictions,  
    [in] Properties As Properties,  
    [out] Result As Rowset)
```

**Parameters**

RequestType [in]
This required parameter comprises a RequestType enumeration value, which determines the type of information to be returned. The RequestType enumeration is used by the Discover method to determine the structure and content of the rowset returned in the Result parameter. The Restrictions parameter format and XML result set are also dependent on the value specified in this parameter. This enumeration can be extended to support provider-specific enumeration strings.

Each RequestType enumeration value corresponds to a return rowset. For rowset definitions, see “XMLA Rowsets” on page 1627. Support is required for the following explicitly named RequestType enumeration values.

<table>
<thead>
<tr>
<th>Enumeration value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISCOVER_DATASOURCES</td>
<td>Returns a list of XMLA data sources available on the server or Web Service. (For an example of how these may be published, see “XMLA Implementation Walkthrough” in the XML for Analysis Specification, available on the Hyperion Developer Network.)</td>
</tr>
<tr>
<td>DISCOVER_PROPERTIES</td>
<td>Returns a list of information and values about the requested properties that are supported by the specified data source (provider).</td>
</tr>
<tr>
<td>DISCOVER_SCHEMA_ROWSETS</td>
<td>Returns the names, values, and other information of all supported RequestType enumeration values (including those listed here), and any additional provider-specific enumeration values.</td>
</tr>
<tr>
<td>DISCOVER_ENUMERATORS</td>
<td>Returns a list of names, data types, and enumeration values of enumerators supported by the provider of a specific data source.</td>
</tr>
<tr>
<td>DISCOVER_KEYWORDS</td>
<td>Returns a rowset containing a list of keywords reserved by the provider.</td>
</tr>
<tr>
<td>DISCOVER_LITERALS</td>
<td>Returns information about literals supported by the data source provider. Schema Rowset Constant Given, a constant that corresponds to one of the schema rowset names defined by OLE DB, such as MDSCHEMA_CUBES, returns the OLE DB schema rowset in XML format. Note that providers also may extend OLEDB by providing additional provider-specific schema rowsets. The schema rowsets that tabular data providers (TDP) and multidimensional data providers (MDP) are required to support are listed in the section &quot;DISCOVER_SCHEMA_ROWSETS Rowset.&quot;</td>
</tr>
</tbody>
</table>

Restrictions [in]

This parameter, of the Restrictions data type, enables the user to restrict the data returned in Result. Result columns are defined by the rowset specified in the RequestType parameter. Some columns of Result can filter the rows returned. For these columns and those that can be restricted, see the rowset tables in “XMLA Rowsets” on page 1627. To obtain the restriction information for provider-specific schema rowsets, use the DISCOVER_SCHEMA_ROWSETS request type. This parameter can be empty, but it must be included.

Properties [in]

This parameter, of the Properties data type, comprises a collection of XMLA properties. Each property enables users to control some aspect of the Discover method, such as specifying the return format of the result set, the timeout, or the locale in which the data should be formatted.

You can obtain the available properties by using the DISCOVER_PROPERTIES request type with the Discover method.

The properties in the Properties parameter have no required order. This parameter can be empty, but it must be included.
This required parameter contains the result set returned by the provider as a Rowset object. The columns and content of the result set are specified by the values in the RequestType and Restrictions parameters. The column layout of the returned result set also is determined by the value specified in RequestType. For information about the rowset layouts that correspond to for each RequestType value, see “XMLA Rowsets” on page 1627.

Example

In the following sample, the client sends the XML Discover call to request a list of cubes from the Demo catalog:

```xml
<SOAP-ENV:Envelope
 xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <SOAP-ENV:Body>
 <Discover xmlns="urn:schemas-microsoft-com:xml-analysis"
 SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <RequestType>MDSCHEMA_CUBES</RequestType>
  <Restrictions>
   <RestrictionList>
    <CATALOG_NAME>Demo</CATALOG_NAME>
   </RestrictionList>
  </Restrictions>
  <Properties>
   <PropertyList>
    <DataSourceInfo>
     Provider=Essbase;Data Source=localhost
    </DataSourceInfo>
    <Format>Tabular</Format>
   </PropertyList>
  </Properties>
 </Discover>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The provider returns the following result to the client:

```xml
<?xml version="1.0"?>
 SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
 <SOAP-ENV:Body>
  <m:DiscoverResponse xmlns:m="urn:schemas-microsoft-com:xml-analysis">
   <m:return xsi:type="xsd:string">
    <root xmlns="urn:schemas-microsoft-com:xml-analysis:rowset"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema">
     <xsd:schema xmlns="urn:schemas-microsoft-com:xml-analysis:rowset"
      targetNamespace="urn:schemas-microsoft-com:xml-analysis:rowset"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:sql="urn:schemas-microsoft-com:xml-sql"
      elementFormDefault="qualified">
      <xsd:element name="root">
      </root>
     </xsd:schema>
    </root>
   </m:return>
  </m:DiscoverResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```
Execute

The Execute method sends action requests, including those involving data transfer, such as retrieving or updating data on the server, to the server.

Namespace

urn:schemas-microsoft-com:xml-analysis

SOAP Action

"urn:schemas-microsoft-com:xml-analysis:Execute"

Syntax

Execute (  
  [in] Command As Command,
  [in] Properties As Properties,
  [out] Result As Resultset)

Parameters

Command [in]

This required parameter is of Command data type and consists of an MDX statement to be executed.
Properties [in]

This parameter is of the Properties data type and consists of a collection of XMLA properties. Each property allows the user to control some aspect of the Execute method, such as defining the information required for the connection, specifying the return format of the result set, or specifying the locale in which the data should be formatted.

The available properties and their values can be obtained by using the DISCOVER_PROPERTIES request type with the Discover method.

The properties in the Properties parameter have no required order. This parameter can be empty, but it must be included.

Result [out]

This parameter contains the Resultset result returned by the provider. The Command parameter and values in the Properties parameter define the shape of the result set. If no shape-defining properties are passed, the XMLA provider may use a default shape. The two result set formats defined by this specification are Tabular and Multidimensional, as specified by the client through the Format property. OLAP data lends itself to the Multidimensional format (although the Tabular format also can be used). A provider may support additional rowset types, and clients aware of the specialized types can request them.

Example

The following is an example of an Execute method call with <Statement> set to an MDX SELECT statement:

```xml
xmns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
xmns:xsd="http://www.w3.org/2001/XMLSchema">   
<SOAP-ENV:Body>
<Execute xmlns="urn:schemas-microsoft-com:xml-analysis"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"> 
<Command>
<Statement>
SELECT  CrossJoin([Measures].CHILDREN , [Market].CHILDREN)
on columns,  [Product].Members on rows
from Sample.Basic
</Statement>
</Command>
<Properties>
<DataSourceInfo>
Provider=Essbase;Data Source=localhost
</DataSourceInfo>
<Catalog>Sample</Catalog>
<Format>Multidimensional</Format>
<AxisFormat>TupleFormat</AxisFormat>
<Content>SchemaData</Content>
</PropertyList></Properties>
</Execute>
</SOAP-ENV:Body>
xmns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
xmns:xsd="http://www.w3.org/2001/XMLSchema">
The abbreviated response for the preceding method call:

```xml
<?xml version="1.0"?>
<SOAP-ENV:Envelope
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<SOAP-ENV:Body>
  <m:ExecuteResponse
xmlns:m="urn:schemas-microsoft-com:xml-analysis">
    <m:return>
      <root xmlns="urn:schemas-microsoft-com:xml-analysis:mddataset">
        <!-The schema for the data goes here. -- >
        <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xars="urn:schemas-microsoft-com:xars">
          ...<!-The data for the data goes here. -- >
        </xsd:schema>
      </root>
    </m:return>
  </m:ExecuteResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

**XMLA Rowsets**

Information returned in the Result parameter of the Discover method is structured according to the rowset column layouts detailed in this section.

**CATALOGS Rowset**

The CATALOGS rowset identifies the physical attributes associated with catalogs accessible from Analytic Services.

**GUID:** DBSCHEMA_CATALOGS

the section called “Flattened Rowset Examples” describes the rowset structure.

<table>
<thead>
<tr>
<th>Table 32</th>
<th>CATALOGS Rowset Structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Column Name</td>
<td>Essbase Mapping</td>
</tr>
<tr>
<td>CATALOG_NAME</td>
<td>Application name</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>Always null</td>
</tr>
</tbody>
</table>

**Request Example**

```xml
xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <SOAP-ENV:Body>
    <Discover xmlns="urn:schemas-microsoft-com:xml-analysis"
      SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
```

XMLA Rowsets 1627
Response Example (truncated)

```xml
<?xml version="1.0"?>
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<SOAP-ENV:Body>
<m:DiscoverResponse xmlns:m="urn:schemas-microsoft-com:xml-analysis">
 <m:return xsi:type="xsd:string"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <root xmlns="urn:schemas-microsoft-com:xml-analysis:rowset"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:sql="urn:schemas-microsoft-com:xml-sql"
 elementFormDefault="qualified">
 <xsd:element name="root">
 <xsd:complexType>
 <xsd:sequence minOccurs="0" maxOccurs="unbounded">
 <xsd:element name="row" type="xsd:string"/>
 </xsd:sequence>
 </xsd:complexType>
 </xsd:element>
 <xsd:complexType name="row">
 <xsd:sequence maxOccurs="unbounded" minOccurs="0">
 <xsd:element name="CATALOG_NAME" type="xsd:string"
 sql:field="CATALOG_NAME"/>
 <xsd:element name="DESCRIPTION" type="xsd:string"
 sql:field="DESCRIPTION" minOccurs="0"/>
 </xsd:sequence>
 </xsd:complexType>
 </xsd:schema>
 <row>
 <CATALOG_NAME>Demo</CATALOG_NAME>
 </row>
 <.................More Rows.............>
 <root>
 </m:return>
 </m:DiscoverResponse>
```
**MDSCHEMA_CUBES Rowset**

The CUBES rowset contains information about the available cubes in a schema (or the catalog, if the provider does not support schemas).

**GUID: MDSCHEMA_CUBES**

Table 33 describes the rowset structure.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Essbase Mapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>CATALOG_NAME</td>
<td>Application name</td>
</tr>
<tr>
<td>CUBE_NAME</td>
<td>Database name</td>
</tr>
<tr>
<td>CUBE_TYPE</td>
<td>“CUBE”</td>
</tr>
<tr>
<td>LAST_SCHEMA_UPDATE</td>
<td>Time stamp of last outline update</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>Database description</td>
</tr>
</tbody>
</table>

**Request Example**

```xml
<SOAP-ENV:Envelope
 xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <SOAP-ENV:Body>
  <Discover xmlns="urn:schemas-microsoft-com:xml-analysis"
   SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <RequestType>MDSCHEMA_CUBES</RequestType>
   <Restrictions>
    <RestrictionList>
     <CATALOG_NAME>Demo</CATALOG_NAME>
    </RestrictionList>
   </Restrictions>
   <Properties>
    <PropertyList>
     <DataSourceInfo>
      Provider=Essbase;Data Source=localhost
     </DataSourceInfo>
     <Format>Tabular</Format>
    </PropertyList>
   </Properties>
  </Discover>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

**Response Example**

```xml
<?xml version="1.0"?>
```
MDSCHEMA_DIMENSIONS Rowset

The DIMENSIONS rowset contains information about the dimensions in a given cube. Each dimension has one row.

GUID: MDSCHEMA_DIMENSIONS

Table 34 describes the rowset structure.
<table>
<thead>
<tr>
<th>Column Name</th>
<th>Essbase Mapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>CATALOG_NAME</td>
<td>Application name</td>
</tr>
<tr>
<td>CUBE_NAME</td>
<td>Database name</td>
</tr>
<tr>
<td>DIMENSION_NAME</td>
<td>Dimension name</td>
</tr>
<tr>
<td>DIMENSION_UNIQUE_NAME</td>
<td>Dimension name</td>
</tr>
<tr>
<td>DIMENSION_CAPTION</td>
<td>Dimension name</td>
</tr>
<tr>
<td>DIMENSION_ORDINAL</td>
<td>Dimension number. First dimension is 1, second is 2, and so on</td>
</tr>
<tr>
<td>DIMENSION_TYPE</td>
<td>If Essbase dimension type is:</td>
</tr>
<tr>
<td></td>
<td>● TIME: MD_DIMTYPE_TIME</td>
</tr>
<tr>
<td></td>
<td>● ACCOUNTS: MD_DIMTYPE_MEASURE</td>
</tr>
<tr>
<td></td>
<td>● ALL OTHER: MD_DIMTYPE_OTHER</td>
</tr>
<tr>
<td>DIMENSION_CARDINALITY</td>
<td>Number of members in the dimension</td>
</tr>
<tr>
<td>DEFAULT_HIERARCHY</td>
<td>Dimension name</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>Comment added for the dimension</td>
</tr>
<tr>
<td>DIMENSION_UNIQUE_SETTINGS</td>
<td>2</td>
</tr>
<tr>
<td>DIMENSION_IS_VISIBLE</td>
<td>True always</td>
</tr>
</tbody>
</table>

Request Example

```xml
<SOAP-ENV:Envelope
xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <SOAP-ENV:Body>
    <Discover xmlns="urn:schemas-microsoft-com:xml-analysis"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <RequestType>MDSCHEMA_DIMENSIONS</RequestType>
      <Restrictions>
        <RestrictionList>
          <CATALOG_NAME>Sample</CATALOG_NAME>
          <CUBE_NAME>Basic</CUBE_NAME>
        </RestrictionList>
      </Restrictions>
      <Properties>
        <PropertyList>
          <DataSourceInfo>Provider=Essbase;Data Source=localhost</DataSourceInfo>
          <Format>Tabular</Format>
        </PropertyList>
      </Properties>
    </Discover>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```
MDSCHEMA_FUNCTIONS Rowset

The FUNCTIONS rowset exposes all functions supported by the MDP. Default sort order: ORIGIN, INTERFACE_NAME, and FUNCTION_NAME.

GUID: MDSCHEMA_FUNCTIONS

Table 35 describes the rowset structure.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Essbase Mapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>FUNCTION_NAME</td>
<td>Name of the function</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>Description of the function</td>
</tr>
<tr>
<td>PARAM_LIST</td>
<td>A comma delimited list of parameters</td>
</tr>
<tr>
<td>RETURN_TYPE</td>
<td>Always 12</td>
</tr>
<tr>
<td>ORIGIN</td>
<td>1 (always:MDX functions)</td>
</tr>
<tr>
<td>INTERFACE_NAME</td>
<td>One of the following: Member, Set, Tuple, Numeric, Dimension, Level, Boolean</td>
</tr>
<tr>
<td>OBJECT</td>
<td>One of the following values: Set, Member, Tuple, Level, Hierarchy, Dimension</td>
</tr>
<tr>
<td>HELP_CONTEXT</td>
<td>Help context ID for the function</td>
</tr>
<tr>
<td>CAPTION</td>
<td>Display caption of the function</td>
</tr>
</tbody>
</table>

Request Example

```xml
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <SOAP-ENV:Body>
    <Discover xmlns="urn:schemas-microsoft-com:xml-analysis"
      SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```
<RequestType>MDSCHEMA_FUNCTIONS</RequestType>
<Restrictions><RestrictionList/></RestrictionList></Restrictions>

<Properties>
  <PropertyList>
    <DataSourceInfo>Provider=Essbase;Data Source=localhost</DataSourceInfo>
    <Format>Tabular</Format>
  </PropertyList>
</Properties>

Response Example (truncated)

<?xml version="1.0"?>
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
    <m:DiscoverResponse xmlns:m="urn:schemas-microsoft-com:xml-analysis">
            <xsd:element name="root">
              <xsd:complexType>
                <xsd:sequence minOccurs="0" maxOccurs="unbounded">
                  <xsd:element name="row" type="xsd:row"/>
                </xsd:sequence>
              </xsd:complexType>
            </xsd:element>
          </xsd:schema>
          <xsd:element name="row" type="xsd:row"/>
        </root>
      </m:return>
    </m:DiscoverResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

1634 Working with XMLA
The HIERARCHIES rowset contains information about the hierarchies available in a dimension.

**GUID**: MDSCHEMA_HIERARCHIES

Table 36 describes the rowset structure.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Essbase Mapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>CATALOG_NAME</td>
<td>Application name</td>
</tr>
<tr>
<td>CUBE_NAME</td>
<td>Database name</td>
</tr>
<tr>
<td>DIMENSION_UNIQUE_NAME</td>
<td>Dimension name</td>
</tr>
<tr>
<td>HIERARCHY_NAME</td>
<td>Dimension name</td>
</tr>
<tr>
<td>HIERARCHY_UNIQUE_NAME</td>
<td>Dimension name</td>
</tr>
<tr>
<td>HIERARCHY_CAPTION</td>
<td>Dimension name</td>
</tr>
<tr>
<td>DIMENSION_TYPE</td>
<td>If Essbase dimension type is:</td>
</tr>
<tr>
<td></td>
<td>● TIME: MD_DIMTYPE_TIME</td>
</tr>
<tr>
<td></td>
<td>● ACCOUNTS: MD_DIMTYPE_MEASURE</td>
</tr>
<tr>
<td></td>
<td>● ALL OTHER: MD_DIMTYPE_OTHER</td>
</tr>
<tr>
<td>HIERARCHY_CARDINALITY</td>
<td>Number of members in the dimension</td>
</tr>
<tr>
<td>Column Name</td>
<td>Essbase Mapping</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>----------------------------------------------</td>
</tr>
<tr>
<td>DEFAULT MEMBER</td>
<td>Dimension name</td>
</tr>
<tr>
<td>ALL_MEMBER</td>
<td>Dimension name</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>Dimension comment</td>
</tr>
<tr>
<td>STRUCTURE</td>
<td>MD_STRUCTURE_UNBALANCED(2)</td>
</tr>
<tr>
<td>HIERARCHY_UNIQUE_SETTINGS</td>
<td>2</td>
</tr>
<tr>
<td>HIERARCHY_IS_VISIBLE</td>
<td>True</td>
</tr>
</tbody>
</table>

**Request Example**

```xml
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <SOAP-ENV:Body>
    <Discover xmlns="urn:schemas-microsoft-com:xml-analysis"
      SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <RequestType>MDSCHEMA_HIERARCHIES</RequestType>
      <Restrictions>
        <RestrictionList>
          <CUBE_NAME>Sample.Basic</CUBE_NAME>
          <DIMENSION_UNIQUE_NAME>Year</DIMENSION_UNIQUE_NAME>
        </RestrictionList>
      </Restrictions>
      <Properties>
        <PropertyList>
          <DataSourceInfo>Provider=Essbase;Data Source=localhost</DataSourceInfo>
          <Format>Tabular</Format>
        </PropertyList>
      </Properties>
    </Discover>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

**Response Example**

```xml
<?xml version="1.0"?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
    <m:DiscoverResponse xmlns:m="urn:schemas-microsoft-com:xml-analysis">
      <m:return xsi:type="xsd:string">
        <root xmlns="urn:schemas-microsoft-com:xml-analysis:rowset"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xmlns:xsd="http://www.w3.org/2001/XMLSchema">
          <xsd:schema xmlns="urn:schemas-microsoft-com:xml-analysis:rowset"
            targetNamespace="urn:schemas-microsoft-com:xml-analysis:rowset"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
```
<CATALOG_NAME>Sample</CATALOG_NAME>
<CUBE_NAME>Sample.Basic</CUBE_NAME>
<DIMENSION_UNIQUE_NAME>[Year]</DIMENSION_UNIQUE_NAME>
<HIERARCHY_NAME>Year</HIERARCHY_NAME>
<HIERARCHY_UNIQUE_NAME>[Year]</HIERARCHY_UNIQUE_NAME>
<HIERARCHY_CAPTION>Year</HIERARCHY_CAPTION>
<DIMENSION_TYPE>1</DIMENSION_TYPE>
<HIERARCHY_CARDINALITY>19</HIERARCHY_CARDINALITY>
<DEFAULT_MEMBER>[Year]</DEFAULT_MEMBER>
<ALL_MEMBER>[Year]</ALL_MEMBER>
<STRUCTURE>2</STRUCTURE>
<HIERARCHY_UNIQUE_SETTINGS>2</HIERARCHY_UNIQUE_SETTINGS>
<HIERARCHY_IS_VISIBLE>true</HIERARCHY_IS_VISIBLE>
**MDSCHEMA_MEASURES Rowset**

The MEASURES rowset contains information about the available measures.

GUID: MDSCHEMA_MEASURES

Table 37 describes the rowset structure.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Essbase Mapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>CATALOG_NAME</td>
<td>Application name</td>
</tr>
<tr>
<td>CUBE_NAME</td>
<td>Database name</td>
</tr>
<tr>
<td>MEASURE_NAME</td>
<td>Member names in the Accounts dimension</td>
</tr>
<tr>
<td>MEASURE_UNIQUE_NAME</td>
<td>Above member name</td>
</tr>
<tr>
<td>MEASURE_CAPTION</td>
<td>Above member name</td>
</tr>
<tr>
<td>MEASURE_AGGREGATOR</td>
<td>Essbase ADDITION: 1</td>
</tr>
<tr>
<td></td>
<td>Essbase SUBTRACTION: 17</td>
</tr>
<tr>
<td></td>
<td>Essbase MULTIPLICATION:18</td>
</tr>
<tr>
<td></td>
<td>Essbase DIVISION:19</td>
</tr>
<tr>
<td></td>
<td>Essbase PERCENT:20</td>
</tr>
<tr>
<td></td>
<td>Essbase NOOP: 21</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>Member comment</td>
</tr>
<tr>
<td>DATA_TYPE</td>
<td>5</td>
</tr>
<tr>
<td>EXPRESSION</td>
<td>Member formula</td>
</tr>
<tr>
<td>MEASURE_IS_VISIBLE</td>
<td>True</td>
</tr>
</tbody>
</table>

**Request Example**

```xml
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <SOAP-ENV:Body>
    <Discover xmlns="urn:schemas-microsoft-com:xml-analysis"
      SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <RequestType>MDSCHEMA_MEASURES</RequestType>
      <Restrictions>
```

1638 Working with XMLA
<RestrictionList>
  <CATALOG_NAME>Sample</CATALOG_NAME>
  <CUBE_NAME>Basic</CUBE_NAME>
</RestrictionList>

</Restrictions>

<Properties>
  <PropertyList>
    <DataSourceInfo>Provider=Essbase;Data Source=localhost</DataSourceInfo>
    <Format>Tabular</Format>
  </PropertyList>
</Properties>

</Discover>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

Response Example (truncated)

<?xml version="1.0"?>
                  xmlns:SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
    <m:DiscoverResponse xmlns:m="urn:schemas-microsoft-com:xml-analysis">
      <m:return xsi:type="xsd:string">
          <xsd:complexType name="row">
            <xsd:sequence minOccurs="0" maxOccurs="unbounded">
              <xsd:element name="CATALOG_NAME" type="xsd:string" sql:field="CATALOG_NAME"/>
              <xsd:element name="CUBE_NAME" type="xsd:string" sql:field="CUBE_NAME"/>
              <xsd:element name="MEASURE_NAME" type="xsd:string" sql:field="MEASURE_NAME"/>
              <xsd:element name="MEASURE_UNIQUE_NAME" type="xsd:string" sql:field="MEASURE_UNIQUE_NAME"/>
              <xsd:element name="MEASURE_CAPTION" type="xsd:string" sql:field="MEASURE_CAPTION"/>
              <xsd:element name="MEASURE_AGGREGATOR" type="xsd:int" sql:field="MEASURE_AGGREGATOR"/>
              <xsd:element name="DESCRIPTION" type="xsd:string" sql:field="DESCRIPTION" minOccurs="0"/>
            </xsd:sequence>
          </xsd:complexType>
        </root>
      </m:return>
    </m:DiscoverResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
MDSCHEMA_MEMBERS Rowset

The MEMBERS rowset contains information about the available members.

GUID: MDSCHEMA_MEMBERS

Table 38 describes the rowset structure.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Essbase Mapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>CATALOG_NAME</td>
<td>Application name</td>
</tr>
<tr>
<td>CUBE_NAME</td>
<td>Database name</td>
</tr>
<tr>
<td>DIMENSION_UNIQUE_NAME</td>
<td>Dimension name</td>
</tr>
<tr>
<td>HIERARCHY_UNIQUE_NAME</td>
<td>Dimension name</td>
</tr>
<tr>
<td>LEVEL_UNIQUE_NAME</td>
<td>Level name</td>
</tr>
<tr>
<td>LEVEL_NUMBER</td>
<td>Level number</td>
</tr>
<tr>
<td>Column Name</td>
<td>Essbase Mapping</td>
</tr>
<tr>
<td>---------------------</td>
<td>---------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>GENERATION_NUMBER</td>
<td>Generation number</td>
</tr>
<tr>
<td>MEMBER_ORDINAL</td>
<td>Member number</td>
</tr>
<tr>
<td>MEMBER_NAME</td>
<td>Member name</td>
</tr>
<tr>
<td>MEMBER_UNIQUE_NAME</td>
<td>Unique member name</td>
</tr>
<tr>
<td>MEMBER_TYPE</td>
<td>1 (REGULAR)</td>
</tr>
<tr>
<td>MEMBER_CAPTION</td>
<td>Member name</td>
</tr>
<tr>
<td>MEMBER_ALIAS</td>
<td>Default alias</td>
</tr>
<tr>
<td>CHILDREN_CARDINALITY</td>
<td>Child count</td>
</tr>
<tr>
<td>PARENT_LEVEL</td>
<td>Level number of the parent. For dimension, same level number as the dimension level number</td>
</tr>
<tr>
<td>PARENT_UNIQUE_NAME</td>
<td>Name of the parent. For dimension, same name as the dimension name</td>
</tr>
<tr>
<td>PARENT_COUNT</td>
<td>Always 1</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>Member comment</td>
</tr>
</tbody>
</table>

**Request Example**

```xml
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
 secext-1.0.xsd">
 <SOAP-ENV:Header>
  <wsse:Security>
   <wsse:UsernameToken>
    <wsse:Username>system</wsse:Username>
    <wsse:Password>password</wsse:Password>
   </wsse:UsernameToken>
  </wsse:Security>
 </SOAP-ENV:Header>
 <SOAP-ENV:Body>
  <Discover xmlns="urn:schemas-microsoft-com:xml-analysis"
   SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <RequestType>MDSCHEMA_MEMBERS</RequestType>
   <Restrictions>
    <RestrictionList>
     <CATALOG_NAME>Sample</CATALOG_NAME>
     <CUBE_NAME>Basic</CUBE_NAME>
     <DIMENSION_UNIQUE_NAME>Year</DIMENSION_UNIQUE_NAME>
    </RestrictionList>
   </Restrictions>
   <Properties>
    <PropertyList>
     <DataSourceInfo>
      Provider=Essbase;Data Source=localhost
     </DataSourceInfo>
    </PropertyList>
  </Discover>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

XMLA Rowsets 1641
Response Example (truncated)

```xml
<?xml version="1.0"?>
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
SOAP-ENV:Body>
<m:DiscoverResponse xmlns:m="urn:schemas-microsoft-com:xml-analysis">
  <m:return xsi:type="xsd:string">
    <root xmlns="urn:schemas-microsoft-com:xml-analysis:rowset"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <xsd:schema xmlns="urn:schemas-microsoft-com:xml-analysis:rowset"
        targetNamespace="urn:schemas-microsoft-com:xml-analysis:rowset"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:sql="urn:schemas-microsoft-com:xml-sql"
        elementFormDefault="qualified">
        <xsd:element name="row">
          <xsd:complexType>
            <xsd:sequence minOccurs="0" maxOccurs="unbounded">
              <xsd:element name="CATALOG_NAME" sql:field="CATALOG_NAME"/>
              <xsd:element name="CUBE_NAME" sql:field="CUBE_NAME"/>
              <xsd:element name="DIMENSION_UNIQUE_NAME" sql:field="DIMENSION_UNIQUE_NAME"/>
              <xsd:element name="HIERARCHY_UNIQUE_NAME" sql:field="HIERARCHY_UNIQUE_NAME"/>
              <xsd:element name="LEVEL_UNIQUE_NAME" sql:field="LEVEL_UNIQUE_NAME"/>
              <xsd:element name="LEVEL_NUMBER" sql:field="LEVEL_NUMBER" type="xsd:unsignedInt"/>
              <xsd:element name="GENERATION_NUMBER" sql:field="GENERATION_NUMBER" type="xsd:unsignedInt"/>
              <xsd:element name="MEMBER_ORDINAL" sql:field="MEMBER_ORDINAL" type="xsd:unsignedInt"/>
              <xsd:element name="MEMBER_NAME" sql:field="MEMBER_NAME"/>
              <xsd:element name="MEMBER_UNIQUE_NAME" sql:field="MEMBER_UNIQUE_NAME"/>
              <xsd:element name="MEMBER_TYPE" sql:field="MEMBER_TYPE" type="xsd:int"/>
              <xsd:element name="MEMBER_CAPTION" sql:field="MEMBER_CAPTION" type="xsd:string"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:schema>
    </root>
  </m:return>
</m:DiscoverResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```
MDSCHEMA_PROPERTIES Rowset

The PROPERTIES rowset contains information about the available properties for each level of the dimension, assuming that each level defines a class of members. The properties of all members in this class are the same. For a data store that does not support named levels, a dummy level includes all members in the dimension. The name of this level is the same as the name of the dimension.

The default sort order: PROPERTY_TYPE, CATALOG_NAME, SCHEMA_NAME, CUBE_NAME, DIMENSION_UNIQUE_NAME, HIERARCHY_UNIQUE_NAME, and LEVEL_UNIQUE_NAME.

GUID: MDSCHEMA_PROPERTIES
Table 39 describes the rowset structure.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Essbase Mapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>CATALOG_NAME</td>
<td>Application name</td>
</tr>
<tr>
<td>CUBE_NAME</td>
<td>Database name</td>
</tr>
<tr>
<td>HIERARCHY_UNIQUE_NAME</td>
<td>Dimension name</td>
</tr>
<tr>
<td>LEVEL_UNIQUE_NAME</td>
<td>Dimension name</td>
</tr>
<tr>
<td>PROPERTY_TYPE</td>
<td>1 (MDPROP_MEMBER)</td>
</tr>
<tr>
<td>PROPERTY_NAME</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• For attribute dimension, the name of the dimension is the name of the property</td>
</tr>
<tr>
<td></td>
<td>• For UDA, the UDA name</td>
</tr>
<tr>
<td></td>
<td>• For aliases, the alias name</td>
</tr>
<tr>
<td>PROPERTY_CAPTION</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• For attribute dimensions, the attribute dimension name</td>
</tr>
<tr>
<td></td>
<td>• For UDA, the UDA name</td>
</tr>
<tr>
<td></td>
<td>• For aliases, the alias name</td>
</tr>
<tr>
<td>DATA_TYPE</td>
<td>1 (double) – attribute dimension</td>
</tr>
<tr>
<td></td>
<td>2 (boolean) – attribute dimension</td>
</tr>
<tr>
<td></td>
<td>3 (string) – attribute dimension, UDA or alias</td>
</tr>
<tr>
<td></td>
<td>4 (integer) – attribute dimension</td>
</tr>
<tr>
<td>CHARACTER_MAXIMUM_LENGTH</td>
<td>80 (for UDA or an attribute dimension)</td>
</tr>
<tr>
<td></td>
<td>30 (for alias)</td>
</tr>
<tr>
<td>CHARACTER_OCTET_LENGTH</td>
<td>320 (for UDA or an attribute dimension)</td>
</tr>
<tr>
<td></td>
<td>120 (for alias)</td>
</tr>
<tr>
<td>PROPERTY_ORIGIN</td>
<td>1 (MD_USER_DEFINED)</td>
</tr>
<tr>
<td>PROPERTY_ATTRIBUTE_HIERARCHY_NAME</td>
<td>For attribute dimensions, the attribute dimension name</td>
</tr>
<tr>
<td>PROPERTY_CARDINALITY</td>
<td>ONE (for UDA and aliases)</td>
</tr>
<tr>
<td></td>
<td>MANY (for attribute dimension)</td>
</tr>
<tr>
<td>Column Name</td>
<td>Essbase Mapping</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>PROPERTY_IS_VISIBLE</td>
<td>True</td>
</tr>
</tbody>
</table>

**Request Example**

```xml
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<SOAP-ENV:Body>
<Discover xmlns="urn:schemas-microsoft-com:xml-analysis"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <RequestType>MDSCHEMA_PROPERTIES</RequestType>
  <Restrictions>
    <RestrictionList>
      <CATALOG_NAME>Sample</CATALOG_NAME>
      <CUBE_NAME>Basic</CUBE_NAME>
      <DIMENSION_UNIQUE_NAME>Product</DIMENSION_UNIQUE_NAME>
      <LEVEL_UNIQUE_NAME>SKU</LEVEL_UNIQUE_NAME>
    </RestrictionList>
  </Restrictions>
  <Properties>
    <PropertyList>
      <DataSourceInfo>
        Provider=Essbase;Data Source=localhost
      </DataSourceInfo>
      <Format>Tabular</Format>
    </PropertyList>
  </Properties>
</Discover>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

**Response Example (truncated)**

```xml
<?xml version="1.0"?>
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<SOAP-ENV:Body>
<m:DiscoverResponse xmlns:m="urn:schemas-microsoft-com:xml-analysis">
  <m:return xsi:type="xsd:string">
    <root xmlns="urn:schemas-microsoft-com:xml-analysis:rowset"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <xsd:schema xmlns="urn:schemas-microsoft-com:xml-analysis:rowset"
        targetNamespace="urn:schemas-microsoft-com:xml-analysis:rowset"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:sql="urn:schemas-microsoft-com:xml-sql"
        elementFormDefault="qualified">
        <xsd:element name="root">
          <xsd:complexType>
            <xsd:sequence minOccurs="0" maxOccurs="unbounded">
              <xsd:element name="row" type="row"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:schema>
    </root>
  </m:return>
</m:DiscoverResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```
<table>
<thead>
<tr>
<th>CATALOG_NAME</th>
<th>CUBE_NAME</th>
<th>DIMENSION_UNIQUE_NAME</th>
<th>HIERARCHY_UNIQUE_NAME</th>
<th>LEVEL_UNIQUE_NAME</th>
<th>MEMBER_UNIQUE_NAME</th>
<th>PROPERTY_TYPE</th>
<th>PROPERTY_NAME</th>
<th>PROPERTY_CAPTION</th>
<th>DATA_TYPE</th>
<th>CHARACTER_MAXIMUM_LENGTH</th>
<th>CHARACTER_OCTET_LENGTH</th>
<th>NUMERIC_PRECISION</th>
<th>NUMERIC_SCALE</th>
<th>PROPERTY_CONTENT_TYPE</th>
<th>PROPERTY_ORIGIN</th>
<th>PROPERTY_ATTRIBUTE_HIERARCHY_NAME</th>
<th>PROPERTY_CARDINALITY</th>
<th>MIME_TYPE</th>
<th>PROPERTY_IS_VISIBLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>xsd:short</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
MDSCHEMA_SETS Rowset

The SETS rowset contains information about the sets in a schema (or the catalog, if the provider does not support schemas).

GUID: MDSCHEMA_SETS

Table 40 describes the rowset structure.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Essbase Mapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>CATALOG_NAME</td>
<td>Application name</td>
</tr>
<tr>
<td>CUBE_NAME</td>
<td>Database name</td>
</tr>
<tr>
<td>SET_NAME</td>
<td>Name of the set</td>
</tr>
<tr>
<td>SCOPE</td>
<td>Session</td>
</tr>
</tbody>
</table>

MDSCHEMA_LEVELS Rowset

The LEVELS rowset contains information about the levels available in a dimension.

GUID: MDSCHEMA_LEVELS

Table 41 describes the rowset structure.
<table>
<thead>
<tr>
<th>Column Name</th>
<th>Essbase Mapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>CATALOG_NAME</td>
<td>Application name</td>
</tr>
<tr>
<td>CUBE_NAME</td>
<td>Database name</td>
</tr>
<tr>
<td>DIMENSION_UNIQUE_NAME</td>
<td>Name of the dimension to which the level belongs</td>
</tr>
<tr>
<td>HIERARCHY_UNIQUE_NAME</td>
<td>Name of the dimension to which the level belongs</td>
</tr>
<tr>
<td>LEVEL_NAME</td>
<td>Unique level name</td>
</tr>
<tr>
<td>LEVEL_UNIQUE_NAME</td>
<td>Unique level name</td>
</tr>
<tr>
<td>LEVEL_CAPTION</td>
<td>Level name</td>
</tr>
<tr>
<td>LEVEL_NUMBER</td>
<td>Level number</td>
</tr>
<tr>
<td>LEVEL_CARDINALITY</td>
<td>Number of members in the level</td>
</tr>
<tr>
<td>LEVEL_TYPE</td>
<td>MDLEVEL_TYPE_ALL (for dimension level)</td>
</tr>
<tr>
<td></td>
<td>MDLEVEL_TYPE_TIME (for dimension type TIME)</td>
</tr>
<tr>
<td></td>
<td>MDLEVEL_TYPE_REGULAR (for all others)</td>
</tr>
<tr>
<td>LEVEL_UNIQUE_SETTINGS</td>
<td>2 (MDDIMENSIONS_MEMBER_NAME_UNIQUE)</td>
</tr>
<tr>
<td>LEVEL_IS_VISIBLE</td>
<td>True</td>
</tr>
<tr>
<td>ESSBASE_GEN_UNIQUE_NAME</td>
<td>Generation unique name</td>
</tr>
<tr>
<td>ESSBASE_GEN_CAPTION</td>
<td>Generation caption</td>
</tr>
</tbody>
</table>

**Request Example**

```xml
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<SOAP-ENV:Body>
   <Discover xmlns="urn:schemas-microsoft-com:xml-analysis"
             SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <RequestType>MDSCHEMA_LEVELS</RequestType>
   <Restrictions>
     <RestrictionList>
       <CATALOG_NAME>Sample</CATALOG_NAME>
       <CUBE_NAME>Basic</CUBE_NAME>
       <DIMENSION_UNIQUE_NAME>Year</DIMENSION_UNIQUE_NAME>
     </RestrictionList>
   </Restrictions>
   <Properties>
     <PropertyList>
       <DataSourceInfo>Provider=Essbase;Data Source=localhost</DataSourceInfo>
       <Format>Tabular</Format>
     </PropertyList>
   </Properties>
</SOAP-ENV:Body>
```
Response Example

```xml
<?xml version="1.0"?>
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
    <m:DiscoverResponse xmlns:m="urn:schemas-microsoft-com:xml-analysis">
      <m:return xsi:type="xsd:string">
        <root xmlns="urn:schemas-microsoft-com:xml-analysis:rowset"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xmlns:xsd="http://www.w3.org/2001/XMLSchema">
          <xsd:schema xmlns="urn:schemas-microsoft-com:xml-analysis:rowset"
            targetNamespace="urn:schemas-microsoft-com:xml-analysis:rowset"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:sql="urn:schemas-microsoft-com:xml-sql"
            elementFormDefault="qualified">
            <xsd:element name="root">
              <xsd:complexType>
                <xsd:sequence minOccurs="0" maxOccurs="unbounded">
                  <xsd:element name="row" type="xsd:row"/>
                </xsd:sequence>
              </xsd:complexType>
            </xsd:element>
            <xsd:complexType name="row">
              <xsd:sequence maxOccurs="unbounded" minOccurs="0">
                <xsd:element name="CATALOG_NAME" type="xsd:string"
                  sql:field="CATALOG_NAME"/>
                <xsd:element name="CUBE_NAME" type="xsd:string"
                  sql:field="CUBE_NAME"/>
                <xsd:element name="DIMENSION_UNIQUE_NAME" type="xsd:string"
                  sql:field="DIMENSION_UNIQUE_NAME"/>
                <xsd:element name="HIERARCHY_UNIQUE_NAME" type="xsd:string"
                  sql:field="HIERARCHY_UNIQUE_NAME"/>
                <xsd:element name="LEVEL_NAME" type="xsd:string"
                  sql:field="LEVEL_NAME"/>
                <xsd:element name="LEVEL_UNIQUE_NAME" type="xsd:string"
                  sql:field="LEVEL_UNIQUE_NAME"/>
                <xsd:element name="LEVEL_CAPTION" type="xsd:string"
                  sql:field="LEVEL_CAPTION"/>
                <xsd:element name="LEVEL_NUMBER" type="xsd:unsignedInt"
                  sql:field="LEVEL_NUMBER"/>
                <xsd:element name="LEVEL_CARDINALITY" type="xsd:unsignedInt"
                  sql:field="LEVEL_CARDINALITY"/>
                <xsd:element name="LEVEL_TYPE" type="xsd:int"
                  sql:field="LEVEL_TYPE"/>
                <xsd:element name="LEVEL_UNIQUE_SETTINGS" type="xsd:int"
                  sql:field="LEVEL_UNIQUE_SETTINGS"/>
                <xsd:element name="LEVEL_IS_VISIBLE" type="xsd:boolean"
                  sql:field="LEVEL_IS_VISIBLE"/>
                <xsd:element name="DESCRIPTION" type="xsd:string"
                  sql:field="DESCRIPTION" minOccurs="0"/>
              </xsd:sequence>
            </xsd:complexType>
          </xsd:schema>
        </root>
      </m:return>
    </m:DiscoverResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```
DISCOVER_SCHEMA_ROWSETS Rowset

GUID: DISCOVER_SCHEMA_ROWSETS

Table 42 describes the rowset structure.

Table 42  DISCOVER_SCHEMA Rowset Structure

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Essbase Mapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>SchemaName</td>
<td>The name of the schema/request. This returns the values in the RequestTypes enumeration, plus any additional types supported by the provider. The provider defines rowset structures for the additional types.</td>
</tr>
<tr>
<td>Restrictions</td>
<td>List of restrictions allowed</td>
</tr>
<tr>
<td>Description</td>
<td>Description of the schema</td>
</tr>
</tbody>
</table>

DISCOVER_DATASOURCES Rowset

GUID: DISCOVER_DATASOURCES

Table 43 describes the rowset structure.
Table 43  DISCOVER_DATASOURCES Rowset Structure

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Essbase Mapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataSourceName</td>
<td>Name of the data source</td>
</tr>
<tr>
<td>DataSourceDescription</td>
<td>Description of the data source</td>
</tr>
<tr>
<td>DataSourceInfo</td>
<td>Provider=Essbase Data Source= name of the Analytic Server</td>
</tr>
<tr>
<td>ProviderName</td>
<td>XMLA for Essbase</td>
</tr>
<tr>
<td>ProviderType</td>
<td>MDP</td>
</tr>
<tr>
<td>AuthenticationMode</td>
<td>Authenticated</td>
</tr>
</tbody>
</table>

DISCOVER_PROPERTIES Rowset

GUID: DISCOVER_PROPERTIES

Table 44 describes the rowset structure.

Table 44  DISCOVER_PROPERTIES Rowset Structure

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Essbase Mapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>PropertyName</td>
<td>Name of the property</td>
</tr>
<tr>
<td>PropertyDescription</td>
<td>Description of the property</td>
</tr>
<tr>
<td>PropertyType</td>
<td>XML data type of the property.</td>
</tr>
<tr>
<td>PropertyAccessType</td>
<td>Access for the property. The value can be Read, Write, or ReadWrite</td>
</tr>
<tr>
<td>IsRequired</td>
<td>True if a property is required, false if it is not required</td>
</tr>
<tr>
<td>Value</td>
<td>Current value of the property</td>
</tr>
</tbody>
</table>

Request Example

```xml
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <SOAP-ENV:Body>
  <Discover xmlns="urn:schemas-microsoft-com:xml-analysis"
 SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <RequestType>DISCOVER_PROPERTIES</RequestType>
   <Restrictions>
    <RestrictionList></RestrictionList>
   </Restrictions>
   <Properties>
    <PropertyList>
     <DataSourceInfo>Provider=Essbase; Data Source=localhost</DataSourceInfo>
     <Format>Tabular</Format>
    </PropertyList>
  </Discover>
</SOAP-ENV:Body>
```
Response Example

<?xml version="1.0"?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
    <m:DiscoverResponse xmlns:m="urn:schemas-microsoft-com:xml-analysis">
      <m:return xsi:type="xsd:string">
        <root xmlns="urn:schemas-microsoft-com:xml-analysis:rowset"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema">
          <xsd:schema xmlns="urn:schemas-microsoft-com:xml-analysis:rowset"
              targetNamespace="urn:schemas-microsoft-com:xml-analysis:rowset"
              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xmlns:xsd="http://www.w3.org/2001/XMLSchema"
              xmlns:sql="urn:schemas-microsoft-com:xml-sql"
              elementFormDefault="qualified">
            <xsd:element name="root">
              <xsd:complexType>
                <xsd:sequence minOccurs="0" maxOccurs="unbounded">
                  <xsd:element name="row" type="xsd:string"/>
                </xsd:sequence>
              </xsd:complexType>
            </xsd:element>
            <xsd:complexType name="row">
              <xsd:sequence maxOccurs="unbounded" minOccurs="0">
                <xsd:element name="PropertyName" type="xsd:string" sql:field="PropertyName"/>
                <xsd:element name="PropertyDescription" type="xsd:string" sql:field="PropertyDescription"/>
                <xsd:element name="PropertyType" type="xsd:string" sql:field="PropertyType"/>
                <xsd:element name="PropertyAccessType" type="xsd:string" sql:field="PropertyAccessType"/>
                <xsd:element name="IsRequired" type="xsd:boolean" sql:field="IsRequired"/>
                <xsd:element name="Value" type="xsd:string" sql:field="Value"/>
              </xsd:sequence>
            </xsd:complexType>
          </xsd:schema>
          <row>
            <PropertyName>ProviderName</PropertyName>
            <PropertyDescription>The name of the Analytic Services Provider</PropertyDescription>
            <PropertyType>string</PropertyType>
            <PropertyAccessType>Read</PropertyAccessType>
            <IsRequired>false</IsRequired>
            <Value>Analytic Services XML for Analysis Provider</Value>
          </row>
        </root>
      </m:return>
    </m:DiscoverResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
DISCOVER_ENUMERATORS Rowset

GUID: DISCOVER_ENUMERATORS

Table 45 describes the rowset structure.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Essbase Mapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>EnumName</td>
<td>Name of the enumerator that contains a set of values</td>
</tr>
<tr>
<td>EnumDescription</td>
<td>Description of the enumerator</td>
</tr>
<tr>
<td>ElementName</td>
<td>Name of one of the value elements in the enumerator set Example: TDP</td>
</tr>
<tr>
<td>ElementDescription</td>
<td>Description of the element</td>
</tr>
<tr>
<td>EnumType</td>
<td>Data type of the Enum values</td>
</tr>
<tr>
<td>ElementValue</td>
<td>Value of the element</td>
</tr>
<tr>
<td></td>
<td>Example: 01</td>
</tr>
</tbody>
</table>

Request Example

```xml
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <SOAP-ENV:Body>
  <Discover xmlns="urn:schemas-microsoft-com:xml-analysis"
   SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <RequestType>DISCOVER_ENUMERATORS</RequestType>
   <Restrictions>
    <RestrictionList>
   </RestrictionList>
   </Restrictions>
   <Properties>
    <PropertyList>
     <DataSourceInfo>
      Provider=Essbase;Data Source=localhost
     </DataSourceInfo>
     <Format>Tabular</Format>
    </PropertyList>
   </Properties>
  </Discover>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```
Response Example

```xml
<?xml version="1.0"?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<SOAP-ENV:Body>
<m:DiscoverResponse xmlns:m="urn:schemas-microsoft-com:xml-analysis">
  <m:return>
    <root>
      <EnumName>ProviderType</EnumName>
      <ElementName>TDP</ElementName>
      <EnumType>string</EnumType>
    </root>
    ................More Rows.............
  </m:return>
</m:DiscoverResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```
DISCOVER_KEYWORDS Rowset

GUID: DISCOVER_KEYWORDS

Table 46 describes the rowset structure.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Essbase Mapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keyword</td>
<td>A list of keywords reserved by a provider</td>
</tr>
<tr>
<td></td>
<td>Example: AND</td>
</tr>
</tbody>
</table>

Request Example

```xml
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<SOAP-ENV:Body>
<Discover xmlns="urn:schemas-microsoft-com:xml-analysis"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<RequestType>DISCOVER_KEYWORDS</RequestType>
<Restrictions>
<RestrictionList></RestrictionList>
</Restrictions>
<Properties>
<PropertyList>
<DataSourceInfo>
Provider=Essbase;Data Source=localhost
</DataSourceInfo>
<Format>Tabular</Format>
</PropertyList>
</Properties>
</Discover>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Response Example

```xml
<?xml version="1.0"?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<SOAP-ENV:Body>
<m:DiscoverResponse
xmlns:m="urn:schemas-microsoft-com:xml-analysis">
<m:return xsi:type="xsd:string"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<root xmlns="urn:schemas-microsoft-com:xml-analysis:rowset"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"/>
</root>
</m:return>
</m:DiscoverResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```
DISCOVER_LITERALS Rowset

GUID: DISCOVER_LITERALS

The section called “Example 1” describes the rowset structure.

Table 47  DISCOVER_LITERALS Rowset Structure

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Essbase Mapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>LiteralName</td>
<td>Name of the literal described in the row&lt;br&gt;Example: DBLITERAL_LIKE_PERCENT</td>
</tr>
<tr>
<td>LiteralValue</td>
<td>Contains the literal value&lt;br&gt;Example, if LiteralName is DBLITERAL_LIKE_PERCENT and the percent character (%) is used to match zero or more characters in a LIKE clause, this column’s value would be “%.”</td>
</tr>
<tr>
<td>LiteralInvalidChars</td>
<td>Characters, in the literal, that are not valid&lt;br&gt;Example: If table names can contain anything other than a numeric character, this string would be “0123456789”</td>
</tr>
<tr>
<td>LiteralInvalidStartingChars</td>
<td>Characters that are not valid as the first character of the literal. If the literal can start with any valid character, this is null.</td>
</tr>
<tr>
<td>LiteralMaxLength</td>
<td>Maximum number of characters in the literal. If there is no maximum or the maximum is unknown, the value is -1.</td>
</tr>
</tbody>
</table>
**Flattened Rowset Examples**

Flattening a rowset is a way to present multidimensional data in a grid. This two-dimensional, tabular presentation of data can facilitate understanding of the output of a multidimensional XMLA request.

**MDX Examples**

The following examples illustrate flattened rowsets as MDX queries and results. MDX is used for ease of presentation; however, the example queries are intended to be considered in terms of XMLA SOAP requests. Remember that in XMLA, level 0 represents a dimension, rather than a leaf member, as in MDX. Therefore, although these examples are in MDX, the levels are reversed as if they were in XMLA.

**Example 1**

The following query requests all members of level 1.

```mdx
SELECT NON EMPTY {[Profit]} ON COLUMNS,
     NON EMPTY [Product].Levels(1).ALLMEMBERS ON ROWS
FROM Sample.Basic
```

This query has the following result:

<table>
<thead>
<tr>
<th>[Product].[Family].[MEMBER_CAPTION]</th>
<th>[Profit]</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>30468</td>
</tr>
<tr>
<td>200</td>
<td>27954</td>
</tr>
<tr>
<td>300</td>
<td>25799</td>
</tr>
<tr>
<td>400</td>
<td>21301</td>
</tr>
<tr>
<td>Diet</td>
<td>28826</td>
</tr>
</tbody>
</table>

**Example 2**

The following query requests a maximum of two levels. The flattening of rowsets includes level 1 in this request for levels(2). When using flattened rowsets, if you query for level N, levels 1 through N are returned.

```mdx
SELECT NON EMPTY {[Profit]} ON COLUMNS,
     NON EMPTY [Product].Levels(2).ALLMEMBERS ON ROWS
FROM Sample.Basic
```

This query has the following result (truncated):
### Example 3

The following query builds on the previous, and also asks for the result set to include the member unique name and level number properties for the set of levels 1 through N, where N=2. Each member and each property is allotted a row.

```sql
SELECT NON EMPTY {[Profit]} ON COLUMNS,
       NON EMPTY [Product].Levels(2).ALLMEMBERS
       DIMENSION PROPERTIES MEMBER_UNIQUE_NAME, LEVEL_NUMBER
ON ROWS
FROM Sample.Basic
```

This query has the following result (truncated):

<table>
<thead>
<tr>
<th>[Product].[Family].[MEMBER_CAPTION]</th>
<th>[Product].[SKU].[MEMBER_CAPTION]</th>
<th>[Profit]</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>100–10</td>
<td>22777</td>
</tr>
<tr>
<td>100</td>
<td>100–20</td>
<td>5708</td>
</tr>
<tr>
<td>100</td>
<td>100–30</td>
<td>1983</td>
</tr>
<tr>
<td>200</td>
<td>200–10</td>
<td>7201</td>
</tr>
<tr>
<td>200</td>
<td>200–20</td>
<td>12025</td>
</tr>
<tr>
<td>200</td>
<td>200–30</td>
<td>4636</td>
</tr>
<tr>
<td>200</td>
<td>200–40</td>
<td>4092</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Example 4

By implementing CrossJoin in a flattened rowsets query, you can use multiple dimensions (at least two). In this example, Market and Product dimensions are requested. For each dimension, the same logic as in previous examples applies: Each dimension, level, and property is allotted one column (in this case, one level and one property are requested).

```sql
SELECT NON EMPTY { [Profit] } ON COLUMNS,
       NON EMPTY Crossjoin ([Market].Levels(1).AllMembers,[Product].Levels(1).ALLMEMBERS)
       DIMENSION PROPERTIES MEMBER_CAPTION
ON ROWS
FROM Sample.Basic
```

This query has the following result (truncated):

<table>
<thead>
<tr>
<th>[Market]. Levels(1). [MEMBER_CAPTION]</th>
<th>[Product]. [Family]. [MEMBER_CAPTION]</th>
<th>[Profit]</th>
</tr>
</thead>
<tbody>
<tr>
<td>East</td>
<td>Colas</td>
<td>12656</td>
</tr>
<tr>
<td>East</td>
<td>Root Beer</td>
<td>2534</td>
</tr>
<tr>
<td>East</td>
<td>Cream Soda</td>
<td>2627</td>
</tr>
<tr>
<td>East</td>
<td>Fruit Soda</td>
<td>6344</td>
</tr>
<tr>
<td>East</td>
<td>Diet Drinks</td>
<td>2408</td>
</tr>
<tr>
<td>West</td>
<td>Colas</td>
<td>3549</td>
</tr>
<tr>
<td>West</td>
<td>Root Beer</td>
<td>9727</td>
</tr>
<tr>
<td>West</td>
<td>Cream Soda</td>
<td>10731</td>
</tr>
<tr>
<td>West</td>
<td>Fruit Soda</td>
<td>5854</td>
</tr>
<tr>
<td>West</td>
<td>Diet Drinks</td>
<td>8087</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Example 5

In this example, CrossJoin is used to request levels 1–2 for Market and Product.

```sql
SELECT NON EMPTY { [Profit] } ON COLUMNS,
       NON EMPTY Crossjoin ([Market].Levels(2).AllMembers,[Product].Levels(2).ALLMEMBERS)
       DIMENSION PROPERTIES MEMBER_CAPTION
ON ROWS
FROM Sample.Basic
```
This query has the following result (truncated):

```
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>East</td>
<td>New York</td>
<td>Colas</td>
<td>Cola</td>
<td>3498</td>
</tr>
<tr>
<td>East</td>
<td>New York</td>
<td>Root Beer</td>
<td>Old Fashioned</td>
<td>-2594</td>
</tr>
<tr>
<td>East</td>
<td>New York</td>
<td>Root Beer</td>
<td>Birch Beer</td>
<td>3086</td>
</tr>
<tr>
<td>East</td>
<td>New York</td>
<td>Cream Soda</td>
<td>Dark Cream</td>
<td>2496</td>
</tr>
<tr>
<td>East</td>
<td>New York</td>
<td>Cream Drinks</td>
<td>Vanilla Cream</td>
<td>-1952</td>
</tr>
<tr>
<td>East</td>
<td>New York</td>
<td>Fruit Soda</td>
<td>Grape</td>
<td>1329</td>
</tr>
<tr>
<td>East</td>
<td>New York</td>
<td>Fruit Soda</td>
<td>Orange</td>
<td>1388</td>
</tr>
<tr>
<td>East</td>
<td>New York</td>
<td>Fruit Soda</td>
<td>Strawberry</td>
<td>951</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
```

**Example 6**

The following example uses CrossJoin to represent multiple dimensions, requests a different number of levels for each dimension, and requests multiple properties.

```
SELECT NON EMPTY { [Profit] } ON COLUMNS,
NON EMPTY Crossjoin ([Market].Levels(1).AllMembers, [Product].Levels(2).ALLMEMBERS)
    DIMENSION PROPERTIES MEMBER_CAPTION, LEVEL_NUMBER
ON ROWS
FROM Sample.Basic
```

This query has the following result (truncated):

```
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>East</td>
<td>1</td>
<td>Colas</td>
<td>Diet Cola</td>
<td>2</td>
<td>11129</td>
</tr>
<tr>
<td>East</td>
<td>1</td>
<td>Colas</td>
<td>Diet Cola</td>
<td>2</td>
<td>1114</td>
</tr>
<tr>
<td>East</td>
<td>1</td>
<td>Colas</td>
<td>Caffeine Free Cola</td>
<td>2</td>
<td>413</td>
</tr>
<tr>
<td>East</td>
<td>1</td>
<td>Root Beer</td>
<td>Old Fashioned</td>
<td>2</td>
<td>-2540</td>
</tr>
<tr>
<td>East</td>
<td>1</td>
<td>Root Beer</td>
<td>Diet Root Beer</td>
<td>2</td>
<td>982</td>
</tr>
<tr>
<td>East</td>
<td>1</td>
<td>Root Beer</td>
<td>Birch Beer</td>
<td>2</td>
<td>4092</td>
</tr>
<tr>
<td>East</td>
<td>1</td>
<td>Cream Soda</td>
<td>Dark Cream</td>
<td>2</td>
<td>3233</td>
</tr>
<tr>
<td>East</td>
<td>1</td>
<td>Cream Soda</td>
<td>Vanilla Cream</td>
<td>2</td>
<td>-918</td>
</tr>
</tbody>
</table>
```

1660 Working with XMLA
Example 7

The following example uses multiple, nested CrossJoins.

```
SELECT NON EMPTY { [Profit] } ON COLUMNS,
     NON EMPTY (CROSSJOIN
                     (CROSSJOIN( [Market].Levels(1).ALLMEMBERS,
                                  [Product].[Family].ALLMEMBERS ),
                      [Year].Levels(1).ALLMEMBERS ),
         )
    ) DIMENSION PROPERTIES MEMBER_CAPTION
ON ROWS FROM Sample.Basic
```

This query has the following result (truncated):

<table>
<thead>
<tr>
<th>[Market]. Levels(1). [MEMBER_CAPTION]</th>
<th>[Product]. [Family]. [MEMBER_CAPTION]</th>
<th>[Year]. Levels(1). [MEMBER_CAPTION]</th>
<th>[Profit]</th>
</tr>
</thead>
<tbody>
<tr>
<td>East</td>
<td>Colas</td>
<td>Qtr1</td>
<td>2747</td>
</tr>
<tr>
<td>East</td>
<td>Colas</td>
<td>Qtr2</td>
<td>3352</td>
</tr>
<tr>
<td>East</td>
<td>Colas</td>
<td>Qtr3</td>
<td>3740</td>
</tr>
<tr>
<td>East</td>
<td>Colas</td>
<td>Qtr4</td>
<td>2817</td>
</tr>
<tr>
<td>East</td>
<td>Root Beer</td>
<td>Qtr1</td>
<td>562</td>
</tr>
<tr>
<td>East</td>
<td>Root Beer</td>
<td>Qtr2</td>
<td>610</td>
</tr>
<tr>
<td>East</td>
<td>Root Beer</td>
<td>Qtr3</td>
<td>372</td>
</tr>
<tr>
<td>East</td>
<td>Root Beer</td>
<td>Qtr4</td>
<td>990</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

XMLA Examples

The following examples illustrate an XMLA response and request.

This is an example of a flattened rowset request. To flatten the result, you must use Tabular format in the PropertyList element, as shown in the example.

```
     xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```
An example of a flattened rowset response:

```xml
<?xml version="1.0"?>
                 xmlns:SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
    <m:ExecuteResponse xmlns:m="urn:schemas-microsoft-com:xml-analysis">
      <m:return
          SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
        <root xmlns="urn:schemas-microsoft-com:xml-analysis:rowset"
              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xmlns:xsd="http://www.w3.org/2001/XMLSchema">
          <xsd:schema xmlns="urn:schemas-microsoft-com:xml-analysis:rowset"
                      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                      targetNamespace="urn:schemas-microsoft-com:xml-analysis:rowset"
                      xmlns:sql="urn:schemas-microsoft-com:xml-sql"
                      elementFormDefault="qualified">
            <xsd:element name="root">
              <xsd:complexType>
                <xsd:sequence minOccurs="0" maxOccurs="unbounded">
                  <xsd:element name="row" type="xsd:string" sql:field="[Year].Levels(1).[MEMBER_CAPTION]"><xsd:element name="column2" type="xsd:string" sql:field="[Year].Levels(2).[MEMBER_CAPTION]" minOccurs="0"/>
                  <xsd:element name="column3" type="xsd:double" sql:field="[Profit]" minOccurs="0"/>
                </xsd:sequence>
              </xsd:complexType>
            </xsd:element><![CDATA[
```

1662 Working with XMLA
<table>
<thead>
<tr>
<th>Quarter</th>
<th>Month</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Qtr1</td>
<td>Jan</td>
<td>8024.00</td>
</tr>
<tr>
<td></td>
<td>Feb</td>
<td>8346.00</td>
</tr>
<tr>
<td></td>
<td>Mar</td>
<td>8333.00</td>
</tr>
<tr>
<td>Qtr2</td>
<td>Apr</td>
<td>8644.00</td>
</tr>
<tr>
<td></td>
<td>May</td>
<td>8929.00</td>
</tr>
<tr>
<td></td>
<td>Jun</td>
<td>9534.00</td>
</tr>
<tr>
<td>Qtr3</td>
<td>Jul</td>
<td>9878.00</td>
</tr>
<tr>
<td></td>
<td>Aug</td>
<td>9545.00</td>
</tr>
<tr>
<td>Quarter</td>
<td>Month</td>
<td>Amount</td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
<td>----------</td>
</tr>
<tr>
<td>Qtr3</td>
<td>Sep</td>
<td>8489.00</td>
</tr>
<tr>
<td>Qtr4</td>
<td>Oct</td>
<td>8653.00</td>
</tr>
<tr>
<td>Qtr4</td>
<td>Nov</td>
<td>8367.00</td>
</tr>
<tr>
<td>Qtr4</td>
<td>Dec</td>
<td>8780.00</td>
</tr>
<tr>
<td></td>
<td>calctest</td>
<td>4.00</td>
</tr>
</tbody>
</table>
Sample C API Program 1 (cs1.c)

This file contains an annotated Essbase C API program. This fundamental sample program can be used in a C++ programming environment as a starting point for more functional programs.

This file is to be used with the Oracle Essbase API Reference to illustrate basic points in API programming. A complete set of actual C code files is also included with the Essbase API. Look in the samples directory of this documentation for the *.c files, executables, projects, and workspaces.

/*
Copyright 1992-2008 Oracle Corporation. All Rights Reserved.

NAME
  cs1.c

DEPENDENCIES
You must add ESSAPIN.LIB to your project.
You must also identify the /API/Include and /API/Lib directories to the compiler/linker.

DESCRIPTION
This file is used for testing of the Main API and describing the most fundamental aspects of the Essbase API.
This simple application program is intended as a starting point for more complex programs. This program performs only the most basic initialization and login functions. It connects to a server/application/database, performs only the most basic of tasks (lists connected users), disconnects, logs out and terminates. Because all Essbase API programs must do these things, this program represents the most simple API program possible. It is applicable in the most general sense to being used as a starting point for
more useful and complex production-oriented programs.

NOTES
This program has three sections:
1 - The includes and function definitions
2 - The function declarations
3 - The main flow

MODIFIED
* Created 26 Aug 1999 publications

*/
*******************************************************************************/
*******************************************************************************/
*******************************************************************************/

/*
Declaration of Include files
*/

#if defined _WIN32 || defined _WINDOWS
#include <windows.h>
#endif
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#pragma pack (1)
#include <essapi.h>
#include <essotl.h>
#pragma pack ()

/*
Declaration of handles and connection information variables
*/

ESS_HINST_T hInst;
ESS_HCTX_T hCtx;
ESS_SVRNAME_T srvrName = "";
ESS_USERNAME_T userName = "";
ESS_PASSWORD_T pswd = "";

/*
Declaration of all the Essbase API functions used in this program. You could declare all the functions here, and have them available for the prototype section. This program only uses a few functions.
*/

/* Initialization and Login functions */
void ESS_Init();
void ESS_AutoLogin();
void ESS_Login(); //This app uses EssAutoLogin().
void ESS_LoginSetPassword(); //I declared these other logMODE  
void ESS_AutoLoginSetPassword(); //functions for future use.
void ESS_Logout();
void ESS_Term();
void ESS_GetVersion();
void ESS_GetAPIVersion();
void ESS_SetActive();
void ESS_ListDatabases();
void ESS_ListUsers();
void ESS_Free();

/************* START FUNCTION DECLARATIONS ******************/
/******* make sure you are in the correct API version ******/
/************* END FUNCTION DECLARATIONS ********************/

void ESS_Init()
{
    ESS_STS_T sta;
    ESS_INIT_T InitStruct = {ESS_API_VERSION,
                             NULL,
                             0L,
                             255,
                             NULL,
                             NULL,
                             NULL,
                             NULL,
                             NULL,
                             NULL,
                             0L};
    if ((sta = EssInit(&InitStruct, &hInst)) != ESS_STS_NOERR)
    {
        printf("EssInit failure: %ld\n", sta);
        exit((int) sta);
    }
    printf("EssInit sts: %ld\n", sta);
}

Sample C API Program 1 (cs1.c)
exit ((int) sts);
}
}

void ESS_AutoLogin()
{

ESS_STS_T sts = ESS_STS_NOERR;
ESS_CHAR_T SvrName[ESS_SVRNAMELEN]; //this is different in VC++6
ESS_CHAR_T UserName[ESS_USERNAMELEN];
ESS_CHAR_T Password[ESS_PASSWORDLEN];
ESS_CHAR_T AppName[ESS_APPNAMELEN];
ESS_CHAR_T DbName[ESS_DBNAMELEN];

ESS_USHORT_T Option;
ESS_ACCESS_T Access;

//ESS_HCTX_T hCtx; Don't set this again, it is set in EssInit

/* Initialize parameters */
strcpy(SvrName,"localhost");
strcpy(UserName,"Admin");
strcpy(Password,"Password");
strcpy(AppName,""ственно
strcpy(DbName,""ственно
Option = AUTO_DEFAULT;

/* Login to Essbase Server */
sts = EssAutoLogin (hInst, SvrName, UserName, Password,
AppName, DbName, Option, &Access, &hCtx);
printf("EssAutoLogin sts: %ld\r\n", sts);
}

void ESS_LoginSetPassword()
{

ESS_STS_T sts = ESS_STS_NOERR;
ESS_USHORT_T Items;
ESS_PAPPDB_T pAppsDbs = NULL;

ESS_PASSWORD_T newPswd = "password2";

sts = EssLoginSetPassword (hInst, srvrName, userName, pswd, newPswd,
&Items, &pAppsDbs, &hCtx);
printf("EssLoginSetPassword sts: %ld\r\n", sts);
if (sts)
{
    printf("\n\tEssLoginSetPassword sts: %ld\n",sts);
    exit ((int) sts);
}

}
void ESS_GetAPIVersion()
{
    ESS_STS_T    sts = ESS_STS_NOERR;
    ESS_ULONG_T  Version;

    sts = EssGetAPIVersion(&Version);
    if(!sts)
        printf("API Version %#x\n", Version);
}

void ESS_Term()
{
    ESS_STS_T sts = ESS_STS_NOERR;
    if ((sts = EssTerm(hInst)) != ESS_STS_NOERR)
    {
        /* error terminating API */
        exit((ESS_USHORT_T) sts);
    }
    printf("EssTerm sts: %ld\r\n", sts);
}

void ESS_Logout()
{
    ESS_STS_T sts = ESS_STS_NOERR;

    sts = EssLogout (hCtx);
    printf("\n\nEssLogout sts: %ld\r\n", sts);
}

void ESS_GetVersion()
{
    ESS_STS_T sts = ESS_STS_NOERR;
    ESS_USHORT_T Release;
    ESS_USHORT_T Version;
    ESS_USHORT_T Revision;

    sts = EssGetVersion (hCtx, &Release, &Version, &Revision);
    printf("EssGetVersion sts: %ld\r\n", sts);
    if(!sts)
    {
        printf("\r\nEssbase Application Server - ");
        printf("Version %d.%d.%d\r\n", Release, Version, Revision);
    }
}
void ESSSetActive()
{
    ESS_STS_T sts = ESS_STS_NOERR;
    ESS_ACCESS_T Access;
    ESS_STR_T AppName;
    ESS_STR_T DbName;

    AppName = "sample";
    DbName = "basic";
    sts = EssSetActive(hCtx, AppName, DbName, &Access);
    printf("EssSetActive sts: %ld\n",sts);
}

void ESSListDatabases()
{
    ESS_STS_T sts = ESS_STS_NOERR;
    ESS_USHORT_T Items;
    ESS_USHORT_T ind;
    ESS_PAPPDB_T pAppsDbs = NULL;

    sts = EssListDatabases(hCtx, NULL, &Items, &pAppsDbs);
    printf("EssListDatabases sts: %ld\n",sts);

    if(!sts)
    {
        if(Items && pAppsDbs)
        {
            printf("\n--Applications/databases available--\n--\n");
            for (ind = 0; ind<Items; ind++)
            {
                if((pAppsDbs+ind) !=NULL)
                {
                    if((pAppsDbs[ind].AppName != NULL)
                        && (pAppsDbs[ind].DbName != NULL))
                    {
                        printf("%s",pAppsDbs[ind].AppName);
                        printf(" ==> ");
                        printf("%s",pAppsDbs[ind].DbName);
                        printf("\n\n");
                    }
                }
            }
            EssFree(hInst, pAppsDbs);
        }
        else
        {
            printf("\nDatabase List is Empty\n\n");
        }
    }
}
void ESS_ListUsers()
{

    ESS_STS_T    sts;
    ESS_USHORT_T  Count;
    ESS_PUSERINFO_T Users = NULL;
    ESS_USHORT_T   ind;

    sts = EssListUsers (hCtx, NULL, NULL, &Count, &Users);
    if (!sts)
    {
        if (Count && Users)
        {
            if (Count && Users)
            {
                printf ("\r\n-----User List from EssListUsers()-----\r\n\n");
                for (ind = 0; ind < Count; ind++)
                {
                    printf ("Name->%s	Application->%s	database->%s\r\n",
                            Users[ind].Name, Users[ind].AppName,
                            Users[ind].DbName);
                    // printf("Login %d\r\n",Users[ind].Login);
                    // printf("Type %d\r\n",Users[ind].Type);
                    // printf("Access %d\r\n",Users[ind].Access);
                    // printf("MaxAccess %d\r\n",Users[ind].MaxAccess);
                    // printf("Expiration %d\r\n",Users[ind].Expiration);
                    // printf("LastLogin %d\r\n",Users[ind].LastLogin);
                    // printf("FailCount %d\r\n",Users[ind].FailCount);
                    // printf("LoginId %ld\r\n",Users[ind].LoginId);
                    }
                    // printf("end of userlist %d\", count);
                printf ("\r\n-----User List from EssListUsers()-----\r\n\n"");
                EssFree (hInst, Users);
                printf("\r\n");
            }
            else
            {
                printf ("\r\nUsers list is empty\r\n\n");
            }
        }
    }

}
handle (hCtx). The EssLogin is required to connect to a
database/application. Almost any functions can follow the Init and Login.
We used EssAutoLogin to display the Connect dialog box, but this
program could have used EssLogin and retrieve the Username and
Password as command line arguments. Following sample programs will
illustrate the use of command line arguments.

The following statements perform some of the most simple actions. The
output, in the form of printf statements, is done by the individual
functions. The EssFree functions that release allocated memory are
also in the individual functions. More complex programs will not free
memory in the individual functions because the allocated structures
and handles are needed until the end.

These simple actions can easily be more complex. Additional operations
would be added in this section. Following sample programs will do more,
but this program merely retrieves some basic information and displays it.

ESS_GetVersion();
ESS_GetAPIVersion();
ESS_SetActive();
ESS_ListDatabases();
ESS_ListUsers();

The EssLogout disconnects the user from the Essbase Server, application,
and database. The EssTerm ends the program and frees allocated memory,
such as the context handle.

ESS_Logout();
ESS_Term();
}

End of program

Sample C API Program 2 (cs2.c)

This file contains an annotated Essbase C API program. This fundamental sample program can
be used in a C++ programming environment as a starting point for more functional programs.

This file is to be used with the Oracle Essbase API Reference to illustrate basic points in API
programming. A complete set of actual C code files is also included with the Essbase API. Look
in the samples directory for the *.c files, executables, projects, and workspaces.

Copyright 1992-2008 Oracle Corporation. All Rights Reserved.

NAME
cs2.c

DEPENDENCIES
DESCRIPTION
This file is used as an example of a simple applications program. This program performs basic initialization and login and queries the active application/database. It then manipulates the user list, adding, renaming, and deleting a new user.

NOTES
This program has three sections:
1 - The includes and function definitions
2 - The function declarations
3 - The main flow

MODIFIED
* Modified 03 Sep 1999 Publications
*/
/*****************************************************************************/
/*****************  START FUNCTION DEFINITIONS  **********************/
/*****************************************************************************/
#if defined _WIN32 || defined _WINDOWS
#include <windows.h>
#endif
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#pragma pack (1)
#include <essapi.h>
#include <essotl.h>
#pragma pack ()

ESS_HINST_T hInst;
ESS_HCTX_T hCtx;
ESS_SVRNAME_T srvrName = "";
ESS_USERNAME_T userName = "";
ESS_PASSWORD_T pswd = "";

/* Initialization and Login functions */
void ESS_Init();
// void ESS_Login(); /* Requires command line arguments */
void ESS_Logout();
void ESS_Term();
void ESS_AutoLogin(); /* Displays the login dialog box */
void ESS_LoginSetPassword(); /* Called if EssAutoLogin returns error */
void ESS_GetVersion();
void ESS_GetAPIVersion();

/* Application functions */
void ESS_SetActive();
// void ESS_GetActive();
void ESS_ListApplications();
void ESS_ListDatabases();
void ESS_GetDatabaseInfo();

void ESS_ListUsers(); /* These functions will be called repeatedly */
void ESS_CreateUser(); /* to create a user, list users, rename the */
void ESS_RenameUser(); /* new user, list users again, then delete */
void ESS_DeleteUser(); /* the new users and list users again */
void ESS_GetUserInfo();

 /**************************************************************************/
/********************  START FUNCTION DECLARATIONS  **********************/
/***************************************************************************/

void ESS_Init()
{
    ESS_STS_T             sts;
    ESS_INIT_T InitStruct = {ESS_API_VERSION,
                             NULL,
                             0L,
                             255,
                             NULL,
                             NULL,
                             NULL,
                             NULL,
                             NULL,
                             NULL,
                             NULL,
                             NULL,
                             NULL,
                             0L};
    if ((sts = EssInit(&InitStruct, &hInst)) != ESS_STS_NOERR)
    {
        printf("EssInit failure: %ld\n", sts);
        exit ((int) sts);
    }
    printf("EssInit sts: %ld\n", sts);
}

/***************************************************************************/
void ESS_Login()
{
    ESS_STS_T sts = ESS_STS_NOERR;
    ESS_USHORT_T Items;
    ESS_PAPPDB_T pAppsDbs = NULL;
    sts = EssLogin (hInst, srvrName, userName, pswd, &Items,
                    &pAppsDbs, &hCtx);
    printf("EssLogin sts: %ld\r\n", sts);
    if ( (sts == 1051093L) || (sts == 1051090L) )
    {
        ESS_LoginSetPassword();
    }
    else
    if ( (sts != 0) && (sts != 1051093L) && (sts != 1051090L) )
    {  
        printf("MAINAPI servername username password\n");
        printf("Default: servername: local\n");
        printf("user name: admin\n");
        exit ((int) sts);
    }
}

/******************************************************************************/

1674 API Sample Programs
void ESS_AutoLogin ()
{
    ESS_STS_T sts = ESS_STS_NOERR;
    ESS_CHAR_T SvrName[ESS_SVRNAMELEN];         //this is different in VC++6
    ESS_CHAR_T UserName[ESS_USERNAMELEN];
    ESS_CHAR_T Password[ESS_PASSWORDLEN];
    ESS_CHAR_T AppName[ESS_APPNAMELEN];
    ESS_CHAR_T DbName[ESS_DBNAMELEN];

    ESS_USHORT_T Option;
    ESS_ACCESS_T Access;
    //  ESS_HCTX_T hCtx; Don't set this again, it is set at the top

    /* Initialize parameters */
    strcpy(SvrName,"localhost");
    strcpy(UserName,"Admin");
    strcpy(Password,"Password");
    strcpy(AppName,"");
    strcpy(DbName,"");
    Option = AUTO_DEFAULT;

    /* Login to Essbase Server */
    sts = EssAutoLogin (hInst, SvrName, UserName, Password,
                         AppName, DbName, Option, &Access, &hCtx);
    printf("EssAutoLogin sts: %ld\r\n", sts);
}

/******************************************************************************/
void ESS_LoginSetPassword()
{
    ESS_STS_T sts = ESS_STS_NOERR;
    ESS_USHORT_T Items;
    ESS_PAPPDB_T pAppsDbs = NULL;

    ESS_PASSWORD_T newPswd = "password2";

    sts = EssLoginSetPassword (hInst, srvrName, userName, pswd, newPswd,
                                &Items, &pAppsDbs, &hCtx);
    printf("EssLoginSetPassword sts: %ld\r\n", sts);
    if (sts)
    {   printf("\n\tEssLoginSetPassword sts: %ld\n",sts);
        exit ((int) sts);
    }
}

/******************************************************************************/
void ESS_GetAPIVersion()
{
    ESS_STS_T    sts = ESS_STS_NOERR;
    ESS_ULONG_T  Version;

    sts = EssGetAPIVersion(&Version);
    if(!sts)
        printf("API Version %#x\n",Version);
}
void ESS_Term()
{
    ESS_STS_T sts = ESS_STS_NOERR;
    if (sts = EssTerm(hInst)) != ESS_STS_NOERR)
    {
        /* error terminating API */
        exit((ESS_USHORT_T) sts);
    }
    printf("EssTerm sts: %ld\n", sts);
}

void ESS_Logout()
{
    ESS_STS_T sts = ESS_STS_NOERR;
    sts = EssLogout (hCtx);
    printf("\n
EssLogout sts: %ld\n", sts);
}

void ESS_GetVersion()
{
    ESS_STS_T sts = ESS_STS_NOERR;
    ESS_USHORT_T Release;
    ESS_USHORT_T Version;
    ESS_USHORT_T Revision;
    sts = EssGetVersion (hCtx, &Release, &Version, &Revision);
    printf("EssGetVersion sts: %ld\n", sts);
    if(!sts)
    {
        printf("\r\nEssbase Application Server - ");
        printf("Version %d.%d.%d\n", Release, Version, Revision);
    }
}

void ESS_GetActive()
{
    ESS_STS_T sts = ESS_STS_NOERR;
    ESS_STR_T pDbName;
    ESS_STR_T pAppName;
    ESS_ACCESS_T Access;
    if((sts = EssAlloc (hInst, 80, (ESS_PPVOID_T)&pAppName)) == 0)
    {
        if((sts = EssAlloc (hInst, 80, (ESS_PPVOID_T)&pDbName)) == 0)
        {
            if((sts = EssGetActive(hCtx, &pAppName, &pDbName, &Access)) == 0)
            {
                if(pAppName)
                {
                    if(*pAppName)
printf("Current active app: [%s]\r\n", pAppName);

else
printf("No active Application is set\r\n");
}
EssFree(hInst, pDbName);
}
EssFree(hInst, pAppName);
}

/**************************************************************************************/
void ESS_SetActive()
{
    ESS_STS_T sts = ESS_STS_NOERR;
    ESS_ACCESS_T Access;
    ESS_STR_T AppName;
    ESS_STR_T DbName;

    AppName = "sample";
    DbName = "basic";
    sts = EssSetActive(hCtx, AppName, DbName, &Access);
    printf("EssSetActive sts: %ld\r\n", sts);
}

/**************************************************************************************/
void ESS_ListApplications()
{
    ESS_STS_T sts = ESS_STS_NOERR;
    ESS_PAPPNAME_T strp = NULL;
    ESS_USHORT_T Items;
    ESS_USHORT_T ind;

    sts = EssListApplications(hCtx, &Items, &strp);
    if(!sts)
    {
        if(Items && strp)
        {
            printf("Applications availables\r\n");
            for(ind = 0; ind < Items; ind++)
            {
                if(strp[ind] != NULL)
                    printf("%s\r\n", strp[ind]);
            }
            EssFree(hInst, strp);
        }
        else
        printf("Application List is Empty\r\n\r\n");
    }
}

/**************************************************************************************/
void ESS_ListDatabases()
{
    ESS_STS_T sts = ESS_STS_NOERR;
    ESS_USHORT_T Items;
    ESS_USHORT_T ind;
ESS_PAPPDB_T pAppsDbs = NULL;

sts = EssListDatabases(hCtx, NULL, &Items, &pAppsDbs);
printf("EssListDatabases sts: %ld\n",sts);

if(!sts)
{
    if(Items && pAppsDbs)
    {
        printf("\n--Applications/databases available--\n");
        for (ind = 0; ind<Items; ind++)
        {
            if((pAppsDbs+ind) !=NULL)
            {
                if((pAppsDbs[ind].AppName != NULL) && (pAppsDbs[ind].DbName != NULL))
                {
                    printf("%s",pAppsDbs[ind].AppName);
                    printf(" ==> ");
                    printf("%s",pAppsDbs[ind].DbName);
                    printf("\n\n");
                }
            }
        }
        EssFree(hInst, pAppsDbs);
    }
    else
    
        printf("\nDatabase List is Empty\n\n");
    }
}

void ESS_GetDatabaseInfo()
{
    ESS_STS_T sts = ESS_STS_NOERR;
    ESS_PDBINFO_T DbInfo;
    ESS_STR_T AppName;
    ESS_STR_T DbName;

    AppName = "Sample";
    DbName = "Basic";

    sts = EssGetDatabaseInfo(hCtx, AppName, DbName, &DbInfo);
    if(!sts)
    {
        printf("\n------ Results of EssGetDatabaseInfo ------\n");
        printf("AppName: %s\n",DbInfo->AppName);
        printf("DbName: %s\n",DbInfo->Name);
        printf("DbType: %d\n",DbInfo->DbType);
        printf("Status: %d\n",DbInfo->Status);
        printf("nConnects: %d\n",DbInfo->nConnects);
        printf("nLocks: %d\n",DbInfo->nLocks);
        printf("nDims: %d\n",DbInfo->Data);
        printf("Country: %s\n",DbInfo->Country);
        printf("Time: %s\n",DbInfo->Time);
        printf("Category: %s\n",DbInfo->Category);
        printf("Type: %s\n",DbInfo->Type);
printf("CrPartition: %s\r\n",DbInfo->CrPartition);
printf("\r\n------ Results of EssGetDatabaseInfo ------\r\n");

if(DbInfo)
{
   EssFree(hInst, DbInfo);
}
}

/*********************************************************************/
void ESS_ListUsers()
{

   ESS_STS_T      sts;
   ESS_USHORT_T   Count;
   ESS_PUSERINFO_T Users = NULL;
   ESS_USHORT_T   ind;

   sts = EssListUsers (hCtx, NULL, NULL, &Count, &Users);
   if (!sts)
   {
      if (Count && Users)
      {
         printf ("\r\n------User List from EssListUsers()------\r\n\r\n");
         for (ind = 0; ind < Count; ind++)
         {
            printf ("Name->%s	Application->%s	database->%s\r\n",
                     Users[ind].Name, Users[ind].AppName,
                     Users[ind].DbName);
         }
         printf ("\r\n------User List from EssListUsers()------\r\n\r\n");
         EssFree (hInst, Users);
         printf("\r\n");
      }
      else
      {
         printf ("\r\nUsers list is empty\r\n\r\n");
      }
   }

} /* ***************************************************************/
void ESS_CreateUser()
{

   ESS_STS_T     sts = ESS_STS_NOERR;

   ESS_CHAR_T UserName[] = "newuser";  //this is different in VC++6
   ESS_CHAR_T Password[] = "password"; //compare to API reference example

   printf("Begin EssCreateUser Function");
   sts = EssCreateUser (hCtx, UserName, Password);
   printf("EssCreateUser sts: %ld",sts);
}

/***************************************************************************/
void ESS_RenameUser()
{

}
ESS_STS_T     sts = ESS_STS_NOERR;
ESS_CHAR_T OldName[] = "newuser";
ESS_CHAR_T NewName[] = "user4";

sts = EssRenameUser (hCtx, OldName, NewName);
}

/***************************************************************************/
void ESS_DeleteUser()
{
    ESS_STS_T     sts = ESS_STS_NOERR;
    ESS_CHAR_T UserName[] = "user4";

    sts = EssDeleteUser (hCtx, UserName);
    printf("EssDeleteUser sts: %ld",sts);
}

/***************************************************************************/
void ESS_GetUserInfo ()
{
    ESS_STS_T        sts  = ESS_STS_NOERR;
    ESS_PUSERINFO_T  User = NULL;

    sts = EssGetUser (hCtx, "Jim Smith", &User);
    printf("EssGetUserInfo %ld\r\n",sts);

    if (!sts)
    {
        printf ("Name->%s Application->%s database->%s\r\n",  
            User->Name, User->AppName, User->DbName);
        printf("Login %d\r\n",User->Login);
        printf("Type %d\r\n",User->Type);
        printf("Access %d\r\n",User->Access);
        printf("MaxAccess %d\r\n",User->MaxAccess);
        printf("Expiration %d\r\n",User->Expiration);
        printf("LastLogin %d\r\n",User->LastLogin);
        printf("FailCount %d\r\n",User->FailCount);
        printf("LoginId %ld\r\n",User->LoginId);

        if (User)
            EssFree (hInst, User);
    }
}

/***************************************************************************/
void getCmdLineArgs(int argc, char *argv[])
{
    if (argc>1)
        strcpy(srvrName,argv[1]);

    if (argc>2)
        strcpy(userName,argv[2]);

    if (argc>3)
        strcpy(pswd,argv[3]);

    printf("Server name:  %s\n",srvrName);
Sample C API Program 3 (cs3.c)

This file contains an annotated Essbase C API program. This fundamental sample program can be used in a C++ programming environment as a starting point for more functional programs.

This file is to be used with the Oracle Essbase API Reference to illustrate basic points in API programming. A complete set of actual C code files is also included with the Essbase API. Look in the samples directory for the *.c files, executables, projects, and workspaces.

/*
   Copyright 1992-2008 Oracle Corporation. All Rights Reserved.
*/

NAME
 cs3.c

DEPENDENCIES
 You must add ESSAPIN.LIB to your project.
 You must also identify the API/Include and API/Lib directories to the compiler/linker.
DESCRIPTION
This file is used as an extended example of API programming techniques. This program illustrates the sequence of function call expected by the Essbase Server and shows the syntax of actual API function calls in an actual working program.

NOTES
This program has three sections:
1 - the includes and function definitions
2 - the function declarations
3 - the main program flow

MODIFIED
* Created 26July99 Publications
*/

/************************** Includes and Definitions **************************/
/************************** Includes and Definitions **************************/

#if defined _WIN32 || defined _WINDOWS
#include <windows.h>
#endif
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#pragma pack (1)
#include <essapi.h>
#include <essotl.h>
#pragma pack ()

ESS_HINST_T hInst;
ESS_HCTX_T hCtx;
ESS_SVRNAME_T srvrName = "";
ESS_USERNAME_T userName = "";
ESS_PASSWORD_T pswd = "";

/* Initialization and Login functions */
void ESS_Init();
void ESS_Login();
void ESS_Logout();
void ESS_Term();
void ESS_AutoLogin();
void ESS_GetVersion();
void ESS_GetAPIVersion();
void ESS_LoginSetPassword();
void ESS_SetActive();
void ESS_ListDatabases();
void ESS_UnloadDb();
void ESS_ClearDatabase();

/* Report - updating - Calculation */
void ESS_Report();
void ESS_RunRept ();
void ESS_ReportFile ();

void ESS_Update();
void ESS_UpdateFile();

void ESS_Calc();
void ESS_CalcLine();
void ESS_RunCalc ();
void ESS_CalcFile();
void ESS_Import ();

void ESS_Free();

/******************  START FUNCTION DECLARATIONS  ******************/
/*******************************************************************/
void ESS_Init()
{
    ESS_STS_T     sts;
    ESS_INIT_T   InitStruct = {ESS_API_VERSION,
                                NULL,
                                0L,
                                255,
                                NULL,
                                NULL,
                                NULL,
                                NULL,
                                NULL,
                                NULL,
                                NULL,
                                0L };

    if ((sts = EssInit(&InitStruct, &hInst)) != ESS_STS_NOERR)
    {       printf("EssInit failure: %ld\n", sts);
            exit ((int) sts);
    }
    printf("EssInit sts: %ld\n", sts);
}

/**********************************************************************/
void ESS_Login()
{
    ESS_STS_T     sts = ESS_STS_NOERR;
    ESS_USHORT_T   Items;
    ESS_PAPPDB_T  pAppsDbs = NULL;

    sts = EssLogin (hInst, srvrName, userName, pswd, &Items, &pAppsDbs,
                    &hCtx);
    printf("EssLogin sts: %ld\r\n", sts);
    if ( (sts == 1051093L) || (sts == 1051090L) )
    {   ESS_LoginSetPassword();
    } else
    if ( (sts != 0) && (sts != 1051093L) && (sts != 1051090L) )
    {   printf("\n\Usage:  MAINAPI servername username password\n");
        printf("\n\Default:  mainAPI servername local\n");
}
printf("user name: admin\n\tpassword: password\n");
exit((int) sts);
}

void ESS_AutoLogin ()
{
    ESS_STS_T sts = ESS_STS_NOERR;
    ESS_CHAR_T SvrName[ESS_SVRNAMELEN];      //this is different in VC++6
    ESS_CHAR_T UserName[ESS_USERNAMELEN];
    ESS_CHAR_T Password[ESS_PASSWORDLEN];
    ESS_CHAR_T AppName[ESS_APPNAMELEN];
    ESS_CHAR_T DbName[ESS_DBNAMELEN];

    ESS_UHINT_T Option;
    ESS_ACCESS_T Access;
    // ESS_HCTX_T hCtx;  Don't set this again, it is set at the top

    /* Initialize parameters */
    strcpy(SvrName,"localhost");
    strcpy(UserName,"Admin");
    strcpy(Password,"Password");
    strcpy(AppName,"" );
    strcpy(DbName,"" );
    Option = AUTO_DEFAULT;

    /* Login to Essbase Server */
    sts = EssAutoLogin (hInst, SvrName, UserName, Password,
    AppName, DbName, Option, &Access, &hCtx);
    printf("EssAutoLogin sts: %ld\r\n", sts);
}

void ESS_LoginSetPassword()
{
    ESS_STS_T sts = ESS_STS_NOERR;
    ESS_UHINT_T Items;
    ESS_PAPPDB_T pAppsDbs = NULL;
    ESS_PASSWORD_T newPswd = "password2";

    sts = EssLoginSetPassword (hInst, srvrName, userName, pswd, newPswd, &Items,
    &pAppsDbs, &hCtx);
    printf("EssLoginSetPassword sts: %ld\r\n", sts);
    if (sts)
    {   printf("\n\tEssLoginSetPassword sts: %ld\n",sts);
        exit ((int) sts);
    }
}

void ESS_GetAPIVersion()
{
    ESS_STS_T    sts = ESS_STS_NOERR;
    ESS_ULONG_T  Version;

    sts = EssGetAPIVersion(&Version);
}
if(!sts)
    printf("API Version %#x\n",Version);

/**********************************************************************/
void ESS_Term() {
    ESS_STS_T sts = ESS_STS_NOERR;
    if ((sts = EssTerm(hInst)) != ESS_STS_NOERR) {
        /* error terminating API */
        exit((ESS_USHORT_T) sts);
        printf("EssTerm sts: %ld\n", sts);
    }

/**********************************************************************/
void ESS_Logout() {
    ESS_STS_T sts = ESS_STS_NOERR;
    sts = EssLogout (hCtx);
    printf("\n\nEssLogout sts: %ld\n",sts);
}

/**********************************************************************/
void ESS_GetVersion() {
    ESS_STS_T sts = ESS_STS_NOERR;
    ESS_USHORT_T Release;
    ESS_USHORT_T Version;
    ESS_USHORT_T Revision;
    sts = EssGetVersion (hCtx, &Release, &Version, &Revision);
    printf("EssGetVersion sts: %ld\n", sts);
    if(!sts) {
        printf("\r\nEssbase Application Server - ");
        printf("Version %d.%d.%d\n", Release, Version, Revision);
    }
}

/**********************************************************************/
void ESS_GetActive() {
    ESS_STS_T sts = ESS_STS_NOERR;
    ESS_STR_T pDbName;
    ESS_STR_T pAppName;
    ESS_ACCESS_T Access;
    if((sts = EssAlloc (hInst, 80, (ESS_PPVOID_T)&pAppName)) == 0) {
        if((sts = EssAlloc (hInst, 80, (ESS_PPVOID_T)&pDbName)) == 0) {
            if((sts =
EssGetActive(hCtx, &pAppName, &pDbName, &Access)) == 0)
{
    if(pAppName)
    {
        if(*pAppName)
            printf("Current active app: [%s]\r\n",pAppName);
        else
            printf("No active Application is set\r\n");
    }
    EssFree(hInst, pDbName);
}
    EssFree(hInst, pAppName);
}

/**************************************************************************************/

void ESS_SetActive()
{
    ESS_STS_T sts = ESS_STS_NOERR;
    ESS_ACCESS_T Access;
    ESS_STR_T AppName;
    ESS_STR_T DbName;

    AppName = "sample";
    DbName = "basic";
    sts = EssSetActive(hCtx, AppName, DbName, &Access);
    printf("EssSetActive sts: %ld\r\n",sts);
}

/**************************************************************************************/

void ESS_ListDatabases()
{
    ESS_STS_T sts = ESS_STS_NOERR;
    ESS_USHORT_T Items;
    ESS_USHORT_T ind;
    ESS_PAPPDB_T pAppsDbs = NULL;

    sts = EssListDatabases(hCtx, NULL, &Items, &pAppsDbs);
    printf("EssListDatabases sts: %ld\r\n",sts);
    if(!sts)
    {
        if(Items && pAppsDbs)
        {
            printf("\r\n--Applications/databases available--\r\n");
            for (ind = 0; ind<Items; ind++)
            {
                if((pAppsDbs+ind) != NULL)
                {
                    if((pAppsDbs[ind].AppName != NULL)
                        && (pAppsDbs[ind].DbName != NULL))
                    {
                        printf("%s",pAppsDbs[ind].AppName);
                        printf(" ==> ");
                        printf("%s",pAppsDbs[ind].DbName);
                        printf("\n\r");
                    }
                }
            }
        }
    }
}
void ESS_ClearDatabase()
{
    ESS_STS_T sts = ESS_STS_NOERR;
    sts = EssClearDatabase(hCtx);
    printf("EssClearDatabase sts:%ld\r\n",sts);
    printf("The database is now empty\n");
}

void ESS_Report()
{
    ESS_STS_T sts = ESS_STS_NOERR;
    ESS_STR_T rString;
    ESS_CHAR_T pszReportIn[512];
    strcpy(pszReportIn,
        " {TABDELIMIT} \\
        {SUPALL COLHEADING NAMESON BLOCKHEADERS PAGEHEAD INDENTGEN 2 DECIMALS \ 
        VARIABLE} \\
        (BRACKET) \ 
        <SINGLECOLUMN \ 
        <QUOTEEMBRNAMES \ 
        (SUPMISSING) \ 
        <BOTTOM ( 4, @DATACOL(1) ) \ 
        <SYM \ 
        <PAGE( 'Measures') \ 
        'Measures' \ 
        <COL( 'Market','Scenario') \ 
        ( OUTALTNAMES } \ 
        <ICHOILDREN 'Market' \ 
        'Actual' \ 
        'Budget' \ 
        <ROW( 'Year','Product') \ 
        <ICHOILDREN 'Year' \ 
        <ICHOILDREN 'Product' \ 
        ! " );

    sts = EssReport (hCtx, ESS_TRUE, ESS_FALSE, pszReportIn);
    //sts = EssReport (hCtx, ESS_TRUE, ESS_FALSE, "<Desc &ThisMonth !");
    //sts = EssReport (hCtx, ESS_TRUE, ESS_FALSE, "<Desc Year !");
    printf("EssReport sts: %ld\r\n",sts);

    if(!sts)
        sts = EssGetString(hCtx, &rString);
    while (((!sts) && (rString != NULL))
    {
}
printf("\%s", rString);
EssFree (hInst, rString);
sts = EssGetString (hCtx, &rString);
}
printf("\r\n");

//----------------------------------------------------------------------------
void ESS_Update()
{
    ESS_STS_T sts = ESS_STS_NOERR;

    sts = EssUpdate(hCtx, ESS_TRUE, ESS_FALSE,
                    "Year Market Scenario Measures Product 123456");
    printf("EssUpdate sts: \ld\r\n",sts);
}

//----------------------------------------------------------------------------
void ESS_CalcLine()
{
    ESS_STS_T sts = ESS_STS_NOERR;
    ESS_STR_T Script;
    ESS_PROCSTATE_T pState;

    Script = "CALC DIM (Measures, Product, Market, Year, Scenario)";

    sts = EssCalc(hCtx, ESS_TRUE, Script);
    printf("EssCalc sts: \ld\r\n",sts);

    if (!sts)
    {
        sts = EssGetProcessState (hCtx, &pState);
        while (!sts || (pState.State != ESS_STATE_DONE))
            sts = EssGetProcessState (hCtx, &pState);
    }
}

//----------------------------------------------------------------------------
void ESS_Calc()
{
    ESS_STS_T sts = ESS_STS_NOERR;
    ESS_STR_T Script;
    ESS_PROCSTATE_T pState;
    Script = "CALC ALL;";

    sts = EssBeginCalc (hCtx, ESS_TRUE);
    printf("EssBeginCalc sts: \ld\r\n",sts);

    if (!sts)
    {
        sts = EssSendString (hCtx, Script);
        printf("EssSendString sts: \ld\r\n",sts);
    }

    if (!sts)
    {
        sts = EssEndCalc (hCtx);
        printf("EssEndCalc sts: \ld\r\n",sts);
    }
}
if (!sts)
{
    sts = EssGetProcessState (hCtx, &pState);
    while(!sts && (pState.State != ESS_STATE_DONE))
        sts = EssGetProcessState (hCtx, &pState);
}

/******************************************************************************/
void ESS_ReportFile ()
{
    ESS_STS_T   sts = ESS_STS_NOERR;
    ESS_HCTX_T   hSrcCtx;
    ESS_STR_T    rString;
    ESS_STR_T    AppName;
    ESS_STR_T    DbName;
    ESS_STR_T    FileName;

    hSrcCtx = hCtx;
    AppName = "Sample";
    DbName  = "Basic";
    FileName = "cdlockdb";

    sts = EssReportFile (hCtx, hSrcCtx, AppName, DbName, FileName,
                        ESS_TRUE, ESS_FALSE);
    printf("EssReportFile sts: %ld\r\n",sts);
    if (!sts)
        sts = EssGetString (hCtx, &rString);
    while ((!sts) && (rString != NULL))
    {
        printf ("%s", rString);
        EssFree (hInst, rString);
        sts = EssGetString (hCtx,&rString);
    }
}

/******************************************************************************/
void ESS_UpdateFile ()
{
    ESS_STS_T   sts = ESS_STS_NOERR;
    ESS_HCTX_T   hSrcCtx;
    ESS_BOOL_T   isStore;
    ESS_BOOL_T   isUnlock;
    ESS_STR_T    AppName;
    ESS_STR_T    DbName;
    ESS_STR_T    FileName;

    AppName  = "Sample";
    DbName   = "Basic";
    hSrcCtx  = hCtx;
    FileName = "cdupdtodb.txt";
    isStore  = ESS_TRUE;
    isUnlock = ESS_FALSE;

    sts = EssUpdateFile (hCtx, hSrcCtx, AppName, DbName, FileName,
                        isStore, isUnlock);
printf("EssUpdateFile sts: %ld\r\n",sts);
}

/************************************************************/
void ESS_RunCalc ()
{
    ESS_STS_T     sts = ESS_STS_NOERR;
    ESS_HCTX_T    hSrcCtx;
    ESS_BOOL_T    isObject = ESS_FALSE;
    ESS_STR_T     AppName;
    ESS_STR_T     DbName;
    ESS_STR_T     FileName;
    ESS_PROCSTATE_T pState;

    hSrcCtx  = hCtx;
    AppName  = "Sample";
    DbName   = "Basic";
    FileName = "calc5dim";

    sts = EssCalcFile (hCtx, hSrcCtx, AppName, DbName, FileName,
                        ESS_TRUE);
    printf("EssCalcFile sts: %ld\r\n",sts);

    if (!sts)
    {
        sts = EssGetProcessState (hCtx, &pState);
        while (!sts || (pState.State != ESS_STATE_DONE))
            sts = EssGetProcessState (hCtx, &pState);
    }
}

/************************************************************/
void ESS_Import ()
{
    eSS_STS_T       sts = eSS_STS_NOERR;
    eSS_SHORT_T     isAbortOnError;
    eSS_OBJDEF_T    Rules;
    eSS_OBJDEF_T    Data;
    eSS_PMBRERR_T   pMbrErr = NULL;

    Data.hCtx      = hCtx;
    Data.AppName   = "Sample";
    Data.DbName    = "Basic";
    Data.ObjType   = ESS_OBJTYPE_TEXT;
    Data.FileName  = "calcdat.txt";

    Rules.hCtx     = hCtx;
    Rules.AppName  = "Olap";
    Rules.DbName   = "Demo";
    Rules.ObjType  = eSS_OBJTYPE_RULES;
    Rules.FileName = "Actmap";

    // Running conditions *
    isAbortOnError = eSS_TRUE;

    sts = EssImport (hCtx, NULL, &Data, &pMbrErr, NULL, isAbortOnError);
printf("EssImport sts: %ld\n",sts);

if(pMbrErr)
    EssFreeMbrErr(hCtx, pMbrErr);
}

//**********************************************************************/

/*
 This routine gets arguments from the command line. The routine under
 stands a number of arguments will be present up to 3 arguments total
 The first parameter, argc, is the number of arguments present
 following the command to start (csamp3). The second parameter, argv,
 is the array of arguments. This program (csamp3) has been built to
 override the command line arguments, but could be easily modified to
 use them. In other words, this routine is not used.
 */

void getCmdLineArgs(int argc, char *argv[])
{
    if (argc>1)
        strcpy(srvrName,argv[1]);
    if (argc>2)
        strcpy(userName,argv[2]);
    if (argc>3)
        strcpy(pswd,argv[3]);

        printf("Server name:  %s\n",srvrName);
        printf("User name:  %s\n",userName);
        printf("Password:  %s\n",pswd);
}

//*************************************************************************/

/* *******************  Program Main Flow  ****************************/ */

void main(int argc, char *argv[])
{
    getCmdLineArgs(argc,argv);

    //**** Initialization and Login Functions ****/

    ESS_Init();
    ESS_AutoLogin();
    ESS_GetVersion();
    ESS_GetAPIVersion();
    ESS_SetActive();
    ESS_ListDatabases();

    //**** Report and Updating Calculation ****/

    /*
    This section issues a report to show what is in the database, then
    clears all the data, runs another report to show that the database
    is empty, then imports data from calcdat.txt, then finally, issues
    another report to show that the database now has data.
    */

    ESS_Report();
ESS_ClearDatabase();
ESS_Report();
ESS_Import();
ESS_Report();

/*
This section runs a calculation from a file. (ESS_RunCalc calls
EssCalcFile, which specifies the calculation script in the file
calc5dim.csc.) Then issues yet another report to show the results.
*/

ESS_CalcLine();
ESS_Report();

ESS_ReportFile();
ESS_UpdateFile();
ESS_ReportFile();

ESS_Logout();
ESS_Term();
}

/*
End of program
*/

Sample Visual Basic API Program 1 (initialize.vbp)

This file contains an annotated Essbase Visual Basic API program. This fundamental sample
program can be used in a Visual Basic programming environment as a starting point for more
functional programs.

This file is to be used with the Oracle Essbase API Reference to illustrate basic points in API
programming. A complete set of actual VB code files is also included in the samples directory
of this documentation.

Option Explicit

'*** Always obtain and process the return error status
Dim lngStatus As Long   ' Return error status

'*** EsbGetAPIVersion() needs the following
Dim lngAPIVersion As Long

'*** EsbInit() accepts an initialization structure
'*** and returns an instance handle
Dim structInit As ESB_INIT_T   ' Create an instance of the initialization structure
Dim lngInstHndl As Long        ' Instance handle for program (returned by EsbInit())

'*** EsbGetMessage() (enabled by cmdInit()) needs the following
'*** (see EsbListMessages() for intMsgLen and strMsg)
Dim intMsgLev As Integer   ' Whether information/warning/serious error/fatal error
Dim lngMsgNmbr As Long     ' Message number in Essbase.mdb

'*** EsbAutoLogin() needs the following
Dim strServer As String * ESB_SVRNAMELEN   ' Empty string okay
Dim strUser As String * ESB_USERNAMELEN ' Empty string okay
Dim strPassword As String * ESB_PASSWORDLEN ' Empty string okay
Dim strAppName As String * ESB_APPNAMELEN ' Empty string okay
Dim strDbName As String * ESB_DBNAMELEN ' Empty string okay
Dim intOption As Integer ' Flags whether to display dialog box, allow user to log in without selecting the application/database, or allow user to interact with dialog box to log in and select the application/database
Dim intAccess As Integer ' User's access level to application/database
Dim lngCtxHndl As Long ' Context handle for login (returned by EsbAutoLogin()

'***
' Initialized, logged in, able to log out, able to terminate
' Ready to work with databases, users, objects
'***

'*** MORE DECLARATIONS HERE OR IN SUB PROCEDURES
Dim intArrayIndex As Integer ' Declare an integer, for example

Private Sub ESB_ListErrorStackMsgs()

'*** EsbGetMessage() needs the following
'*** (see Declarations for intMsgLev and lngMsgNmbr)
Const intMsgLen = 256 ' Set maximum message length as a constant,
Dim strMsg As String * intMsgLen ' then Dim message string at that length

'*** Get all messages from error stack and display them in list box
lngStatus = EsbGetMessage(lngInstHndl, intMsgLev, lngMsgNmbr, strMsg, intMsgLen)
 ' Retrieves strMsg from stack and decrements stack pointer

Dim intStackNmbr As Integer ' To track the number of messages on the error stack
intStackNmbr = 1

Do While Mid$(strMsg, 1, 1) <> Chr$(0) ' Do while the error stack has messages
MsgBox "Error stack #" & (intStackNmbr) & " is level "/message "/" & (intMsgLev) & "/" & (lngMsgNmbr)
intStackNmbr = intStackNmbr + 1 ' Increment the stack number displayed
lngStatus = EsbGetMessage(lngInstHndl, intMsgLev, lngMsgNmbr, strMsg, intMsgLen)
Loop

End Sub

Private Sub cmdAutoLogin_Click()

intOption = ESB_AUTO_DEFAULT ' Allows user to interact with login dialog box

'*** Call EsbAutoLogin() and obtain the return error status
lngStatus = EsbAutoLogin(lngInstHndl, _
 strServer, strUser, strPassword, _
 strAppName, strDbName, _
 intOption, _
 intAccess, _
 lngCtxHndl) ' EsbAutoLogin() returns a unique context handle for each login, even if the user and server are the same

'*** Display the return error status

If lngStatus = 0 Then
    MsgBox "This login ID (context handle) is logged in: " & lngCtxHndl
    Call ESB_ListErrorStackMsgs ' Even successful logins return useful messages
    cmdAutoLogin.Enabled = False ' True would allow other login IDs (context handles)
    cmdLogout.Enabled = True ' Log out;
    cmdTerm.Enabled = False ' then terminate the API
Else
    MsgBox "Login failed: " & lngStatus
    Call ESB_ListErrorStackMsgs ' Always handle messages if function call fails
End If
End Sub

Private Sub cmdGetAPIVers_Click()

    '***
    ' You can call EsbGetAPIVersion() before or after you call EsbInit()
    '***

    '*** Call EsbGetAPIVersion() and obtain the return error status
    lngStatus = EsbGetAPIVersion(lngAPIVersion)

    '*** Display the API version or that the call failed
    If lngStatus = 0 Then
        MsgBox "The API version is " & lngAPIVersion
    Else
        MsgBox "EsbGetAPIVersion() failed: " & lngStatus
    End If
End Sub

Private Sub cmdInit_Click()

    '*** Initialize the structure before you call EsbInit()
    structInit.Version = ESB_API_VERSION
    structInit.MaxHandles = 10
    structInit.LocalPath = "C:\\Hyperion\\products\\Essbase\\EssbaseClient" ' <ARBORPATH>
    structInit.MessageFile = "" ' The default message file
    structInit.ClientError = ESB_TRUE ' Enables EsbGetMessage() to retrieve
                        '   top message in stack
    structInit.ErrorStack = 100 ' No. of messages allowed in stack;
                        '   stack initialized on each call

    '*** Call EsbInit() to initialize the API; obtain the return error status
    lngStatus = EsbInit(structInit, lngInstHndl)

    '*** Display the return error status
    If lngStatus = 0 Then
        MsgBox "The API is initialized: " & lngInstHndl
        cmdAutoLogin.Enabled = True ' You can log in only after you initialize the API
        cmdInit.Enabled = False ' Initialization endures until you terminate the API
        cmdTerm.Enabled = True
    Else
        MsgBox "The API failed to initialize: " & lngStatus
    End If
End Sub
Private Sub cmdLogout_Click()

'*** Call EsbLogout() and obtain return error status
lngStatus = EsbLogout(lngCtxHndl) ' Logs user out for the specified login context

'*** Display whether the logout succeeded or failed
If lngStatus = 0 Then  ' Should test that all login IDs (contexts) are logged out
    MsgBox "This login ID (context handle) is logged out: " & (lngCtxHndl)
    cmdLogout.Enabled = False  ' Log out;
    cmdTerm.Enabled = True     '   then terminate the API
Else
    MsgBox "EsbLogout() failed: " & (lngStatus)
End If
End Sub

Private Sub cmdTerm_Click()

'*** Call EsbTerm() after all other calls are completed
EsbTerm (lngInstHndl)

'*** Display whether the API terminated
If lngStatus = 0 Then
    MsgBox "The API is terminated"
    cmdGetAPIVers.Enabled = True  ' After you terminate the API,
    cmdInit.Enabled = True        '    you can call only EsbInit() and EsbGetVersion()
    cmdTerm.Enabled = False
    cmdAutoLogin.Enabled = False
Else
    MsgBox "EsbTerm() failed: " & (lngStatus)
End If
End Sub

Private Sub Form_Load()

' *** Must set boolean values in the form
ESB_TRUE = 1   ' ESB_TRUE
ESB_FALSE = 0  ' and ESB_FALSE are variables, not constants
End Sub

---

**Sample Visual Basic API Program 2 (appdb.vbp)**

This file contains an annotated Essbase Visual Basic API program. This fundamental sample program can be used in a Visual Basic programming environment as a starting point for more functional programs.

This file is to be used with the *Oracle Essbase API Reference* to illustrate basic points in API programming. A complete set of actual VB code files is also included in the *samples* directory of this documentation.

**Code in the Form**

This code is attached to the form itself. It calls functions in `Code.bas` (which follows). This arrangement allows you to include `Code.bas` in other projects.
Private Sub Form_Load()
    Call SetBeforeStart
End Sub

Private Sub SetBeforeStart()
    cmdStart.Enabled = True
    cmdStop.Enabled = False
    cmdClearMsg.Enabled = False
    lstMessages.Enabled = False
    cmdListApps.Enabled = False
    cmdListDbs.Enabled = False
    cmdSetActive.Enabled = False
    cmdGetActive.Enabled = False
    cmdGetDbInfo.Enabled = False
End Sub

Private Sub SetAfterLogin()
    cmdStart.Enabled = False
    cmdStop.Enabled = True
    cmdClearMsg.Enabled = True
    lstMessages.Enabled = True
    cmdListApps.Enabled = True
    cmdListDbs.Enabled = True
    cmdSetActive.Enabled = True
    cmdGetActive.Enabled = True
    cmdGetDbInfo.Enabled = True
End Sub

Private Sub cmdClearMsg_Click()
    lstMessages.Clear
End Sub

**Code in the Code.bas Module**

This code is in code.bas.

Option Explicit

'*******************
'RETURN ERROR STATUS
'*******************
    Dim lngStatus As Long

'***********
'INIT GLOBAL
'***********
    Dim structInit  As ESB_INIT_T
    Dim lngInstHndl As Long

'*******************
'ESB_GetMESSAGE GLOBAL
'*******************
Dim intMsgLev As Integer
Dim lngMsgNmbr As Long

'********************
'ESB_LOGIN GLOBAL
'********************

Dim lngCtxHndl As Long

'******************************************************
'ESB_SetACTIVE and ESB_ClearDATABASE GLOBAL
'******************************************************
Dim strActiveApp As String
Dim strActiveDb As String

'******************************************************
'Init and turn error handle turned off
'******************************************************
Sub ESB_Init()

ESB_TRUE = 1         ' ESB_TRUE
ESB_FALSE = 0        ' and ESB_FALSE are variables, not constants

'********************
' Define init structure
'********************
structInit.Version = ESB_API_VERSION
structInit.MaxHandles = 10
structInit.LocalPath = "C:\Hyperion\products\Essbase\EssbaseClient"
structInit.MessageFile = ""
structInit.ClientError = ESB_TRUE
structInit.ErrorStack = 100

'********************
'Initialize the API
'********************
lngStatus = EsbInit(structInit, lngInstHndl)
If lngStatus = 0 Then
    MsgBox "The API is initialized: " & (lngInstHndl)
Else
    MsgBox "The API failed to initialize: " & (lngStatus)
End If

End Sub

'******************************************************
'Login in user Admin. All login parameters are hardcoded
'******************************************************
Sub ESB_Login()

Dim strServer As String * ESB_SVRNAMELEN
Dim strUser As String * ESB_USERNAMELEN
Dim strPassword As String * ESB_PASSWORDLEN
Dim intNumAppDb As Integer
strServer = "Localhost"
strUser = "Admin"
strPassword = "password"

lngStatus = EsbLogin(lngInstHndl, _
    strServer, strUser, strPassword, _
    intNumAppDb, _
    lngCtxHndl)

'**************
'Error Checking
'**************
If lngStatus = 0 Then
    MsgBox "Admin is logged in, with login ID (context handle) " & (lngCtxHndl)
    Call ESB_ListErrorStackMsgs ' Even successful logins return useful messages
Else
    MsgBox "Login failed: " & (lngStatus)
End If
End Sub

'*******
' Logout
'*******
Sub ESB_Logout()
    lngStatus = EsbLogout(lngCtxHndl)

'*****************************************************************************
'Display whether the logout succeeded or failed
'*****************************************************************************
If lngStatus = 0 Then
    MsgBox "Admin, with login ID (context handle) " & (lngCtxHndl) _
        & ", is logged out"
Else
    MsgBox "EsbLogout() failed: " & (lngStatus)
End If
End Sub

'*****************************************************************************
'Terminate the VB API
'*****************************************************************************
Sub ESB_Term()

    EsbTerm (lngInstHndl)

'*****************************************************************************
'Display whether the API terminated
'*****************************************************************************
If lngStatus = 0 Then
    MsgBox "The API is terminated"
Else
    MsgBox "EsbTerm() failed: " & (lngStatus)
End If
End Sub

This is an error checking subroutine that uses EsbGetMessage

Sub ESB_ListErrorStackMsgs()

    Const intMsgLen = 256
    Dim strMsg As String * intMsgLen
    lngStatus = EsbGetMessage(lngInstHndl, intMsgLev, lngMsgNmbr, _
                               strMsg, intMsgLen)
    Dim intStackNmbr As Integer
    intStackNmbr = 1
    Do While Mid$(strMsg, 1, 1) <> Chr$(0)
        lstMessages "MESSAGE ON ERROR STACK:"
        lstMessages "Stack #" & (intStackNmbr) & 
        lstMessages "Level #" & (intMsgLev) & 
        lstMessages "Message #" & (lngMsgNmbr) & 
        lstMessages (strMsg) & 
        intStackNmbr = intStackNmbr + 1
        lngStatus = EsbGetMessage(lngInstHndl, intMsgLev, lngMsgNmbr, strMsg, intMsgLen)
    Loop

End Sub

This subroutine gets the names of the caller's current active application and database

Sub ESB_GetActive()

    Const intAppNameSize = ESB_APPNAMELEN
    Const intDbNameSize = ESB_DBNAMELEN
    Dim strAppName As String * intAppNameSize
    Dim strDbName As String * intDbNameSize
    Dim intUserAccess As Integer
    lngStatus = EsbGetActive(lngCtxHndl, strAppName, intAppNameSize, _
                               strDbName, intDbNameSize, intUserAccess)

End Sub

'Gets the names of the caller's current active application and database
'**********************************************************************
'**********************************************************************
Else
    lstMessages (strAppName)
    lstMessages "/ " & (strDbName)
End If
Else
    MsgBox "EsbGetActive() failed: " & (lngStatus)
End If

End Sub

**********************************************************************
'Gets a database's information structure, which contains non
'user-configurable parameters for the database. Sample Basic Hardcoded.
**********************************************************************
Sub Esb_GetDbInfo()

    Dim strAppName   As String
    Dim strDbName    As String
    Dim structDbInfo As ESB_DBINFO_T
    Dim structDbReqInfo As ESB_DBREQINFO_T
    Dim intI As Integer

    'Number of database info structures;
    'Applies where database is an empty string
    Dim intNumDbInfo As Integer

    strAppName = "Sample"
    strDbName = "Basic"

    lngStatus = EsbGetDatabaseInfo(lngCtxHndl, strAppName, strDbName, _
                                    structDbInfo, intNumDbInfo)

    '****************************************
    'Error Checking and Message display
    ****************************************
    If lngStatus = 0 Then
        MsgBox "You have retrieved a list of database info structures" & Chr(10) _
                & "EsbGetNextItem() will now generate a list"
    Else
        MsgBox "EsbGetDatabaseInfo() failed: " & (lngStatus)
        MsgBox "Note: Sample / Basic are Hardcoded for this Example"
    End If

    '****************************************
    'Get database information and display in list box
    '****************************************
    For intI = 1 To intNumDbInfo
        lngStatus = EsbGetNextItem(lngCtxHndl, ESB_DBREQINFO_TYPE, structDbReqInfo)
        If lngStatus = 0 Then
            MsgBox "EsbGetNextItem() succeeded"
            'Return values for the structDbReqInfo.DbReqType:
            ' 0 = Data load
            ' 1 = Calculation
            ' 2 = Outline update
            lstMessages "Type of request is: " & (structDbReqInfo.DbReqType)
            lstMessages "User is: " & (structDbReqInfo.User)
            ' User does not display - none is loading, calculating, or updating outline
    Next intI

End Sub
' BUT, cannot display structDbInfo fields, which is reason for call
Else
    MsgBox "EsbGetNextItem() failed: " & (lngStatus)
End If
Next
End Sub

'*********************************************************
'Lists all applications which are accessible to the caller
'*********************************************************
Sub Esb_ListApps()

    Dim intNumApps As Integer
    Dim strAppName As String * ESB_APPNAMELEN
    Dim intI As Integer  ' Index for loop

    lngStatus = EsbListApplications(lngCtxHndl, intNumApps)

    '**********************************
    'Error Checking and Message display
    '**********************************
    If lngStatus = 0 Then
        MsgBox "You have retrieved the application names" & Chr(10) _
            & "EsbGetNextItem() will now generate a list"
    Else
        MsgBox "EsbListApplications() failed: " & (lngStatus)
    End If

    '************************************************
    'Get list of applications and display in list box
    '************************************************
    For intI = 1 To intNumApps

        lngStatus = EsbGetNextItem(lngCtxHndl, ESB_APPNAME_TYPE, ByVal strAppName)

        If lngStatus = 0 Then
            MsgBox "EsbGetNextItem() succeeded"
            lstMessages (strAppName)
        Else
            MsgBox "EsbGetNextItem() failed: " & (lngStatus)
        End If

    Next

End Sub

'*********************************************************
'Lists all databases which are accessible to the caller,
'either within a specific application, or on an entire server.
'*********************************************************
Sub Esb_ListDbs()

    Dim strAppName  As String

Dim intNumDbs   As Integer
Dim structAppDb As ESB_APPDB_T
Dim intI As Integer  ' Index for loop

lngStatus = EsbListDatabases(lngCtxHndl, strAppName, intNumDbs)

'*************************************************
'display error checking and message display
'*************************************************
If lngStatus = 0 Then
    MsgBox "You have retrieved a list of application/database structures" & Chr(10) _
    & "EsbGetNextItem() will now generate a list"
Else
    MsgBox "EsbListDatabases() failed: " & (lngStatus)
End If

'**********************************************************
'Get list of applications/databases and display in list box
'**********************************************************
For intI = 1 To intNumDbs
    lngStatus = EsbGetNextItem(lngCtxHndl, ESB_APPDB_TYPE, structAppDb)
    If lngStatus = 0 Then
        MsgBox "EsbGetNextItem() succeeded"
        lstMessages (structAppDb.AppName)
        lstMessages "/ " & (structAppDb.DbName)
    Else
        MsgBox "EsbGetNextItem() failed: " & (lngStatus)
    End If
Next

End Sub

'*************************************************
'Sets the caller's active application and database
'*************************************************
Sub Esb_SetActive()

Dim strAppAnswer  As String
Dim strDbAnswer   As String
Dim intUserAccess As Integer

'*************************************************
'Input boxes allow users to select an app/db
'*************************************************
strAppAnswer = InputBox("Type the Application Name to Set Active. (May be case sensitive)")
strDbAnswer = InputBox("Type the Database Name to Set Active. (May be case sensitive)")

lngStatus = EsbSetActive(lngCtxHndl, strAppAnswer, strDbAnswer, intUserAccess)

'**************************************************
'Error Checking and Message display
'******************************************
If lngStatus = 0 Then
    MsgBox strAppAnswer & "/" & strDbAnswer & " is now active"
Else
    MsgBox "EsbSetActive() failed: " & (lngStatus)
End If

End Sub

Sub lstMessages(strItem As String)
    frmAppDb.lstMessages.AddItem (strItem)
End Sub

Sub lstMessagesClear()
    frmAppDb.lstMessages.Clear
End Sub

Sample Visual Basic API Program 3 (reports.vbp)

This file contains an annotated Essbase Visual Basic API program. This fundamental sample program can be used in a Visual Basic programming environment as a starting point for more functional programs.

This file is to be used with the Oracle Essbase API Reference to illustrate basic points in API programming. A complete set of actual VB code files is also included in the samples directory of this documentation.

Note: This sample program uses update, report, and calc scripts. By default, Essbase Server assumes such scripts are located in the application/database directory of the connected database. In the case of this sample program that directory is $ARBORPATH/App/Sample/Basic. The next place the server will look for the script files is the directory in which the program is running. The standard Oracle Essbase installation will have the calcdat.txt data load file, but the other script files need to be copied from $ARBORPATH/Docs/Api/Samples/vbexecs/V3Report to $ARBORPATH/App/Sample/Basic. You could put the files in any other location, but then you would have to specify the fully qualified paths to them in your program.

Code in the Form

This code is attached to the form itself. It calls functions in Code.bas (which follows). This arrangement allows you to include Code.bas in other projects.

Sub cmdStart_Click()
    Call Code.ESB_Init   ' Initializes ESB_INIT_T and calls EsbInit()
    Call ESB_Login       ' EsbLogin() sets server, user and password
    Call SetAfterLogin
End Sub

Sub cmdStop_Click()
    Call ESB_Logout      ' Should logout all login IDs (context handles)
Call ESB_Term          ' EsbTerm() terminates the API
Call lstMessagesClear
Call SetBeforeStart
End Sub

Sub cmdClearMsg_Click()
    lstMessages.Clear    'Clear Messages
End Sub

Sub cmdCalcFile_Click()
    Call ESB_CalcFile   'Calculate
End Sub

Sub cmdClrData_Click()
    Call ESB_SetActive  'Set the active database before calling EsbClearDatabase()
    Call ESB_ClrData    'Clear data
End Sub

Sub cmdLdData_Click()
    MsgBox "WAIT!! Don't do anything until this process completes. Click OK and wait about 15 seconds. "
    Call ESB_LdData     'Import Data
End Sub

Sub cmdQryFile_Click()
    Call ESB_QryFile
End Sub

Sub cmdQryStr_Click()
    Call ESB_QryStr
End Sub

Sub cmdQryStrs_Click()
    '    Call QryStrs
    Call ESB_BeginReport  ' 1. EsbBeginReport()
    Call ESB_SendString   ' 2. EsbSendString() - for each string in the report spec
    Call ESB_EndReport    ' 3. EsbEndReport()

    '*** Display returned data strings; assumes EsbBeginReport()'s ouput flag is TRUE
    If lngStatus = 0 Then    ' If EsbEndReport() succeeded, call EsbGetString()
        Call ESB_GetString  ' Server outputs data if intWhetherOutput = ESB_TRUE;
                          '    ESB_GetString calls EsbGetString() to read the returned
                          '    data until an empty string is returned
    End If
End Sub

Sub cmdUpdFile_Click()
    Call ESB_UpdFile
End Sub

Sub Form_Load()
    Call SetBeforeStart
End Sub

Sub SetBeforeStart()
'*** Enable cmdStart
    cmdStart.Enabled = True

'*** Disable everything else
    cmdStop.Enabled = False
    cmdClearMsg.Enabled = False
    lstMessages.Enabled = False
    cmdCalcFile.Enabled = False
    cmdClrData.Enabled = False
    cmdLdData.Enabled = False
    cmdQryStr.Enabled = False
    cmdQryStrs.Enabled = False
    cmdQryFile.Enabled = False
    cmdUpdFile.Enabled = False

End Sub

Sub SetAfterLogin()

'*** Disable cmdStart
    cmdStart.Enabled = False

'*** Enable everything else
    cmdStop.Enabled = True
    cmdClearMsg.Enabled = True
    lstMessages.Enabled = True
    cmdCalcFile.Enabled = True
    cmdClrData.Enabled = True
    cmdLdData.Enabled = True
    cmdQryStr.Enabled = True
    cmdQryStrs.Enabled = True
    cmdQryFile.Enabled = True
    cmdUpdFile.Enabled = True

End Sub

**Code in the Code.bas Module**

This code is in code.bas.

Option Explicit

'***********************
'RETURN ERROR STATUS
'***********************
    Dim lngStatus As Long

***************
'INIT GLOBAL
'***************
    Dim structInit As ESB_INIT_T
    Dim lngInstHndl As Long
    Dim lngCtxHndl As Long

'***********************
'ESB_GetMESSAGE GLOBAL

Dim intMsgLev  As Integer
Dim lngMsgNmbr As Long

'******************************************
'ESB_SetACTIVE and ESB_ClearDATABASE GLOBAL

Dim strActiveApp  As String
Dim strActiveDb   As String

'*********************************************

'Init and turn error handle turned off
'*********************************************

Sub ESB_Init()

ESB_TRUE = 1         ' ESB_TRUE
ESB_FALSE = 0        ' and ESB_FALSE are variables, not constants

' Define init structure

structInit.Version = ESB_API_VERSION
structInit.MaxHandles = 10
structInit.LocalPath = "$ARBORPATH"
structInit.MessageFile = ""
structInit.ClientError = ESB_TRUE
structInit.ErrorStack = 100

'Initialize the API

lngStatus = EsbInit(structInit, lngInstHndl)

'Error Checking

If lngStatus = 0 Then
    MsgBox "The API is initialized: " & (lngInstHndl)
Else
    MsgBox "The API failed to initialize: " & (lngStatus)
End If

End Sub

'*******************************************************

'Login in user Admin. All login parameters are hardcoded
'*******************************************************

Sub ESB_Login()

Dim strServer   As String * ESB_SVRNAMELEN
Dim strUser     As String * ESB_USERNAMELEN
Dim strPassword As String * ESB_PASSWORDLEN
Dim intNumAppDb As Integer

1706 API Sample Programs
strServer = "Localhost"
strUser = "Admin"
strPassword = "password"

lngStatus = EsbLogin(lngInstHndl, _
    strServer, strUser, strPassword, _
    intNumAppDb, _
    lngCtxHndl)

'**************
'Error Checking
'**************
If lngStatus = 0 Then
    MsgBox "Admin is logged in, with login ID (context handle) " & (lngCtxHndl) _
    & Chr$(10) & "WAIT! DO NOTHING!" _
    & Chr$(10) & "Retrieving login status; setting Sample/Basic as active"
    '*************************************************
    'Call the SetActive routine to select Sample Basic
    '*************************************************
    Call ESB_ListErrorStackMsgs ' Even successful logins return useful messages
    Call ESB_SetActive
Else
    MsgBox "Login failed: " & (lngStatus)
End If

End Sub

'**************************************************
'Sets the caller's active application and database.
'**************************************************
Sub ESB_SetActive()
Dim intUserAccess As Integer

strActiveApp = "Sample"
strActiveDb = "Basic"

lngStatus = EsbSetActive(lngCtxHndl, strActiveApp, strActiveDb, intUserAccess)

'**************
'Error Checking
'**************
If lngStatus = 0 Then
    MsgBox (strActiveApp) & "/" & (strActiveDb) & " is now active"
Else
    MsgBox "EsbSetActive() failed: " & (lngStatus)
End If

End Sub
'******
' Logout
'******
Sub ESB_Logout()

lngStatus = EsbLogout(lngCtxHndl)

'Display whether the logout succeeded or failed
'*************************************************************************
If lngStatus = 0 Then
    MsgBox "Admin, with login ID (context handle) " & (lngCtxHndl) 
    & ", is logged out"
Else
    MsgBox "EsbLogout() failed: " & (lngStatus)
End If

End Sub

'*************************
' Terminate the VB API
'****************************
Sub ESB_Term()

EsbTerm(lngInstHndl)

'Display whether the API terminated
'*************************************************************************
If lngStatus = 0 Then
    MsgBox "The API is terminated"
Else
    MsgBox "EsbTerm() failed: " & (lngStatus)
End If

End Sub

'***********************************************************************
'Gets a string of data from the active database.
'***********************************************************************
Sub ESB_GetString()

Const intDStringLen = 256

Dim strDataString As String * intDStringLen
Dim intNumGSCalls As Integer

intNumGSCalls = 1

lngStatus = EsbGetString(lngCtxHndl, strDataString, intDStringLen)

'*************************************************************************
'Call EsbGetString() until an empty string (no data) is returned
'*************************************************************************
Do While Mid$(strDataString, 1, 1) <> Chr$(0)
If lngStatus = 0 Then
MsgBox "EsbGetString() call #" & (intNumGSCalls) & " just read the string" & Chr$(10) & (strDataString) ' The server's translation of the query string
lstMessages (strDataString) ' Display each returned string on a line
intNumGSCalls = intNumGSCalls + 1 ' Increment now often EsbGetString() is called
Else
MsgBox "EsbGetString() failed: " & (lngStatus)
End If

lngStatus = EsbGetString(lngCtxHndl, strDataString, intDStringLen)
Loop

End Sub

********************************************************************
' EsbSendString() sends a string of data to the active database.
' This function should be called after EsbBeginReport(), EsbBeginUpdate(),
' or EsbBeginCalc()
**********************************************************************
Sub ESB_SendString()

Dim strQueryString As String
Dim arrQueryStrings(1 To 8) As String
Dim intCounter As Integer

arrQueryStrings(1) = "<PAGE (Market, Measures)"
arrQueryStrings(2) = "<COLUMN (Year, Scenario)"
arrQueryStrings(3) = "<ROW (Product)"
arrQueryStrings(4) = "<ICHILD Market"
arrQueryStrings(5) = "Qtr1 Qtr2"
arrQueryStrings(6) = "Actual Budget Variance"
arrQueryStrings(7) = "<ICHILD Product"
arrQueryStrings(8) = "!

'********************************************
'Send a series of query strings to the active database
'********************************************

For intCounter = 1 To 8
strQueryString = arrQueryStrings(intCounter)
lngStatus = EsbSendString(lngCtxHndl, strQueryString)

'*********
'Error Checking
'*********

If lngStatus = 0 Then
MsgBox "EsbSendString() sent query string # " & (intCounter) & " to the active database"
lstMessages (strQueryString)
Else
MsgBox "EsbSendString() failed: " & (lngStatus)
Exit Sub
End If
Next

End Sub
Sub ESB_QryFile()

Dim lngDbCtxHndl As Long
Dim lngRFCtxHndl As Long
Dim strAppName As String
Dim strDbName As String
Dim strReportFile As String
Dim intWhetherOutput As Integer
Dim intWhetherLock As Integer

lngDbCtxHndl = lngCtxHndl
lngRFCtxHndl = lngCtxHndl
strAppName = "Sample"
strDbName = "Basic"
strReportFile = "MyRpt01"

intWhetherOutput = ESB_TRUE ' If TRUE, data is output from server
intWhetherLock = ESB_FALSE  ' If TRUE, blocks are locked for update
                    ' If both are FALSE, report spec checked for syntax

lngStatus = EsbReportFile(lngDbCtxHndl, lngRFCtxHndl, strAppName, strDbName, _,
                         strReportFile, intWhetherOutput, intWhetherLock)

'**********************
'Error Checking
'**********************
If lngStatus = 0 Then
    MsgBox "The report file" & Chr$(10) & (strReportFile) & Chr$(10) _
        & "was sent to " & (strAppName) & (strDbName) & Chr$(10) _
        & "EsbGetString() will read the data"
Else
    MsgBox "EsbReportFile() failed: " & (lngStatus)
End If
End Sub

Sub ESB_QryStr()

Dim intWhetherOutput As Integer
Dim intWhetherLock As Integer
Dim strQueryString As String

strQueryString = "<DESC Year !" ' One query string
intWhetherOutput = ESB_TRUE ' If TRUE, data is output from server
intWhetherLock = ESB_FALSE ' If TRUE, blocks are locked for update
' If both are FALSE, report spec checked for syntax

lngStatus = EsbReport(lngCtxHndl, intWhetherOutput, intWhetherLock, strQueryString)

'**************
'Error Checking
'**************
If lngStatus = 0 Then
    MsgBox "The report specification" & Chr$(10) & (strQueryString) & Chr$(10) _
    & "was sent to the active database" & Chr$(10) _
    & "EsbGetString() will read the data"

    ' Server outputs data if intWhetherOutput = ESB_TRUE;
    ' ESB_GetString calls EsbGetString() to read the returned
    ' data until an empty string is returned
    '*****************************************************************************
    Call ESB_GetString
Else
    MsgBox "EsbReport() failed: " & (lngStatus)
End If

End Sub

'****************************************************************
'Sends an update specification to the active database from a file
'****************************************************************
Sub ESB_UpdFile()

Dim lngDbCtxHndl As Long
Dim lngUFCtxHndl As Long
Dim strAppName As String
Dim strDbName As String
Dim strUpdateFile As String
Dim intWhetherStore As Integer
Dim intWhetherUnlock As Integer

lngDbCtxHndl = lngCtxHndl
lngUFCtxHndl = lngCtxHndl
strAppName = "Sample"
strDbName = "Basic"
strUpdateFile = "CDupdtDb"

intWhetherStore = ESB_TRUE ' Database is updated & data is stored (on server)
intWhetherUnlock = ESB_TRUE ' Locked blocks are unlocked after data is updated

'*******************************************
'Lock database blocks before you update them
'*******************************************
Call ESB_LockDatabase

'*******************************************
'Send update file to the specified database
'*******************************************
lngStatus = EsbUpdateFile(lngDbCtxHndl, lngUFCtxHndl, strAppName, strDbName, _
strUpdateFile, intWhetherStore, intWhetherUnlock)

'**************
'Error Checking
'**************
If lngStatus = 0 Then
    MsgBox "The update file" & Chr$(10) & (strUpdateFile) & Chr$(10) _
    & "was sent to " & (strAppName) & (strDbName)
Else
    MsgBox "EsbUpdateFile() failed: " & (lngStatus)
End If

'****************************************
'Calls error checking sub routine
'****************************************
Call ESB_ListErrorStackMsgs

End Sub

Sub ESB_BeginReport()
Dim intWhetherOutput As Integer
Dim intWhetherLock   As Integer
Dim strQueryString   As String
intWhetherOutput = ESB_TRUE  ' If TRUE, data is output from server
intWhetherLock = ESB_FALSE   ' If TRUE, blocks are locked for update
    ' If both are FALSE, report spec checked for syntax
lngStatus = EsbBeginReport(lngCtxHndl, intWhetherOutput, intWhetherLock)

'**************
'Error Checking
'**************
If lngStatus = 0 Then
    MsgBox "EsbBeginReport() succeeded"
Else
    MsgBox "EsbBeginReport() failed: " & (lngStatus)
End If
End Sub

Sub ESB_EndReport()
lngStatus = EsbEndReport(lngCtxHndl)

End Sub
Error Checking

If lngStatus = 0 Then
    MsgBox "EsbEndReport() succeeded"
Else
    MsgBox "EsbEndReport() failed: " & (lngStatus)

' Calls error checking sub routine
Call ESB_ListErrorStackMsgs
Exit Sub
End If
End Sub

Executes a calc script against the active database from a file

Sub ESB_CalcFile()
    Dim lngDbCtxHndl      As Long
    Dim lngCSCtxHndl      As Long
    Dim strAppName        As String
    Dim strDbName         As String
    Dim strCalcScriptFile As String
    Dim intWhetherCalc    As Integer  ' If TRUE, the calc script is executed
    lngDbCtxHndl = lngCtxHndl
    lngCSCtxHndl = lngCtxHndl
    strAppName = "Sample"
    strDbName = "Basic"
    strCalcScriptFile = "Calc5Dim"
    intWhetherCalc = ESB_TRUE

    lngStatus = EsbCalcFile(lngDbCtxHndl, lngCSCtxHndl, strAppName, strDbName, _,
                            strCalcScriptFile, intWhetherCalc)

Error Checking

If lngStatus = 0 Then
    MsgBox (strAppName) & (strDbName) & " is being calculated" & Chr$(10) _
        & "using the calc script in " & (strCalcScriptFile)
' Call Esb_GetProcessState to get the current state of calc
Call ESB_GetProcessState
Else
MsgBox "EsbCalcFile() failed: " & (lngStatus)

'************************************************
'Calls error checking sub routine
'************************************************
Call ESB_ListErrorStackMsgs
End If
End Sub

'*************************************************************************
'Clear data from the active database
'*************************************************************************
Sub ESB_ClrData()

lngStatus = EsbClearDatabase(lngCtxHndl)

'*************************************************************************
'Begin error checking
'*************************************************************************
If lngStatus = 0 Then
    MsgBox "WAIT!! Data is being cleared from " & (strActiveApp) & (strActiveDb)

'*************************************************************************
'Call Esb_GetProcessState to get the current state of process
'*************************************************************************
Call ESB_GetProcessState
Else
    MsgBox "EsbClearDatabase() failed: " & (lngStatus)

'*************************************************************************
'Calls error checking sub routine
'*************************************************************************
Call ESB_ListErrorStackMsgs
End If
End Sub

'*************************************************************************
'Import data from different sources
'*************************************************************************
Sub ESB_LdData()

Dim structRulesFile As ESB_OBJDEF_T
Dim structDataFile As ESB_OBJDEF_T
Dim structSQLSource As ESB_MBRUSER_T

structDataFile.hCtx = lngCtxHndl
structDataFile.Type = ESB_OBJTYPE_TEXT
structDataFile.AppName = "Sample"
structDataFile.DbName = "Basic"
structDataFile.FileName = "CalcDat"

strErrorsOnLoadFile = "ErrsOnLd.txt"

intWhetherAbortOnError = ESB_TRUE

' Import data from CalcDat.txt to Sample/Basic
lngStatus = EsbImport(lngCtxHndl, structRulesFile, structDataFile, structSQLSource, _
    strErrorsOnLoadFile, intWhetherAbortOnError)

'**************
' Error Checking
'**************
If lngStatus = 0 Then
    MsgBox "WAIT!! Data from " & (structDataFile.FileName) & Chr$(10) _
    & "is being imported to " & (structDataFile.AppName) & (structDataFile.DbName)
    ' Call Esb_GetProcessState to get the current state of import
    Call ESB_GetProcessState
Else
    MsgBox "EsbImport() failed: " & (lngStatus)
    ' Calls error checking sub routine
    Call ESB_ListErrorStackMsgs
End If
End Sub

'*******************************************************************
' ESB_LockDatabase() calls EsbReportFile() to lock blocks for update
'*******************************************************************
Sub ESB_LockDatabase()

    Dim lngDbCtxHndl As Long
    Dim lngRFCtxHndl As Long
    Dim strAppName As String
    Dim strDbName As String
    Dim strReportFile As String
    Dim intWhetherOutput As Integer  ' If TRUE, data is output from server
    Dim intWhetherLock As Integer  ' If TRUE, blocks are locked for update

    lngDbCtxHndl = lngCtxHndl
    lngRFCtxHndl = lngCtxHndl
    strAppName = "Sample"
    strDbName = "Basic"
    strReportFile = "CDlockDb"

    intWhetherOutput = ESB_FALSE     ' FALSE: no data is output from server
    intWhetherLock = ESB_TRUE        ' TRUE: blocks are locked for update

    lngStatus = EsbReportFile(lngDbCtxHndl, lngRFCtxHndl, strAppName, strDbName, _
        strReportFile, intWhetherOutput, intWhetherLock)
'Error Checking

If lngStatus = 0 Then
    MsgBox "The report file" & Chr$(10) & (strReportFile) & Chr$(10) _
    & "was sent to " & (strAppName) & (strDbName) & Chr$(10) _
    & "Blocks are locked for update" & Chr$(10) _
    & "EsbUpdateFile() will update the CalcData database"
Else
    MsgBox "EsbReportFile() failed: " & (lngStatus)
End If
End Sub

Sub ESB_GetProcessState()
    Dim structProcessState As ESB_PROCSTATE_T

    lngStatus = EsbGetProcessState(lngCtxHndl, structProcessState)
    Do Until structProcessState.State = ESB_STATE_DONE
        lngStatus = EsbGetProcessState(lngCtxHndl, structProcessState)
    Loop

    MsgBox "Asynchronous Process Completed"
End Sub

Sub ESB_ListErrorStackMsgs()
    Const intMsgLen = 256
    Dim strMsg As String * intMsgLen

    lngStatus = EsbGetMessage(lngInstHndl, intMsgLev, lngMsgNmbr, _
    strMsg, intMsgLen)
    Dim intStackNmbr As Integer

    intStackNmbr = 1

    Do While Mid$(strMsg, 1, 1) <> Chr$(0)
        lstMessages "MESSAGE ON ERROR STACK:"
        lstMessages "Stack #" & (intStackNmbr)
        intStackNmbr = intStackNmbr + 1
    Loop
End Sub
lstMessages "Level #" & (intMsgLev)
lstMessages "Message #" & (lngMsgNmbr)
lstMessages (strMsg)
    intStackNmbr = intStackNmbr + 1
    lngStatus = EsbGetMessage(lngInstHndl, intMsgLev, lngMsgNmbr, strMsg, intMsgLen)
Loop
End Sub

Sub lstMessagesClear()
    frmRprts.lstMessages.Clear
End Sub
Shared Services Migration and User Management API Example

/*
Declaration of Include files
*/

#if defined _WIN32 || defined _WINDOWS
#include
#endif
#include
#include
#include
#pragma pack (1)
#include
#include
#pragma pack ()

 предложения Example Usage Starts Here ------------------------*/

/comments ESS_FUNC_M EssSetSSSecurityMode(ESS_HCTX_T   hCtx,
ESS_USHORT_T  Option,
ESS_STR_T     Password);

ESS_FUNC_M ESS_SS_SetSSSecurityMode(ESS_HCTX_T   hCtx)
{
    ESS_STS_T         sts = ESS_STS_NOERR;
    ESS_STR_T         newPassword = ESS_NULL;
    ESS_USHORT_T      option;

    /* New Shared Services Native User Password Option:
     * 0 to use user provided password
     * 1 to use the user name as password
     * 2 to automatically generate a password
     */
    option = 1; /* Using user name as password */

    if(sts)
        printf("Failed to migrate Analytic Services Server to Shared Services mode.");

}
ESS_FUNC_M EssGetEssbaseSecurityMode (ESS_HCTX_T hCtx,
    ESS_PSECURITY_MODE_T mode);

ESS_FUNC_M ESS_SS_GetEssbaseSecurityMode(ESS_HCTX_T  hCtx)
{
    ESS_STS_T           sts = ESS_STS_NOERR;
    ESS_SECURITY_MODE_T mode;

    sts = EssGetEssbaseSecurityMode(hCtx, &mode);

    if(sts)
    {
        printf("Failed to get Essbase Security mode.\n");
    }
    else
    {
        printf("Essbase Security Mode            : %d\n", mode);
    }
    return(sts);
}

ESS_FUNC_M ESS_SS_ListSSMigrFailedUsers(ESS_HCTX_T  hCtx, ESS_HINST_T hInst)
{
    ESS_STS_T         sts = ESS_STS_NOERR;
    ESS_PUSERNAME_T   pNativeUserList = NULL;
    ESS_USHORT_T      Count = 0,
    index;

    sts = EssListSSMigrFailedUsers(hCtx, &Count, &pNativeUserList);

    if (!sts)
    {
        if (Count && pNativeUserList)
        {
            printf ("\n------- User List -------\n\n");

            for (index = 0; index < Count; index++)
            {
                if (pNativeUserList[index])
                    printf (%s
, pNativeUserList[index]);
            }

            EssFree(hInst, pNativeUserList);
        }
        else
            printf("\nUser list is empty\n");
}  
else  
    printf("Failed to get Shared Services migration failed Users list.\n");  
  
return (sts);  
}  
/*
ESS_FUNC_M EssListSSMigrFailedGroups(ESS_HCTX_T,  
    ESS_PUSHORT_T,  
    ESS_PPUSERNAME_T);*/

ESS_FUNC_M ESS_SS_ListSSMigrFailedGroups(ESS_HCTX_T  hCtx, ESS_HINST_T hInst)  
{  
    ESS_STS_T         sts = ESS_STS_NOERR;  
    ESS_PUSERNAME_T   pNativeUserList = NULL;  
    ESS_USHORT_T      Count = 0,  
        index;  
    
sts = EssListSSMigrFailedGroups(hCtx, &Count, &pNativeUserList);  
    
    if (!sts)  
    {  
        if (Count && pNativeUserList)  
        {  
            printf ("\n------- Group List -------\n\n");  
            for (index = 0; index < Count; index++)  
            {  
                if (pNativeUserList[index])  
                    printf ("%s\n", pNativeUserList[index]);  
            }  
        }  
        EssFree(hInst, pNativeUserList);  
    }  
    else  
        printf("\nGroup list is empty\n\n");  
    
    return ("Failed to get Shared Services migration failed Groups list.\n");  
}  
/*
ESS_FUNC_M EssSetUserToSS (ESS_HCTX_T    hCtx,  
    ESS_STR_T     UserName,  
    ESS_USHORT_T  Option,  
    ESS_STR_T     Password);*/

ESS_FUNC_M ESS_SS_SetUserToSS(ESS_HCTX_T  hCtx, ESS_HINST_T hInst)  
{  
    ESS_STS_T         sts = ESS_STS_NOERR;  
    ESS_USHORT_T      option;  
    ESS_STR_T         userName = ESS_NULL;  
    ESS_STR_T         newPassword = ESS_NULL;  

sts = EssAlloc(hInst, sizeof(ESS_USERNAME_T), &userName);
if(sts)
    return (sts);
memset(userName, 0, sizeof(ESS_USERNAME_T));
strcpy( userName, "essexer");

/* New Shared Services Native User Password Option:
 * 0 to use user provided password
 * 1 to use the user name as password
 * 2 to automatically generate a password
 **/
option = 1; /* Using user name as password */

sts = EssSetUserToSS(hCtx, userName, option, newPassword);
if(sts)
    printf("Failed to migrate User %s to Shared Services mode.\n", userName);
if (userName)
    EssFree(hInst, userName);

return (sts);

} /*
ESS_FUNC_M EssSetGroupToSS (ESS_HCTX_T    hCtx,
                             ESS_STR_T     GroupName);
*/

ESS_FUNC_M ESS_SS_SetGroupToSS(ESS_HCTX_T  hCtx, ESS_HINST_T hInst)
{
    ESS_STS_T         sts = ESS_STS_NOERR;
    ESS_STR_T         groupName = ESS_NULL;

    sts = EssAlloc(hInst, sizeof(ESS_USERNAME_T), &groupName);
    if(sts)
        return (sts);
    memset(groupName, 0, sizeof(ESS_USERNAME_T));
    strcpy( groupName, "essgrp");

    sts = EssSetGroupToSS(hCtx, groupName);
    if(sts)
        printf("Failed to migrate Group %s to Shared Services mode.\n", groupName);
    if (groupName)
        EssFree(hInst, groupName);

    return (sts);
}

/*
ESS_FUNC_M EssSetUsersToSS(ESS_HCTX_T    hCtx,
                             ESS_USHORT_T  Option,
                             ESS_STR_T     Password);
*/
ESS(FuncM) ESS_SS_SetUsersToSS(ESS_CTX_T hCtx, ESS_HINST_T hInst)
{
    ESS_STS_T        sts = ESS_STS_NOERR;
    ESS_USHORT_T     option;
    ESS_STR_T        newPassword = ESS_NULL;

    /* New Shared Services Native User Password Option:
    *   0 to use user provided password
    *   1 to use the user name as password
    *   2 to automatically generate a password
    **/

    option = 0; /* Using user provided password */

    sts = EssAlloc(hInst, sizeof(ESS_PASSWORD_T), &newPassword);
    if (!sts)
        return (sts);
    memset(newPassword, 0, sizeof(ESS_PASSWORD_T));
    strcpy(newPassword, "password");

    sts = EssSetUsersToSS(hCtx, option, newPassword);
    if (!sts)
        printf("Failed to migrate Users to Shared Services mode.\n");

    if (newPassword)
        EssFree(hInst, newPassword);

    return (sts);
}

ESS(FuncM) ESS_SS_SetGroupsToSS(ESS_CTX_T hCtx);

ESS(FuncM) ESS_SS_SetGroupsToSS(ESS_CTX_T hCtx)
{
    ESS_STS_T        sts = ESS_STS_NOERR;

    sts = EssSetGroupsToSS(hCtx);
    if (!sts)
        printf("Failed to migrate Groups to Shared Services mode.\n");

    return (sts);
}

ESS(FuncM) EssSetEasLocation (ESS_CTX_T hCtx, ESS_CTX_T hInst);

ESS(FuncM) ESS_SS_SetEasLocation (ESS_CTX_T hCtx, ESS_HINST_T hInst)
{
    ESS_STS_T        sts = ESS_STS_NOERR;
    ESS_STR_T        easLoc = ESS_NULL;

    /* Eas Location */
sts = EssAlloc(hInst, sizeof(ESS_PATHLEN), &easLoc);
if (sts)
    return (sts);
memset(easLoc, 0, sizeof(ESS_PATHLEN));
strcpy(easLoc, "localhost:10080");

sts = EssSetEasLocation(hCtx, easLoc);
if (sts)
    printf("Failed to set EAS Location.\n");
if (easLoc)
    EssFree(hInst, easLoc);

return (sts);
}

ESS_FUNC_M ESS_SS_ReRegisterApplication (ESS_HCTX_T hCtx,
                           ESS.ReadString_T AppName,
                           ESS.BOOL_T AllApps);

ESS_FUNC_M ESS_SS_ReRegisterApplication(ESS_HCTX_T hCtx, ESS_HINST_T hInst)
{
    ESS_STS_T         sts = ESS_STS_NOERR;
    ESS.BOOL_T        allApps;
    ESS.ReadString_T appName = ESS_NULL;

    sts = EssAlloc(hInst, sizeof(ESS_APPNAME_T), &appName);
    if (sts)
        return (sts);
    memset(appName, 0, sizeof(ESS_APPNAME_T));
    strcpy(appName, "Sample");

    /* Do you want All applications re-registered?
     * Enter ESS_TRUE for Yes
     * ESS_FALSE for No
     **/
    allApps = ESS_FALSE; /* Re-registering only 1 application */
    sts = EssReRegisterApplication(hCtx, appName, allApps);
    if (sts)
        printf("Failed to Re-register Application %s.\n", appName);
    if (appName)
        EssFree(hInst, appName);

    return (sts);
}
DEVELOPMENT

#include "ess.h"

int main()
{
    ESS_HINST_T    hInst;
    ESS_HCTX_T      hCtx;
    ESS_SVRNAME_T   svrName   =   "localhost";
    ESS_USERNAME_T  userName   =   "essexer";
    ESS_PASSWORD_T  pswd       =   "password";
    ESS_STS_T       sts        =   ESS_STS_NOERR;
    ESS_USHORT_T    Items;
    ESS_PAPPDB_T    pAppsDbs   =   ESS_NULL;

    ESS_INIT_T InitStruct = (ESS_API_VERSION, 
                             ESS_NULL, 
                             0L, 
                             255, 
                             ESS_NULL, 
                             ESS_NULL, 
                             ESS_NULL, 
                             ESS_NULL, 
                             ESS_NULL, 
                             ESS_NULL, 
                             ESS_NULL, 
                             ESS_NULL, 
                             0L);

    sts = EssInit(&InitStruct, &hlnst);
    if (sts)
    {
        printf("EssInit failure: %ld\n", sts);
        exit ((int) sts);
    }

    sts = EssLogin(hInst, svrName, userName, pswd, &Items, &pAppsDbs, &hCtx);
    if (sts)
    {
        printf("EssLogin failure: %ld\n", sts);
        exit ((int) sts);
    }

    sts = ESS_SS_SetSSSecurityMode(hCtx);
    if (sts)
printf("ESS_SS_SetSSSecurityMode failed: %ld\n", sts);
sts = ESS_SS_GetEssbaseSecurityMode(hCtx);
if (sts)
    printf("ESS_SS_GetEssbaseSecurityMode failed: %ld\n", sts);
sts = ESS_SS_ListSSMigrFailedUsers(hCtx, hInst);
if (sts)
    printf("ESS_SS_ListSSMigrFailedUsers failed: %ld\n", sts);
sts = ESS_SS_ListSSMigrFailedGroups(hCtx, hInst);
if (sts)
    printf("ESS_SS_ListSSMigrFailedGroups failed: %ld\n", sts);
sts = ESS_SS_SetUserToSS(hCtx, hInst);
if (sts)
    printf("ESS_SS_SetUserToSS failed: %ld\n", sts);
sts = ESS_SS_SetGroupToSS(hCtx, hInst);
if (sts)
    printf("ESS_SS_SetGroupToSS failed: %ld\n", sts);
sts = ESS_SS_SetUsersToSS(hCtx, hInst);
if (sts)
    printf("ESS_SS_SetUsersToSS failed: %ld\n", sts);
sts = ESS_SS_SetGroupsToSS(hCtx);
if (sts)
    printf("ESS_SS_SetGroupsToSS failed: %ld\n", sts);
sts = ESS_SS_SetEasLocation(hCtx, hInst);
if (sts)
    printf("ESS_SS_SetEasLocation failed: %ld\n", sts);
sts = ESS_SS_ReRegisterApplication(hCtx, hInst);
if (sts)
    printf("ESS_SS_ReRegisterApplication failed: %ld\n", sts);
sts = EssLogout(hCtx);
stas = EssTerm(hInst);
In This Appendix

Name Limits ................................................................. 1727
Drill-through URL Limits ................................................... 1729

Name Limits

Subtopics

- Essbase Server (Host) Name Limits
- Application Name Limits
- Database Name Limits
- Filter Name Limits
- Group Name Limits
- Object Name Limits
- Password Limits
- User Name Limits

Essbase Server (Host) Name Limits

- Non-Unicode application limit: 1024 bytes
- Unicode-mode application limit: 1024 characters

Application Name Limits

- Non-Unicode application limit: 8 bytes
- Unicode-mode application limit: 30 characters

Application names can contain all special characters allowed in DOS file names. No spaces, commas, tabs, slashes, backslashes, or periods are allowed. The use of some special characters is not recommended because they are often used by the operating system (for example, @, $, %, and &).
Database Name Limits

- Non-Unicode application limit: 8 bytes
- Unicode-mode application limit: 30 characters

Database names can contain all special characters allowed in DOS file names. No spaces, commas, tabs, slashes, backslashes, or periods are allowed. The use of some special characters is not recommended because they are often used by the operating system (for example, @, $, %, and &).

Filter Name Limits

- Non-Unicode application limit: 256 bytes
- Unicode-mode application limit: 256 characters

Group Name Limits

- Non-Unicode application limit: 256 bytes
- Unicode-mode application limit: 256 characters

Object Name Limits

- Non-Unicode application limit: 8 bytes
- Unicode-mode application limit: 30 characters

Object names can contain all special characters allowed in DOS file names. No spaces, commas, backslashes, or periods are allowed.

Password Limits

- Non-Unicode application limit: 100 bytes
- Unicode-mode application limit: 100 characters

User Name Limits

- Non-Unicode application limit: 256 bytes
- Unicode-mode application limit: 256 characters

User names are not case sensitive and must not contain the backslash character (\).
Drill-through URL Limits

The following limits apply to drill-through URLs:

- The number of drill-through URLs per database is limited to 255.
- The number of drillable regions in a drill-through URL is limited to 256.
- The number of characters per drillable region is limited to 65536.
Index

Symbols
.BAS files, 32
.H files, 32

A
access
security requirements, 681, 1447
accessing file objects
C, 83
VB, 1122
active applications and databases
C, 90
VB, 1128
AIX programs
building, 33
alias table functions
C Main API, 199
C Outline API, 713
VB Main API, 1199
VB Outline API, 1465
allocating memory, 681
API
architecture with server, 25
API example
Shared Services migration and user management, 1719
API libraries, 42
API overview, 25
appdb.vbp sample program, 1695
application functions
C Main API, 200
VB Main API, 1200
ARBORMSGPATH
defined, 40
architecture of API and server, 25
architecture of Grid API, 992
array types
C, 99
attributes
C constants, 100
attributes functions
C Main API, 201
C Outline API, 714
VB Main API, 1201
VB Outline API, 1466
audience for this document, 26
autologin, 41
B
bitmask data types
C, 96
VB, 1161
boolean flags
VB, 1163
building a program on UNIX, 33
byte-aligned structures, 32
C
C constant definitions, 100
C Grid API
constants, 997
data types, 1004
error codes, 1115
structures, 1006
C language types, 95
calc script specification strings
C, 92
VB, 1130
calculating the database
C, 92
VB, 1130
calculation functions
C Main API, 210
Callback functions, 141
calling C API functions, 88
calling VB API functions, 1125
checking calculation progress
  C, 92
  VB, 1131
cleanup functions
  C Outline API, 718
  VB Outline API, 1469
CLIENT directory
  customizing, 39
common problems and solutions, 1132
compilers
  supported, 29
concepts
  Grid API, 991
connecting to an Essbase Server
  C, 90
  VB, 1128
constant and structure definitions for partitions
  C, 109
constant definitions
  C Grid API, 997
constants
  C definitions, 100
  C definitions of linked reporting objects, 107
definitions
  C Grid API, 997
  for linked reporting objects, 1155
  for LROs, 107
  VB definitions of linked reporting objects, 1155
  Visual Basic definitions, 1153
corect handles
  C, 82
  VB, 1121
conventions
  document, 26
document systems, 995
cs1.c sample program, 1665
cs2.c sample program, 1672
cs3.c sample program, 1681
currency conversion category values
  C constants, 693
  VB constants, 1455
customizing memory in C programs, 84
customizing message handling
  C, 86
customizing the run-time environment, 38

data types
  C, 95, 96
  C Grid API, 1004
  VB, 1159
database functions
  C Main API, 201
  VB Main API, 1201
database member functions
  C Main API, 202
  VB Main API, 1202
declarations
  understanding in Visual Basic, 1119
  VB Outline API, 1451
default calc scripts
  C, 1131
  VB, 1131
definitions
  C API, 100
  VB API, 1157
dimension categories (tags)
  C constants, 694
  VB constants, 1456
dimension tag
  C constants, 103
  VB constants, 1154
directory structures, 37
disconnecting from the Essbase Server
  C, 93
  VB, 1131
Discover method, 1622
djfslk, 450
drill-through functions
  C Main API, 203
  VB Main API, 1202
drill-through URL limits, 1729
dynamic libraries provided with API, 42
dynamic time series functions
  C Outline API, 715
  VB Outline API, 1466

empty strings in Visual Basic, 1119
error codes
  C Grid API, 1115
error handling, 679

E
EEP_TRANSACTION_REQSPECIFIC_T, 188
Index
VB Outline API, 1445
error return values
C Outline API, 685
VB Outline API, 1451
ESB32.BAS, 1119
ESB_APPDB_T, 1165
ESB_APPINFO_T, 1165
ESB_APPINFOEX_T, 1166
ESB_CELLADDR_API_T, 1156
ESB_DBINFO_T, 1171
ESB_DBREQINFO_T, 1173
ESB_DBSTATE_T, 1173
ESB_DBSTATS_T, 1176
ESB_DIMENSIONINFO_T, 1177
ESB_DIMSTATS_T, 1178
ESB_DURLINFO_T, 1179
ESB_GENLEVELNAME_T, 1459
ESB_GLOBAL_T, 1179
ESB_GROUPAPP_T, 1192
ESB_GROUPDB_T, 1192
ESB_GROUPINFO_T, 1193
ESB_INIT_T, 1180
ESB_LOCKINFO_T, 1181
ESB_LRODESC_API_T, 1156
ESB_LROHANDLE_API_T, 1156
ESB_LROINFO_API_T, 1157
ESB_MBRALT_T, 1181
ESB_MBRCOUNTS_T, 1459
ESB_MBRINFO_T, 1460
ESB_MBRUSER_T, 1182
ESB_MEMBERINFO_T, 1182
ESB_OBJDEF_T, 1184
ESB_OBJINFO_T, 1184
ESB_OUTERROR_T, 1184
ESB_OUTERR_T, 1462
ESB_OUTLINEINFO_T, 1463
ESB_PART_CONNECT_INFO_T, 1185
ESB_PART_DEFINED_T, 1186
ESB_PART_INFO_T, 1186
ESB_PART_REPL_T, 1188
ESB_PARTOTL_QRY_FILTER_T, 1188
ESB_PARTOTL_QUERY_T, 1189
ESB_PARTSCLCT_T, 1189
ESB_PREDICATE_T, 1464
ESB_PROCSTATE_T, 1190
ESB_RATEINFO_T, 1191
ESB_TIMERECORD_T, 1191
ESB_USERAPP_T, 1192
ESB_USERDB_T, 1192
ESB_USERINFO_T, 1193
ESB_VARIABLE_T, 1196
EsbAddToGroup, 1209
EsbArchive, 1210
EsbArchiveBegin, 1210
EsbArchiveEnd, 1211
EsbAutoLogin, 1212
EsbBeginCalc, 1215
EsbBeginDataload, 1217
EsbBeginReport, 1217
EsbBeginUpdate, 1219
EsbBuildDimension, 1220
EsbBuildDimFile, 1222
EsbBuildDimStart, 1224
EsbCalc, 1225
EsbCalcFile, 1227
EsbCancelProcess, 1228
EsbCheckMemberName, 1232
EsbClearActive, 1233
EsbClearAliases, 1233
EsbClearDatabase, 1234
EsbClsSpanRelationalSource, 1235
EsbCommitDatabase, 1236
EsbCopyApplication, 1236
EsbCopyDatabase, 1237
EsbCopyFilter, 1238
EsbCopyObject, 1240
EsbCreateApplication, 1242
EsbCreateDatabase, 1243
EsbCreateExtUser, 1245
EsbCreateFilter, 1246
EsbCreateGroup, 1246
EsbCreateLocalContext, 1247
EsbCreateLocationAlias, 1248
EsbCreateObject, 1250
EsbCreateStorageTypedApplication, 1251
EsbCreateUser, 1253
EsbCreateVariable, 1254
EsbDefaultCalc, 1255
EsbDeleteApplication, 1256
EsbDeleteDatabase, 1257
EsbDeleteFilter, 1258
EsbDeleteFromGroup, 1259
EsbDeleteGroup, 1260
EsbDeleteLocalContext, 1261
EsbDeleteLocationAlias, 1262
EsbDeleteLogFile, 1263
EsbDeleteObject, 1264
EsbDeleteUser, 1266
EsbDeleteVariable, 1266
EsbDisplayAlias, 1267
EsbEndCalc, 1268
EsbEndDataload, 1269
EsbEndReport, 1270
EsbEndUpdate, 1271
EsbExport, 1272
EsbGetActive, 1273
EsbGetAlias, 1274
EsbGetAPIVersion, 1275
EsbGetApplicationAccess, 1276
EsbGetApplicationInfo, 1278
EsbGetApplicationInfoEx, 1279
EsbGetApplicationState, 1280
EsbGetAssociatedAttributesInfo, 1281
EsbGetAttributeInfo, 1283
EsbGetAttributeSpecifications, 1285
EsbGetCalcList, 1287
EsbGetCurrencyRateInfo, 1290
EsbGetDatabaseAccess, 1292
EsbGetDatabaseInfo, 1293
EsbGetDatabaseInfoEx, 1294
EsbGetDatabaseNote, 1295
EsbGetDatabaseState, 1296
EsbGetDatabaseStats, 1297
EsbGetDefaultCalc, 1299
EsbGetDimensionInfo, 1300
EsbGetFilter, 1302
EsbGetFilterList, 1303
EsbGetFilterRow, 1304
EsbGetGlobalState, 1305
EsbGetGroup, 1306
EsbGetGroupList, 1307
EsbGetLocalPath, 1308
EsbGetLocationAliasList, 1310
EsbGetLogFile, 1311
EsbGetMemberCalc, 1312
EsbGetMemberInfo, 1314
EsbGetMessage, 1315
EsbGetNextItem, 1316
EsbGetObject, 1319
EsbGetObjectInfo, 1321
EsbGetProcessState, 1322
EsbGetString, 1323
EsbGetStringBuf, 1324
EsbGetUser, 1325
EsbGetUserEx, 1326
EsbGetVariable, 1327
EsbGetVersion, 1328
EsbImport, 1328
EsbInit, 1331
EsbKillRequest, 1332
EsbListAliases, 1333
EsbListApplications, 1334
EsbListCalcFunctions, 1335
EsbListConnections, 1336
EsbListCurrencyDatabases, 1338
EsbListDatabases, 1339
EsbListDbFiles, 1340
EsbListFilters, 1342
EsbListGroups, 1343
EsbListLocks, 1344
EsbListLogins, 1346
EsbListObjects, 1346
EsbListRequests, 1347
EsbListUsers, 1349
EsbListUsersEx, 1350
EsbListVariables, 1351
EsbLoadAlias, 1353
EsbLoadApplication, 1354
EsbLoadDatabase, 1354
EsbLockObject, 1355
EsbLogin, 1357
EsbLoginSetPassword, 1358
EsbLogout, 1360
EsbLogoutUser, 1361
EsbLogSize, 1362
EsbLROAddObject, 1363
EsbLRODeleteCellObjects, 1364
EsbLRODeleteObject, 1366
EsbLROGetCatalog, 1367
EsbLROGetMemberCombo, 1368
EsbLROGetObject, 1370
EsbLROGetObjects, 1371
EsbLROListObjects, 1373
EsbLROUpdateObject, 1374
EsbOtlAddAliasCombination, 1470
EsbOtlAddDimension, 1472
EsbOtlAddMember, 1474
EsbOtlAssociateAttributeDimension, 1478
Index 1735

EsbOtlAssociateAttributeMember, 1479
EsbOtlClearAliasTable, 1481
EsbOtlClearAliasTableLanguages, 1482
EsbOtlCloseOutline, 1484
EsbOtlCopyAliasTable, 1485
EsbOtlCreateAliasTable, 1486
EsbOtlDeleteAliasCombination, 1487
EsbOtlDeleteAliasTable, 1488
EsbOtlDeleteDimension, 1489
EsbOtlDeleteDTSMemberAlias, 1491
EsbOtlDeleteGenName, 1492
EsbOtlDeleteLevelName, 1493
EsbOtlDeleteMember, 1494
EsbOtlDeleteMemberAlias, 1495
EsbOtlDeleteMemberFormula, 1496
EsbOtlDeleteUserAttribute, 1497
EsbOtlEnableDTSMember, 1501
EsbOtlFindAlias, 1502
EsbOtlFindMember, 1505
EsbOtlFreeMember, 1506
EsbOtlGenerateCurrencyOutline, 1507
EsbOtlGetAliasTableLanguages, 1509
EsbOtlGetChild, 1515
EsbOtlGetDTSMemberAlias, 1518
EsbOtlGetEnabledDTSMembers, 1519
EsbOtlGetFirstMember, 1520
EsbOtlGetGenName, 1521
EsbOtlGetGenNames, 1522
EsbOtlGetLevelName, 1524
EsbOtlGetLevelNames, 1525
EsbOtlGetMemberAlias, 1527
EsbOtlGetMemberFormula, 1528
EsbOtlGetMemberInfo, 1529
EsbOtlGetMemberLastFormula, 1531
EsbOtlGetNextAliasCombination, 1532
EsbOtlGetNextSharedMember, 1533
EsbOtlGetNextSibling, 1535
EsbOtlGetOutlineInfo, 1536
EsbOtlGetParent, 1537
EsbOtlGetPrevSibling, 1538
EsbOtlGetUpdateTime, 1539
EsbOtlGetUserAttributes, 1540
EsbOtlMoveMember, 1541
EsbOtlNewOutline, 1543
EsbOtlOpenOutline, 1544
EsbOtlOpenOutlineQuery, 1545
EsbOtlQueryMembers, 1548
EsbOtlQueryMembersByName, 1551
EsbOtlRenameAliasTable, 1555
EsbOtlRenameMember, 1556
EsbOtlRestructure, 1557
EsbOtlSetAliasTableLanguage, 1559
EsbOtlSetDTSMemberAlias, 1563
EsbOtlSetGenName, 1564
EsbOtlSetLevelName, 1565
EsbOtlSetMemberAlias, 1566
EsbOtlSetMemberFormula, 1568
EsbOtlSetMemberInfo, 1569
EsbOtlSetOutlineInfo, 1572
EsbOtlSetUserAttribute, 1573
EsbOtlSortChildren, 1574
EsbOtlVerifyOutline, 1575
EsbPartitionApplyOtlChangeFile, 1376
EsbPartitionApplyOtlChangeRecs, 1377
EsbPartitionGetAreaCellCount, 1378
EsbPartitionGetList, 1379
EsbPartitionGetOtlChanges, 1380
EsbPartitionGetRepCells, 1381
EsbPartitionPurgeOtlChangeFile, 1382
EsbPartitionPutRepCells, 1383
EsbPartitionReadOtlChangeFile, 1385
EsbPartitionResetOtlChangeTime, 1386
EsbPutObject, 1387
EsbQueryDatabaseMembers, 1388
EsbRemoveAlias, 1391
EsbRemoveLocks, 1392
EsbRenameApplication, 1393
EsbRenameDatabase, 1394
EsbRenameFilter, 1395
EsbRenameGroup, 1397
EsbRenameObject, 1397
EsbRenameUser, 1399
EsbReport, 1400
EsbReportFile, 1402
EsbResetUser, 1403
EsbRestore, 1404
EsbSendString, 1405
EsbSetActive, 1405
EsbSetAlias, 1407
EsbSetApplicationAccess, 1407
EsbSetApplicationState, 1408
EsbSetCalcList, 1410
EsbSetDatabaseAccess, 1411
EsbSetDatabaseNote, 1412
EsbSetDatabaseState, 1413
EsbSetDefaultCalc, 1415
EsbSetDefaultCalcFile, 1415
EsbSetFilter, 1417
EsbSetFilterList, 1418
EsbSetFilterRow, 1419
EsbSetGlobalState, 1420
EsbSetGroup, 1421
EsbSetGroupList, 1422
EsbSetPassword, 1423
EsbSetPath, 1424
EsbSetSpanRelationalSource, 1425
EsbSetUser, 1426
EsbSetUserEx, 1427
EsbShutdownServer, 1428
EsbTerm, 1429
EsbUnloadApplication, 1430
EsbUnloadDatabase, 1431
EsbUnlockObject, 1431
EsbUpdate, 1433
EsbUpdateFile, 1435
EsbValidateDB, 1437
EsbValidateHCtx, 1438
EsbVerifyFilter, 1439
EsbVerifyFilterRow, 1440
EsbWriteToLogFile, 1441
ESS_APPDB_T, 114
ESS_APPINFO_T, 115
ESS_APPINFOEX_T, 116
ESS_APPSTATE_T, 117
ESS_BLDLDL_STATE_T, 121
ESS_CELLADDR_API_T, 108
ESS_CONNECTINFOEX_T, 122
ESS_DBINFO_T, 124
ESS_DBREQINFO_T, 126
ESS_DBSTATE_T, 127
ESS_DBSTATS_T, 130
ESS_DIMENSIONINFO_T, 132
ESS_DIMSTATS_T, 133
ESS_DISKVOLUME_REPLACE_T, 137
ESS_DTAPICOLUMN_T, 134
ESS_DTAPIDATA_T, 134
ESS_DTAPIHEADER_T, 135
ESS_DTAPIINFO_T, 135
ESS_DTAPIREPORT_T, 136
ESS_DTBUFFER_T, 136
ESS_DTDATA_T, 136
ESS_DTHEADER_T, 137
ESS_EXTUSERINFO_T, 138
ESSFUNC_M, 84
ESS_GENLEVELNAME_T, 698
ESS_GLOBAL_T, 140
ESS_GROUPDBEX_T, 191
ESS_GROUPINFO_T, 192
ESS_GROUPINFOID_T, 194
ESS_INIT_T, 141
ESS_INIT_T structure, 84
ESS_LOAD_BUFFER_T, 145
ESS_LOCKINFO_T, 146
ESS_LOCKINFOEX_T, 146
ESS_LOG_DATALOAD_T, 147
ESS_LRODESC_API_T, 108
ESS_LROHANDLE_API_T, 109
ESS_LROINFO_API_T, 109
ESS_MBRALT_T, 148
ESS_MBRCOUNTS_T, 699
ESS_MBRERR_T, 148
ESS_MBRINFO_T, 699
ESS_MBRUSER_T, 148
ESS_MEMBERINFO_T, 149
ESS_NEWSHAREDSERVICESNATIVEUSERINFO_T, 151
ESS_OBJDEF_T, 151
ESS_OBJINFO_T, 152
ESS_OUTERROR_T, 704
ESS_OUTLINEINFO_T, 706
ESS_OUTLINEINFOEX_T, 707
ESS_PART_CONNECT_INFO_T, 153
ESS_PART_DEFINED_T, 153
ESS_PART_INFO_T, 154
ESS_PART_REPL_T, 155
ESS_PART_T, 152
ESS_PARTDEF_areas_T, 158
ESS_PARTDEF_CONNECT_T, 156
ESS_PARTDEF_INVALID_T, 155
ESS_PARTDEF_MAP_T, 157
ESS_PARTDEF_T, 157
ESS_PARTDEF_TYPE_T, 158
ESS_PARTHDR_T, 159
ESS_PARTOTL_CHANGE_API_T, 160
ESS_PARTOTL_CHG_FILE_T, 160
ESS_PARTOTL_DIM_ATTRIB_API_T, 161
ESS_PARTOTL_DIMASSOCCHG_API_T, 162
ESS_PARTOTL_DIMCHG_API_T, 162
ESSPARTOTL_MBR_RSVD_API_T, 164
ESSPARTOTL_MBRASSOCCHG_API_T, 164
ESSPARTOTL_MBRATTR_API_T, 164
ESSPARTOTL_MBRCHG_API_T, 165
ESSPARTOTL_NAMECHG_API_T, 167
ESSPARTOTL_NAMED_GENLEV_API_T, 167
ESSPARTOTL_NAMEMAP_API_T, 167
ESSPARTOTL_OSN_RELATIVES_API_T, 168
ESSPARTOTL_QRY_FILTER_T, 170
ESSPARTOTL_QUERY_T, 169
ESSPARTOTL_READ_T, 170
ESSPARTOTL_SELECT_APPLY_T, 171
ESSPARTOTL_SELECT_CHG_T, 171
ESSPARTSLCT_T, 171
ESSPARTSLCT_VALIDATE_T, 172
ESSPERF_ALLOC_ARG_T, 172
ESSPERF_ALLOC_ERROR_T, 173
ESSPERF_ALLOC_T, 174
ESSPERF_CUSTCALC_T, 177
ESS_PREDICATE_T, 709
ESSPROCTSTATE_T, 178
ESS_RATEINFO_T, 179
ESS_REQ_STATE_T, 185
ESS_REQUESTINFO_T, 179
ESS_REQUESTINFOEX_T, 182
ESS_SEQID_T, 186
ESS_STS_T, 88
ESS_TIMERECORD_T, 186
ESS_TRANSACTION_ENTRY_T, 187
ESS_TRANSACTION_REPLAY_INP_T, 188
ESS_USERAPP_T, 189
ESS_USERAPP_EX_T, 189
ESS_USERDB_T, 190
ESS_USERDBEX_T, 191
ESS_USERINFO_T, 192
ESS_USERINFOID_T, 194
EssAddToGroup, 215
EssAddToGroupEx, 216
EssAlloc, 218
EssArchive, 219
EssArchiveBegin, 220
EssArchiveEnd, 223
EssAsyncBuildDim, 224
EssAsyncImport, 226
EssAsyncImportASO, 229
EssAutoLogin, 231
ESSBASEPATH, defined, 37
EssBeginCalc, 233
EssBeginDataload, 235
EssBeginDataloadEx, 239
EssBeginIncrementalBuildDim, 240
EssBeginReport, 242
EssBeginStreamBuildDim, 244, 331
EssBeginUpdate, 246
EssBuildDimension, 247
EssBuildDimFile, 249
EssBuildDimStart, 250
EssCalc, 251
EssCalcFile, 252
EssCancelAsyncProc, 254
EssCancelProcess, 255
EssCheckMemberName, 258
EssClearActive, 259
EssClearAliases, 260
EssClearDatabase, 261
EssCloseAsyncProc, 262
EssClrSpanRelationalPartition, 262
EssCommitDatabase, 264
EssConvertApplicationToUnicode, 266
EssCopyApplication, 267
EssCopyDatabase, 268
EssCopyFilter, 269
EssCopyObject, 271
EssCreateApplication, 272
EssCreateApplicationEx, 273
EssCreateDatabase, 274
EssCreateExtGroup, 277
EssCreateExtUser, 278
EssCreateFilter, 280
EssCreateGroup, 281
EssCreateLocalContext, 282
EssCreateLocationAlias, 283
EssCreateObject, 284
EssCreateStorageTypedApplication, 285
EssCreateStorageTypedApplicationEx, 286
EssCreateUser, 288
EssCreateVariable, 289
EssDefaultCalc, 291
EssDeleteAllSplFiles, 292
EssDeleteApplication, 292
EssDeleteDatabase, 293
EssDeleteFilter, 295
EssDeleteFromGroup, 296
EssDeleteFromGroupEx, 297
EssDeleteGroup, 299
EssDeleteGroupEx, 299
EssDeleteLocalContext, 300
EssDeleteLogFile, 301
EssDeleteObject, 302
EssDeleteSplFile, 303
EssDeleteUser, 304
EssDeleteUserEx, 305
EssDeleteVariable, 306
EssDisplayAlias, 308
EssDisplayTriggers, 310
EssDTAPIClose, 310
EssDTAPIConnect, 311
EssDTAPIExecuteReport, 312
EssDTAPIExit, 312
EssDTAPIGetColumns, 313
EssDTAPIGetData, 314
EssDTAPIGetError, 314
EssDTAPIGetInfo, 315
EssDTAPIGetReports, 316
EssDTAPIInit, 316
EssDTAPISetConnection, 317
EssDTAPISetInfo, 318
EssDTClose, 318
EssDTExit, 319
EssDTGetData, 320
EssDTGetHeader, 320
EssDTGetHeaderInfo, 321
EssDTInit, 321
EssDTListReports, 322
EssDTOpen, 323
EssDumpPerfStats, 323
EssEndCalc, 325
EssEndDataload, 326
EssEndIncrementalBuildDim, 327
EssEndReport, 329
EssEndUpdate, 332
ESSERROR.H, 86, 88, 94
EssExport, 333
EssFixIBH, 334
EssFree, 335
EssFreeMbrErr, 336
ESSG_CONNECTINFO_T, 1006
ESSG_DATA_T, 1006
ESSG_DRILLDATA_T, 1008
ESSG_DTDATA_T, 1009
ESSG_DTHEADER_T, 1009
ESSG_DTINFO_T, 1010
ESSG_DREPORT_T, 1010
ESSG_INIT_T, 1011
ESSG_LRODESC_T, 1011
ESSG_LROINFO_T, 1012
ESSG_PDATA_T, 997
ESSG_PINIT_T, 997
ESSG_PPDATA_T, 997
ESSG_PRANGE_T, 997
ESSG_RANGE_T, 1012
EssGBeginConditionalRetrieve, 1015
EssGBeginConditionalZoomIn, 1017
EssGBeginCreateLRO, 1020
EssGBeginDataPoint, 1021
EssGBeginDeleteLROs, 1023
EssGBeginDrillAcross, 1023
EssGBeginDrillOrLink, 1024
EssGBeginKeepOnly, 1025
EssGBeginLock, 1027
EssGBeginPivot, 1028
EssGBeginRemoveOnly, 1030
EssGBeginReport, 1033
EssGBeginReportFile, 1034
EssGBeginRetrieve, 1037
EssGBeginSamplingZoomIn, 1038
EssGBeginUpdate, 1040
EssGBeginZoomIn, 1041
EssGBeginZoomOut, 1044
EssGCancelOperation, 1046
EssGCell, 1047
EssGConnect, 1052
EssGConnectEx, 1053
EssGDeleteLRO, 1054
EssGDestroyGrid, 1055
EssGDisconnect, 1056
EssGDTBeginDrillThrough, 1057
EssGDTConnect, 1058
EssGDTEndDrillThrough, 1058
EssGDTExecuteReport, 1059
EssGDTGetData, 1060
EssGDTGetHeader, 1060
EssGDTGetInfo, 1061
EssGDTGetReportData, 1062
EssGDTListReports, 1062
EssGDTReportCount, 1063
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>EssGDTRequestDrillThrough</td>
<td>1064</td>
</tr>
<tr>
<td>EssGDTSetInfo</td>
<td>1065</td>
</tr>
<tr>
<td>EssGEndOperation</td>
<td>1065</td>
</tr>
<tr>
<td>EssGetActive</td>
<td>339</td>
</tr>
<tr>
<td>EssGetAlias</td>
<td>341</td>
</tr>
<tr>
<td>EssGetAPIVersion</td>
<td>342</td>
</tr>
<tr>
<td>EssGetApplicationAccess</td>
<td>342</td>
</tr>
<tr>
<td>EssGetApplicationAccessEx</td>
<td>344</td>
</tr>
<tr>
<td>EssGetApplicationInfo</td>
<td>349</td>
</tr>
<tr>
<td>EssGetApplicationInfoEx</td>
<td>350</td>
</tr>
<tr>
<td>EssGetApplicationState</td>
<td>352</td>
</tr>
<tr>
<td>EssGetAssociatedAttributesInfo</td>
<td>353</td>
</tr>
<tr>
<td>EssGetAsyncProcLog</td>
<td>355</td>
</tr>
<tr>
<td>EssGetAsyncProcState</td>
<td>356</td>
</tr>
<tr>
<td>EssGetAttributeInfo</td>
<td>357</td>
</tr>
<tr>
<td>EssGetAttributeSpecifications</td>
<td>359</td>
</tr>
<tr>
<td>EssGetCalcList</td>
<td>362</td>
</tr>
<tr>
<td>EssGetCurrencyRateInfo</td>
<td>365</td>
</tr>
<tr>
<td>EssGetDatabaseAccess</td>
<td>367</td>
</tr>
<tr>
<td>EssGetDatabaseAccessEx</td>
<td>369</td>
</tr>
<tr>
<td>EssGetDatabaseInfo</td>
<td>374</td>
</tr>
<tr>
<td>EssGetDatabaseInfoEx</td>
<td>375</td>
</tr>
<tr>
<td>EssGetDatabaseNote</td>
<td>377</td>
</tr>
<tr>
<td>EssGetDatabaseState</td>
<td>377</td>
</tr>
<tr>
<td>EssGetDatabaseStats</td>
<td>379</td>
</tr>
<tr>
<td>EssGetDefaultCalc</td>
<td>380</td>
</tr>
<tr>
<td>EssGetDimensionInfo</td>
<td>381</td>
</tr>
<tr>
<td>EssGetFilter</td>
<td>385</td>
</tr>
<tr>
<td>EssGetFilterList</td>
<td>387</td>
</tr>
<tr>
<td>EssGetFilterRow</td>
<td>388</td>
</tr>
<tr>
<td>EssGetGlobalState</td>
<td>389</td>
</tr>
<tr>
<td>EssGetGroup</td>
<td>390</td>
</tr>
<tr>
<td>EssGetGroupInfo</td>
<td>391</td>
</tr>
<tr>
<td>EssGetGroupList</td>
<td>395</td>
</tr>
<tr>
<td>EssGetGroupListEx</td>
<td>396</td>
</tr>
<tr>
<td>EssGetIBH</td>
<td>398</td>
</tr>
<tr>
<td>EssGetLocalPath</td>
<td>399</td>
</tr>
<tr>
<td>EssGetLogFile</td>
<td>401</td>
</tr>
<tr>
<td>EssGetMemberCalc</td>
<td>402</td>
</tr>
<tr>
<td>EssGetMemberInfo</td>
<td>404</td>
</tr>
<tr>
<td>EssGetObject</td>
<td>405</td>
</tr>
<tr>
<td>EssGetObjectCalc</td>
<td>402</td>
</tr>
<tr>
<td>EssGetMemberInfoEx</td>
<td>404</td>
</tr>
<tr>
<td>EssGetObjectCalc</td>
<td>405</td>
</tr>
<tr>
<td>EssGetObjectInfo</td>
<td>407</td>
</tr>
<tr>
<td>EssGetProcessState</td>
<td>407</td>
</tr>
<tr>
<td>EssGetServerLocaleString</td>
<td>409</td>
</tr>
<tr>
<td>EssGetServerMode</td>
<td>410</td>
</tr>
<tr>
<td>EssGetSpoolFile</td>
<td>410</td>
</tr>
<tr>
<td>EssGetStatBufSize</td>
<td>412</td>
</tr>
<tr>
<td>EssGetString</td>
<td>413</td>
</tr>
<tr>
<td>EssGetUser</td>
<td>414</td>
</tr>
<tr>
<td>EssGetUserEx</td>
<td>415</td>
</tr>
<tr>
<td>EssGetUserInfoEx</td>
<td>417</td>
</tr>
<tr>
<td>EssGetUserName</td>
<td>421</td>
</tr>
<tr>
<td>EssGetVariable</td>
<td>421</td>
</tr>
<tr>
<td>EssGetVersion</td>
<td>424</td>
</tr>
<tr>
<td>EssGFreeCellLinkResults</td>
<td>1066</td>
</tr>
<tr>
<td>EssGFreeMemberInfo</td>
<td>1067</td>
</tr>
<tr>
<td>EssGFreeRows</td>
<td>1070</td>
</tr>
<tr>
<td>EssGGetAPIContext</td>
<td>1071</td>
</tr>
<tr>
<td>EssGGetAPIInstance</td>
<td>1072</td>
</tr>
<tr>
<td>EssGGetCellLinkResults</td>
<td>1073</td>
</tr>
<tr>
<td>EssGGetDataSetResults</td>
<td>1074</td>
</tr>
<tr>
<td>EssGGetFormattedValue</td>
<td>1075</td>
</tr>
<tr>
<td>EssGGetGridOption</td>
<td>1079</td>
</tr>
<tr>
<td>EssGGetGridPerspective</td>
<td>1080</td>
</tr>
<tr>
<td>EssGGetLinkedPartitionDesc</td>
<td>1081</td>
</tr>
<tr>
<td>EssGGetLRO</td>
<td>1082</td>
</tr>
<tr>
<td>EssGGetLRODesc</td>
<td>1083</td>
</tr>
<tr>
<td>EssGGetMemberInfo</td>
<td>1084</td>
</tr>
<tr>
<td>EssGGetResults</td>
<td>1085</td>
</tr>
<tr>
<td>EssGGetRows</td>
<td>1086</td>
</tr>
<tr>
<td>EssGGetSmartlistforCell</td>
<td>1087</td>
</tr>
<tr>
<td>EssGInit</td>
<td>1088</td>
</tr>
<tr>
<td>EssGLoginSetPass</td>
<td>1089</td>
</tr>
<tr>
<td>EssGNewGrid</td>
<td>1090</td>
</tr>
<tr>
<td>EssGPerformOperation</td>
<td>1092</td>
</tr>
<tr>
<td>EssGSendRows</td>
<td>1092</td>
</tr>
<tr>
<td>EssGSetGridOption</td>
<td>1093</td>
</tr>
<tr>
<td>EssGSetGridPerspective</td>
<td>1095</td>
</tr>
<tr>
<td>EssGSetPath</td>
<td>1096</td>
</tr>
<tr>
<td>EssGTerm</td>
<td>1097</td>
</tr>
<tr>
<td>EssGUnlock</td>
<td>1098</td>
</tr>
<tr>
<td>EssGUpdateLRO</td>
<td>1099</td>
</tr>
<tr>
<td>EssGVersion</td>
<td>1099</td>
</tr>
<tr>
<td>ESSHELP.H</td>
<td>41</td>
</tr>
<tr>
<td>EssImport</td>
<td>425</td>
</tr>
<tr>
<td>EssIncrementalBuildDim</td>
<td>429</td>
</tr>
<tr>
<td>EssInit</td>
<td>430</td>
</tr>
<tr>
<td>EssInit function</td>
<td>81, 84, 89, 90</td>
</tr>
<tr>
<td>EssKillRequest</td>
<td>432</td>
</tr>
<tr>
<td>EssKillRequestEx</td>
<td>434</td>
</tr>
<tr>
<td>EssListAliases</td>
<td>436</td>
</tr>
<tr>
<td>EssListApplications</td>
<td>437</td>
</tr>
<tr>
<td>EssListCalcFunctions</td>
<td>438</td>
</tr>
<tr>
<td>EssListConnections</td>
<td>442</td>
</tr>
</tbody>
</table>

Index 1739
<table>
<thead>
<tr>
<th>Function Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>EssListConnectionsEx</td>
<td>443</td>
</tr>
<tr>
<td>EssListCurrencyDatabases</td>
<td>447</td>
</tr>
<tr>
<td>EssListDatabases</td>
<td>449</td>
</tr>
<tr>
<td>EssListFilters</td>
<td>457</td>
</tr>
<tr>
<td>EssListFilterUsers</td>
<td>458</td>
</tr>
<tr>
<td>EssListGroups</td>
<td>459</td>
</tr>
<tr>
<td>EssListGroupsInfoEx</td>
<td>460</td>
</tr>
<tr>
<td>EssListLocks</td>
<td>464</td>
</tr>
<tr>
<td>EssListLocksEx</td>
<td>466</td>
</tr>
<tr>
<td>EssListLogins</td>
<td>467</td>
</tr>
<tr>
<td>EssListLoginsEx</td>
<td>468</td>
</tr>
<tr>
<td>EssListObjects</td>
<td>469</td>
</tr>
<tr>
<td>EssListRequests</td>
<td>471</td>
</tr>
<tr>
<td>EssListRequestsEx</td>
<td>473</td>
</tr>
<tr>
<td>EssListSpoolFiles</td>
<td>476</td>
</tr>
<tr>
<td>EssListUsers</td>
<td>481</td>
</tr>
<tr>
<td>EssListUsersEx</td>
<td>483</td>
</tr>
<tr>
<td>EssListUsersInfoEx</td>
<td>485</td>
</tr>
<tr>
<td>EssListVariables</td>
<td>489</td>
</tr>
<tr>
<td>EssLoadAlias</td>
<td>492</td>
</tr>
<tr>
<td>EssLoadApplication</td>
<td>493</td>
</tr>
<tr>
<td>EssLoadDatabase</td>
<td>502</td>
</tr>
<tr>
<td>EssLocateIBH</td>
<td>502</td>
</tr>
<tr>
<td>EssLockObject</td>
<td>503</td>
</tr>
<tr>
<td>EssLogin</td>
<td>504</td>
</tr>
<tr>
<td>EssLoginEx</td>
<td>508</td>
</tr>
<tr>
<td>EssLoginSetPassword</td>
<td>510</td>
</tr>
<tr>
<td>EssLogout</td>
<td>512</td>
</tr>
<tr>
<td>EssLogoutUser</td>
<td>513</td>
</tr>
<tr>
<td>EssLogSize</td>
<td>514</td>
</tr>
<tr>
<td>EssLROAddObject</td>
<td>515</td>
</tr>
<tr>
<td>EssLRODeleteCellObjects</td>
<td>517</td>
</tr>
<tr>
<td>EssLRODeleteObject</td>
<td>519</td>
</tr>
<tr>
<td>EssLROGetCatalog</td>
<td>520</td>
</tr>
<tr>
<td>EssLROGetObject</td>
<td>525</td>
</tr>
<tr>
<td>EssLROListObjects</td>
<td>526</td>
</tr>
<tr>
<td>EssLROPurgeObjects</td>
<td>527</td>
</tr>
<tr>
<td>EssLROUpdateObject</td>
<td>529</td>
</tr>
<tr>
<td>EssMdxTrig</td>
<td>531</td>
</tr>
<tr>
<td>EssOtlAddDimension</td>
<td>721</td>
</tr>
<tr>
<td>EssOtlAddMember</td>
<td>726</td>
</tr>
<tr>
<td>EssOtlAssociateAttributeDimension</td>
<td>729</td>
</tr>
<tr>
<td>EssOtlAssociateAttributeMember</td>
<td>731</td>
</tr>
<tr>
<td>EssOtlClearAliasTable</td>
<td>733</td>
</tr>
<tr>
<td>EssOtlClearAliasTableLanguages</td>
<td>734</td>
</tr>
<tr>
<td>EssOtlCloseOutline</td>
<td>736</td>
</tr>
<tr>
<td>EssOtlCompactOutline</td>
<td>737</td>
</tr>
<tr>
<td>EssOtlCopyAliasTable</td>
<td>739</td>
</tr>
<tr>
<td>EssOtlCreateAliasTable</td>
<td>741</td>
</tr>
<tr>
<td>EssOtlDeleteAliasTable</td>
<td>744</td>
</tr>
<tr>
<td>EssOtlDeleteDimension</td>
<td>745</td>
</tr>
<tr>
<td>EssOtlDeleteDTSMemberAlias</td>
<td>747</td>
</tr>
<tr>
<td>EssOtlDeleteGenName</td>
<td>749</td>
</tr>
<tr>
<td>EssOtlDeleteLevelName</td>
<td>750</td>
</tr>
<tr>
<td>EssOtlDeleteMember</td>
<td>753</td>
</tr>
<tr>
<td>EssOtlDeleteMemberAlias</td>
<td>754</td>
</tr>
<tr>
<td>EssOtlDeleteMemberFormula</td>
<td>755</td>
</tr>
<tr>
<td>EssOtlDeleteUserAttribute</td>
<td>758</td>
</tr>
<tr>
<td>EssOtlDisassociateAttributeDimension</td>
<td>763</td>
</tr>
<tr>
<td>EssOtlDisassociateAttributeMember</td>
<td>764</td>
</tr>
<tr>
<td>EssOtlEnableDTSMember</td>
<td>766</td>
</tr>
<tr>
<td>EssOtlFindAlias</td>
<td>768</td>
</tr>
<tr>
<td>EssOtlFindMember</td>
<td>771</td>
</tr>
<tr>
<td>EssOtlFreeMembers</td>
<td>774</td>
</tr>
<tr>
<td>EssOtlFreeStructure</td>
<td>778</td>
</tr>
<tr>
<td>EssOtlGenerateCurrencyOutline</td>
<td>779</td>
</tr>
<tr>
<td>EssOtlGetAliasTableLanguages</td>
<td>783</td>
</tr>
<tr>
<td>EssOtlGetASOCompressionDimension</td>
<td>785</td>
</tr>
<tr>
<td>EssOtlGetAssociatedAttributes</td>
<td>786</td>
</tr>
<tr>
<td>EssOtlGetAttributeAssocLevel</td>
<td>787</td>
</tr>
<tr>
<td>EssOtlGetAttributeInfo</td>
<td>789</td>
</tr>
<tr>
<td>EssOtlGetAttributeSpecifications</td>
<td>791</td>
</tr>
<tr>
<td>EssOtlGetChild</td>
<td>795</td>
</tr>
<tr>
<td>EssOtlGetCountOfDupMemberNameInDim</td>
<td>796</td>
</tr>
<tr>
<td>EssOtlGetDimensionNameUniqueness</td>
<td>799</td>
</tr>
<tr>
<td>EssOtlGetDimensionSolveOrder</td>
<td>800</td>
</tr>
<tr>
<td>EssOtlGetDimensionUserAttributes</td>
<td>801</td>
</tr>
<tr>
<td>EssOtlGetDTSMemberAlias</td>
<td>803</td>
</tr>
<tr>
<td>EssOtlGetEnabledDTSMembers</td>
<td>805</td>
</tr>
<tr>
<td>EssOtlGetFirstMember</td>
<td>806</td>
</tr>
<tr>
<td>EssOtlGetGenName</td>
<td>807</td>
</tr>
<tr>
<td>EssOtlGetGenNameEx</td>
<td>809</td>
</tr>
<tr>
<td>EssOtlGetGenNames</td>
<td>811</td>
</tr>
<tr>
<td>EssOtlGetLevelName</td>
<td>814</td>
</tr>
<tr>
<td>EssOtlGetLevelNameEx</td>
<td>815</td>
</tr>
<tr>
<td>EssOtlGetLevelNames</td>
<td>817</td>
</tr>
<tr>
<td>EssOtlGetLinkedAttributeAttachLevel</td>
<td>819</td>
</tr>
<tr>
<td>EssOtlGetMemberAlias</td>
<td>821</td>
</tr>
<tr>
<td>EssOtlGetMemberCommentEx</td>
<td>822</td>
</tr>
<tr>
<td>EssOtlGetMemberField</td>
<td>823</td>
</tr>
<tr>
<td>EssOtlGetMemberFormula</td>
<td>826</td>
</tr>
<tr>
<td>EssOtlGetMemberInfo</td>
<td>827</td>
</tr>
<tr>
<td>EssOtlGetMemberInfoArray</td>
<td>828</td>
</tr>
<tr>
<td>EssOtlGetMemberLastFormula</td>
<td>830</td>
</tr>
</tbody>
</table>

1740 Index
EssOtlGetMemberSolveOrder, 833
EssOtlGetMemberUniqueName, 836
EssOtlGetNextSharedMember, 838
EssOtlGetNextSibling, 840
EssOtlGetOriginalMember, 845
EssOtlGetOutlineInfo, 847
EssOtlGetParent, 848
EssOtlGetPrevSibling, 849
EssOtlGetUpdateTime, 855
EssOtlGetUserAttributes, 856
EssOtlIsMemberNameNonUnique, 860
EssOtlIsMemberNameUniqueWithinDim, 862
EssOtlIsMemberNameUniqueWithinDimAtGenLevel 1, 863
EssOtlMoveMember, 867
EssOtlNewOutline, 869
EssOtlOpenOutline, 870
EssOtlOpenOutlineEx, 872
EssOtlOpenOutlineQuery, 873
EssOtlQueryAttributes, 877
EssOtlQueryAttributesEx, 879
EssOtlQueryGenerationInfo, 879
EssOtlQueryMembers, 884
EssOtlQueryMembersByName, 891
EssOtlQueryMembersEx, 894
EssOtlRenameAliasTable, 901
EssOtlRenameMember, 903
EssOtlRestructure, 904
EssOtlSetAliasTableLanguage, 908
EssOtlSetAttributeSpecifications, 911
EssOtlSetDimensionNameUniqueness, 917
EssOtlSetDTSMemberAlias, 920
EssOtlSetGenName, 923
EssOtlSetGenNameEx, 924
EssOtlSetLevelName, 927
EssOtlSetLevelNameEx, 929
EssOtlSetMemberAlias, 930
EssOtlSetMemberCommentEx, 932
EssOtlSetMemberFormula, 933
EssOtlSetMemberInfo, 934
EssOtlSetOutlineInfo, 944
EssOtlSetOutlineInfoEx, 945
EssOtlSetUserAttribute, 949
EssOtlSortChildren, 950
EssOtlVerifyFormula, 966
EssOtlVerifyOutline, 967
EssOtlVerifyOutlineEx, 969
EssOtlWriteOutline, 976
EssOtlWriteOutlineEx, 978
EssPartialDataClear, 533
EssPartitionApplyOtlChangeFile, 534
EssPartitionApplyOtlChangeReCS, 537
EssPartitionCloseDefFile, 539
EssPartitionFreeDefCtx, 540
EssPartitionFreeOtlChanges, 541
EssPartitionGetAreaCellCount, 542
EssPartitionGetAreaLevelCellCount, 543
EssPartitionGetList, 544
EssPartitionGetOtlChanges, 546
EssPartitionGetRepCells, 548
EssPartitionNewDefFile, 549
EssPartitionOpenDefFile, 551
EssPartitionPurgeOtlChangeFile, 552
EssPartitionPutRepCells, 554
EssPartitionReadDefFile, 555
EssPartitionReadOtlChangeFile, 556
EssPartitionReplaceDefFile, 558
EssPartitionResetOtlChangeTime, 559
EssPartitionValidateDefinition, 560
EssPartitionValidateLocal, 562
EssPartitionWriteDefFile, 564
EssPerformAllocationASO, 565
EssPerformCustomCalcASO, 567
EssPutObject, 569
EssQueryDatabaseMembers, 570
EssRealloc, 573
EssRemoveAlias, 574
EssRemoveLocks, 575
EssRenameApplication, 580
EssRenameDatabase, 581
EssRenameFilter, 582
EssRenameGroup, 583
EssRenameObject, 584
EssRenameUser, 585
EssReport, 586
EssReportFile, 588
EssResetDatabase, 591
EssResetUser, 593
EssRestore, 594
EssSendString, 596
EssSetActive, 598
EssSetAlias, 599
EssSetApplicationAccess, 599
EssSetApplicationAccessEx, 600
EssSetApplicationState, 605
EssSetCalcList, 606
EssSetCalcListEx, 608
EssSetDatabaseAccess, 610
EssSetDatabaseAccessEx, 611
EssSetDatabaseNote, 616
EssSetDatabaseState, 617
EssSetDefaultCalc, 618
EssSetDefaultCalcFile, 619
EssSetFilter, 621
EssSetFilterList, 623
EssSetFilterListEx, 624
EssSetFilterRow, 626
EssSetGlobalState, 627
EssSetGroup, 628
EssSetGroupList, 629
EssSetGroupListEx, 630
EssSetPassword, 632
EssSetPath, 633
EssSetServerMode, 634
EssSetSpanRelationalPartition, 635
EssSetUser, 639
EssSetUserEx, 640
EssShutdownServer, 644
EssTerm, 645
EssUnloadApplication, 646
EssUnloadDatabase, 647
EssUnlockObject, 647
EssUpdate, 649
EssUpdateBAKFile, 650
EssUpdateEx, 652
EssUpdateFile, 653
EssUpdateFileASOEx, 657
EssUpdateFileEx, 659
EssUpdateFileUtf8Ex, 663
EssUpdateUtf8Ex, 665
EssValidateDB, 666
EssValidateHCtx, 667
EssVerifyFilter, 670
EssVerifyFilterRow, 671
EssVerifyFormula, 672
EssVerifyRulesFile, 672
EssWriteToLogFile, 674
Example of Traversing an Outline, 981
examples, C Grid API, 1101
Execute method, 1625

F
file functions
C Main API, 204
VB Main API, 1203
file objects
C, 82
VB, 1122
files
where stored, 37
files to stored in Grid API programs, 993
files to link with Grid API programs, 42
files to send with Grid API programs, 42
files you need to ship, 41
flattened rowsets, 1657
function call sequence, 682
function categories
VB Main API, 1199
VB Outline API, 1465

G
generation name functions
C Outline API, 715
VB Outline API, 1467
generation options
C constants, 695
VB constants, 1457
Grid API
architecture, 992
definition, 991
libraries, 42
use with main API, 994
grid coordinates, 995
group administration functions
C Main API, 205
VB Main API, 1203
group identity functions
C Main API, 212

H
handles
C, 81
VB, 1120
header files, 32
help file
customizing, 40
HP-UX programs
libraries provided with API, 42
limits, 1727
  application name, 1727
database name, 1728
drill-through URLs, 1729
  Essbase Server (host) name, 1727
  filter name, 1728
group name, 1728
  names, 1727
  object name, 1728
password, 1728
user name, 1728
linked reporting objects
  C constants, 107
Linux programs
  building, 33
list option
  C constants, 105
ListLogins, 467
local contexts
  C, 82, 83
  VB, 1122
location aliases functions
  C Main API, 207
  VB Main API, 1205
logging into an Essbase Server
  C, 90
  VB, 1128
logging out from the Essbase Server
  C, 93
  VB, 1131
login functions
  C Main API, 206
  VB Main API, 1204
LRO functions
  C Main API, 207
  VB Main API, 1204
make file samples, 29, 34, 35, 36
managing memory, 993
maximum string lengths
  C constants, 105
  VB constants, 1153
member administration functions
  C Outline API, 715
  VB Outline API, 1467
member alias functions
C Outline API, 716
VB Outline API, 1468
member formula functions
C Outline API, 716
VB Outline API, 1468
member traversal functions
C Outline API, 717
VB Outline API, 1468
member types
C constants, 694
memory allocation, 681
VB Outline API, 1447
memory allocation functions
C Main API, 207
memory in C programs, 84
memory management, 993
message database
customizing, 39
message handling
C, 85
VB, 1124
miscellaneous functions
C Main API, 208
VB Main API, 1205
miscellaneous types
C, 99
multiple context handles
C, 82
VB, 1121

N
name limits, 1727
network libraries
C, 81

O
object functions
C Main API, 208
VB Main API, 1206
objects
file
C, 82
VB, 1122
operating systems supported, 30
operation task sequence, 682

P
packaging programs for distribution, 43
partition functions
C Main API, 208
partitioning structures
C, 1157
VB, 1157
passing handles
C, 81
VB, 1121
Performance Statistics
Buffer Size, 412
Dump, 323
Reset, 591
performance stats functions
C Main API, 210
platforms
supported OSs, 30
pointer types
C, 98
VB, 1163
problems and solutions, 93

Q
query options
C constants, 695
query types
C constants, 695
VB constants, 1457
recalculating the database
C, 92
VB, 1130
redistributing files, 41
redistributing programs, 43
Relational Partition
  clear span, 262
  set span, 635
report specification strings
  C, 91
  VB, 1129
reporting functions
  C Main API, 210
reports.vbp sample program, 1703
Request Management, 471
request type
  C constants, 106
restructure values
  C constants, 696
  VB constants, 1454
retrieving data
  C, 91
  VB, 1129
return codes
  handling
    C, 88
    VB, 1126
return codes for the API, 88
Run-time Client, 41
run-time environment
  customizing, 38
run-time files, 41
run-time libraries provided with API, 42

sample programs
  C API, 1665, 1672, 1681
  VB API, 1692, 1695, 1703
scenarios
  troubleshooting
    C, 93
    VB, 1132
security filter functions
  C Main API, 211
  VB Main API, 1207
security requirements, 681
Performance Dump, 323
Performance Reset, 591
steps to package programs, 43
string lengths
  maximum
    C constants, 105
    VB constants, 1153
structures
  VB partitioning, 1157
substitution variable functions
  C Main API, 211
  VB Main API, 1208
supported compilers, 29
supported platforms for Grid API, 992
symbolic constant definitions
  C, 692
  VB, 1454
symbolic links, 33

T
task sequence
  VB Outline API, 1448
TCP/IP networking optimization, 44
terminating the API
  C, 93
threads
  relationship to handles
    C, 81
    VB, 1121
time balance skip values
  C constants, 693
time balance values
  C constants, 693
  VB constants, 1455
troubleshooting scenarios
  C, 93
  VB, 1132
typical API task sequence
  C, 89
  VB, 1127
typical function call sequence
  VB Outline API, 1448

U
understanding Visual Basic declarations, 1119
Unicode issues, 73
Unicode mode
  converting applications to, 266
  creating applications, 273
  opening outlines, 872
  retrieving mode of server, 410
  setting mode of server, 634
  writing outlines, 978
Unicode mode functions
  C Main API, 215
  C Outline API, 718
UNIX programs
  building, 33
update specification strings
  C, 92
  VB, 1130
updating data
  C, 91
  VB, 1129
updating functions
  C Main API, 210
user administration functions
  C Main API, 212
  VB Main API, 1208
user attribute functions
  VB Outline API, 1470
user identity functions
  C Main API, 212
user-defined attributes functions
  C Outline API, 719

V
varying attributes functions
  C Outline API, 719
verifying outlines, 680
versioning in the Grid API, 993
Visual Basic API conventions, 1119
Visual Basic constant definitions, 1153
Visual Basic declarations
  understanding, 1119
Visual Basic functions for Excel, 1120
Visual Basic language types, 1157

X
XML for Analysis, 1621
XMLA methods, 1621
XMLA reference, 1621
XMLA rowsets, 1627