Oracle® Essbase

API Reference

RELEASE 11.1.2.1

ORACLE

ENTERPRISE PERFORMANCE
MANAGEMENT SYSTEM

Essbase API Reference, 11.1.2.1

Copyright © 1996, 2011, Oracle and/or its affiliates. All rights reserved.

Authors: EPM Information Development Team

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited. The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this software or related documentation is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS:
Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, duplication, disclosure, modification, and adaptation shall be subject to the restrictions and license terms set forth in the applicable Government contract, and, to the extent applicable by the terms of the Government contract, the additional rights set forth in FAR 52.227-19, Commercial Computer Software License (December 2007). Oracle USA, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

This software is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications which may create a risk of personal injury. If you use this software in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure the safe use of this software. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software in dangerous applications.

Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

This software and documentation may provide access to or information on content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

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

Basic Requirements 48
The Nested Program Model 48
Using Function Return Codes 49
Calling API Functions 50
Initializing the API 51
Logging On to a Server 51
Connecting to a Database 53
Logging Out 54
Terminating the API 54
Assembling a Program 55
Building Dimensions 56
Editing the Outline 56
Loading Data 58
Reporting 58
Updating Data 65
Calculating Data 66
Using Security 67
Maintaining Applications and Databases 68
Handling Messages 69
Managing Memory 71

Chapter 5. Unicode Issues in Essbase API Programs 73
General Programming Considerations 73
Defining Unicode Mode Client Programs 73
Non-Unicode Clients 74
Unicode-enabled Clients in Non-Unicode Mode 74
Unicode-enabled Clients in Unicode Mode 75
Specifying Unicode Mode 75
Specifying the Byte Order Encoding 75
Unicode Mode and Essbase Server 76
Unicode Outlines 77
Grid API 77

Part II. C Main API 79

Chapter 6. Using the C Main API 81
C Main API Instance Handles 81
C Main API Context Handles 82
C Main API File Objects 82
Using Memory in C Programs 84

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

ESS_LOG_DATALOAD_T 147
ESS_MBRALT_T 148
ESS_MBRERR_T 148
ESS_MBRUSER_T 148
ESS_MEMBERINFO_T 149
ESS_NEWSHAREDSERVICESNATIVEUSERINFO_T 151
ESS_OBJDEF_T 151
ESS_OBJINFO_T 152
ESS_PART_T 152
ESS_PART_CONNECT_INFO_T 153
ESS_PART_DEFINED_T 153
ESS_PART_INFO_T 154
ESS_PART_REPL_T 155
ESS_PARTDEF_INVALID_T 155
ESS_PARTDEF_CONNECT_T 156
ESS_PARTDEF_MAP_T 157
ESS_PARTDEF_T 157
ESS_PARTDEF_AREAS_T 158
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
ESS_PARTOTL_MBR_RSRVD_API_T 164
ESS_PARTOTL_MBRASSOCCHG_API_T 164
ESS_PARTOTL_MBRATTR_API_T 164
ESS_PARTOTL_MBRCHG_API_T 165
ESS_PARTOTL_NAMECHG_API_T 167
ESS_PARTOTL_NAMED_GENLEV_API_T 167
ESS_PARTOTL_NAMEMAP_API_T 167
ESS_PARTOTL_OSN_RELATIVES_API_T 168
ESS_PARTOTL_QUERY_T 169
ESS_PARTOTL_QRY_FILTER_T 170
ESS_PARTOTL_READ_T 170
ESS_PARTOTL_SELECT_APPLY_T 171
ESS_PARTOTL_SELECT_CHG_T 171
ESS_PARTSLCT_T 171

ESS_PARTSLCT_VALIDATE_T 172
ESS_PERF_ALLOC_ARG_T 172
ESS_PERF_ALLOC_ERROR_T 173
ESS_PERF_ALLOC_T 174
ESS_PERF_CUSTCALC_T 177
ESS_PROCSTATE_T 178
ESS_RATEINFO_T 179
ESS_REQUESTINFO_T 179
ESS_REQUESTINFOEX_T 182
ESS_REQ_STATE_T 185
ESS_SECURITY_MODE_T 186
ESS_SEQID_T 186
ESS_TIMERECORD_T 186
ESS_TRANSACTION_ENTRY_T 187
ESS_TRANSACTION_REPLAY_INP_T 188
ESS_TRANSACTION_REQSPECIFIC_T 188
ESS_USERAPP_T, ESS_GROUPAPP_T 189
ESS_USERAPPEX_T, ESS_GROUPAPPEX_T 189
ESS_USERDB_T, ESS_GROUPDB_T 190
ESS_USERDBEX_T, ESS_GROUPDBEX_T 191
ESS_USERINFO_T, ESS_GROUPINFO_T 192
ESS_USERINFOID_T, ESS_GROUPINFOID_T 194
ESS_USERINFOEX_T 195
ESS_VARIABLE_T 197

Chapter 8. C Main API Functions 199
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

C Main API Miscellaneous Functions . . . 208
C Main API Object Functions . . . 208
C Main API Partition Functions . . . 208
C Main API Performance Stats Functions . . . 210
C Main API Reporting, Updating, and Calculation Functions . . . 210
C Main API Security Filter Functions . . . 211
C Main API Substitution Variables Functions . . . 211
C Main API User Administration Functions . . . 212
C Main API User and Group Identity Functions . . . 212
C Main API Shared Services Functions . . . 214
C Main API Unicode Mode Functions . . . 215
C Main API Function Reference . . . 215

Part III. C Outline API . . . 677

Chapter 9. Using the C Outline API . . . 679
C Outline API Overview . . . 679
C Outline API Error Handling . . . 679
C Outline API Server Outline Queries . . . 680
C Outline API Outline Verification . . . 680
C Outline API Memory Allocation . . . 681
C Outline API Security Requirements . . . 681
C Outline API Function Call Sequence . . . 682
Typical C Outline API Task Sequence . . . 682

Chapter 10. C Outline API Declarations . . . 685
C Outline API Error Return Values . . . 685
C Outline API DTS Member Structures . . . 691
C Outline API Symbolic Constant Definitions . . . 692
Account Member Currency Conversion Category Values . . . 693
Account Member Time Balance Skip Values . . . 693
Account Member Time Balance Values . . . 693
Dimension Categories . . . 693
Dimension Categories (Tags) . . . 694
Member Types . . . 694
Generation and Level Options . . . 695
Query Options . . . 695
Query Types . . . 695
Restructure Values . . . 696
Share Constants . . . 696

Sorting Options . . . 697
ESS_ATTRIBUTEQUERY_T . . . 697
ESS_GENLEVELNAME_T . . . 698
ESS_GENLEVELNAMEEX_T . . . 699
ESS_MBRCOUNTS_T . . . 699
ESS_MBRINFO_T . . . 699
ESS_OTLQUERYERRORLIST_T . . . 703
ESS_OUTERROR_T . . . 704
ESS_OUTLINEINFO_T . . . 706
ESS_OUTLINEINFOEX_T . . . 707
ESS_PERSPECTIVE_T . . . 708
ESS_PREDICATE_T . . . 709
ESS_SVROTLINFO_T . . . 710
ESS_VALIDITYSET_T . . . 710

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 989

Chapter 13. Using the C Grid API 991
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 997
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 1015
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
EssGCreateMemberwKeyStr 1049
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
EssGDTRequestDrillThrough 1064
EssGDTSetInfo 1065
EssGEndOperation 1065
EssGFreeCellLinkResults 1066
EssGFreeMemberInfo 1067
EssGFreeMemberwKeyStr 1067
EssGFreeRows 1070
EssGGetAPIContext 1071
EssGGetAPIInstance 1072

EssGGetCellLinkResults 1073
EssGGetDataPointResults 1074
EssGGetFormattedValue 1075
EssGGetFromMemberwKey 1076
EssGGetGridOption 1079
EssGGetGridPerspective 1080
EssGGetIsCellDrillable 1080
EssGGetLinkedPartitionDesc 1081
EssGGetLRO 1082
EssGGetLRODesc 1083
EssGGetMemberInfo 1084
EssGGetResults 1085
EssGGetRows 1086
EssGGetSmartlistforCell 1087
EssGInit 1088
EssGLoginSetPass 1089
EssGNewGrid 1090
EssGPerformOperation 1092
EssGSendRows 1092
EssGSetGridOption 1093
EssGSetGridPerspective 1095
EssGSetPath 1096
EssGTerm 1097
EssGUnlock 1098
EssGUpdateLRO 1099
EssGVersion 1099

Chapter 16. C Grid API Examples 1101
C Grid API Example 1101
C Grid API Drill-Through Example 1105
ESSG_OP_MEMBERANDUNIQUENAME Example 1107
ESSG_DT_MEMBERwKEY Example 1109
BuildTable Example Function 1111
DisplayOutput Example Function 1112
FreeTwoDim Example Function 1114

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

Standard C Language Types 1157
Standard Visual Basic Language Types 1159
Visual Basic API Attributes Terminology 1164
Visual Basic Main API Structures 1165
ESB_APPDB_T 1165
ESB_APPINFO_T 1165
ESB_APPINFOEX_T 1166
ESB_APPSTATE_T 1167
ESB_ATTRIBUTEINFO_T 1168
ESB_ATTRSPECS_T 1169
ESB_DBFILEINFO_T 1170
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_GLOBAL_T 1179
ESB_INIT_T 1180
ESB_LOCKINFO_T 1181
ESB_MBRALT_T 1181
ESB_MBRUSER_T 1182
ESB_MEMBERINFO_T 1182
ESB_OBJDEF_T 1184
ESB_OBJINFO_T 1184
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_PARTSLCT_T 1189
ESB_PROCSTATE_T 1190
ESB_RATEINFO_T 1191
ESB_TIMERECORD_T 1191
ESB_USERAPP_T, ESB_GROUPAPP_T 1192
ESB_USERDB_T, ESB_GROUPDB_T 1192
ESB_USERINFO_T, ESB_GROUPINFO_T 1193

ESB_USERINFOEX_T 1194
ESB_VARIABLE_T 1196

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

ESB_MBRCOUNTS_T 1459
ESB_MBRINFO_T 1460
ESB_OUTERROR_T 1462
ESB_OUTLINEINFO_T 1463
ESB_PREDICATE_T 1464

Chapter 23. Visual Basic Outline API Functions 1465
Visual Basic Outline API Function Categories 1465
VB Outline API Alias Table Functions 1465
VB Outline API Attributes Functions 1466
VB Outline API Dynamic Time Series Functions 1466
VB Outline API Generation Name Functions 1467
VB Outline API Level Name Functions 1467
VB Outline API Member Administration Functions 1467
VB Outline API Member Alias Functions 1468
VB Outline API Member Formula Functions 1468
VB Outline API Member Traversal Functions 1468
VB Outline API Outline Administration Functions 1469
VB Outline API Outline Query Functions 1469
VB Outline API Setup and Cleanup Functions 1469
VB Outline API User Attribute Functions 1470
Visual Basic Outline API Function Reference 1470

Chapter 24. Example of Traversing an Outline (VB) 1579

Part VII. Other APIs 1581

Chapter 25. Java API Reference 1583

Chapter 26. MDX Provider API 1585
MDX Provider API General Information 1585
MDX Provider API Reference 1587
MDX Provider Declarations 1587
EssMdxExecuteQuery 1588
EssMdxFreeQuery 1588
EssMdxGetAxes 1589
EssMdxGetAxisInfo 1589
EssMdxGetAxisMembers 1590
EssMdxGetCellAtIndices 1590
EssMdxGetCellAtOffset 1591
EssMdxGetCellInfo 1591

EssMdxGetCellStatus 1592
EssMdxGetClusterDimMembers 1593
EssMdxGetClusterInfo 1593
EssMdxGetClusterMembers 1594
EssMdxGetClusters 1594
EssMdxGetDimInfo 1595
EssMdxGetFormatString 1596
EssMdxGetFormattedValue 1596
EssMdxGetMbrIdentifier 1597
EssMdxGetMbrProperty 1597
EssMdxGetNamedSets 1598
EssMdxGetPropertyInfo 1599
EssMdxGetQueryCellProperties 1600
EssMdxGetQueryOptions 1600
EssMdxGetSmartlistforCell 1601
EssMdxGetValue 1601
EssMDXIsCellGLDrillable 1602
EssMdxNewQuery 1603
EssMdxSetDataLess 1603
EssMDXSetHideData 1604
EssMdxSetMbrIdType 1604
EssMdxSetNeedCellStatus 1604
EssMdxSetQueryCellProperties 1604
EssMdxSetQueryOptions 1605
MDX Sample Client Program 1606

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

MDSCHEMA_MEMBERS Rowset 1640
MDSCHEMA_PROPERTIES Rowset 1643
MDSCHEMA_SETS Rowset 1647
MDSCHEMA_LEVELS Rowset 1647
DISCOVER_SCHEMA_ROWSETS Rowset 1650
DISCOVER_DATASOURCES Rowset 1650
DISCOVER_PROPERTIES Rowset 1651
DISCOVER_ENUMERATORS Rowset 1653
DISCOVER_KEYWORDS Rowset 1655
DISCOVER_LITERALS Rowset 1656
Flattened Rowset Examples 1657
MDX Examples 1657
XMLA Examples 1661

Appendix A. API Sample Programs 1665
Sample C API Program 1 (cs1.c) 1665
Sample C API Program 2 (cs2.c) 1672
Sample C API Program 3 (cs3.c) 1681
Sample Visual Basic API Program 1 (initialize.vbp) 1692
Sample Visual Basic API Program 2 (appdb.vbp) 1695
Sample Visual Basic API Program 3 (reports.vbp) 1703

Appendix B. Shared Services Migration and User Management API Example 1719

Appendix C. Limits 1727
Name Limits 1727
Essbase Server (Host) Name Limits 1727
Application Name Limits 1727
Database Name Limits 1728
Filter Name Limits 1728
Group Name Limits 1728
Object Name Limits 1728
Password Limits 1728
User Name Limits 1728
Drill-through URL Limits 1729

Index 1731

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/trsphonebk.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.

Part I

Preliminary Information

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

1 Introduction to the *Oracle Essbase API Reference*

In This Chapter

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

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

Convention	Purpose	Example
`monospace font`	Function, structure, file, directory, and environment variables names in text	`ESS_STS_T, ESSAPIW.LIB`
italic	Anything you replace with a value in syntax	EsbOtlCloseOutline (*hOutline*)
`" "`	Double quotes enclose text parameters or parameters that include a space	`"appName"` `SETDEFAULTCALC "CALC ALL";`
`()`	Parentheses enclose function parameters, show order of execution for operations	`EsbOtlDeleteMember (hOutline, hMember);` `(a + b) * c`
`//`	Comment marker indicates text from // to end of line should be ignored in processing	`// Gets results`
;	Statement terminator marks end of command	EXIT;

Using Function Reference Entries

Table 2 lists the information supplied by API function entries.

Table 2 API Function Entries

Function Entry	Description
Description	Brief description of the function.
Syntax	Function syntax. Function name and required keywords: **bold**. Parameters: *italics*.
Parameters	Definitions of the parameters of the function.
Return Value	Value returned by the function.
Notes	Notes on using the function.
Access	Level of security or other access required to use the function.
Example	How to use the function.
See Also	Related functions.

2 Building the Program

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 3 Supported Compilers

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

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.

```
# 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 odbc32.lib odbccp32.lib kernel32.lib
user32.lib gdi32.lib winspool.lib comdlg32.lib advapi32.lib shell32.lib ole32.lib
oleaut32.lib uuid.lib odbc32.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 Enterprise Performance Management System Certification Matrix* (http://www.oracle.com/technology/software/products/ias/files/fusion_certification.html)

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:

Format and Parts of Name	Example
InterfaceVerbObject	*Interface*
Programming interface	EssCreateGroup
Ess = C API	EssUpdate = C API
Esb = Visual Basic API	EsbUpdate = Visual Basic API
EssOtl = C Outline API	EssOtlOpenOutline = C Outline API
EsbOtl = Visual Basic Outline API	EsbOtlOpenOutline = Visual Basic Outline API
EssG = C Grid API	EssGSetGridOption = C Grid API
Verb Action to perform, such as "Report"	EssReportFile (no verb) sends the report
Object Object of action, such as "Group"	EssUpdate (no Object) acts on the current object

- **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:

Format and Parts of Name	Example
Interface_Name_Type	*Interface*

Format and Parts of Name	Example
Programming interface, either ESS or ESB	EssCreateGroup
Verb Data type, such as STR (string)	ESB_BOOL_T = Visual Basic typdef for Boolean
Type Type of structure, either T (typedef definition) or M (macro)	ESS_STR_T = C language typedef for String

- **C API constant names**—Begin with the prefix *ESS* that represents the C interface, includes a word that describes the constant, and has no suffix. Underscore characters separate the parts of the name. Names follow this format:

Format and Parts of Name	Example
Structure Data type or structure field, such as "Boolean"	ESS_Structure_Value
Value Type of value the constant stores	ESS_STS_NOERROR could store a value for the ESS_STS_T data type

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:

```
#ifdef WINNT
#pragma pack (1)
#endif
#include
#include
```

```
#pragma pack ()
#endif
```

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 architechtures (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.

```
# 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.

```
# 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.

```
# Compiler Flags
CC=cc_r
```

```
CFLAGS = -qcpluscmt -I$(<Location of API>)/api/include -g

# Library files;
LIBS = -L$(<Location of API>)/api/lib -lessapinuS -lessotlnuS -lessgapinuS
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
```

```
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`.

```
# 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.

3 Integrating Essbase With Your Product

In This Chapter

Essbase Directory Structure....37
Customizing the Run-Time Environment....38
API Files You Need to Ship....41
API Libraries....42
Installing Your Application Program....43
Optimizing TCP/IP Networking for API Clients....44

Essbase Directory Structure

A 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

Directory	Description
`\root`	Root directory: All Essbase files
`\root\bin`	Binary directory: executables, shared libraries, and other program files
`\root\client`	Client directory: Client application and database files
`\root\client\appname`	Files relating to the application appName (one for each application)
`\root\client\appname\dbname`	Files relating to the database dbName in the application appName (one for each database in the application)

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

Note: The *root* directory name cannot include spaces.

Table 5 Essbase API and Run-Time Client Directory Structure (Windows)

Component	Directory
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 \Appdirectory 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: ESS_INIT_T InitStruct;Initstruct.LocalPath = "D:\PRODUCT";

For Visual Basic, Dim pInit as eSB_INIT_TpInit.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 ES*X*_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: ESS_INIT_T InitStruct;Initstruct.MessageFile = "C:\PRODUCT\MESSAGE\PRODUCT.MDB";

For Visual Basic,

```
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 yourAUTOEXEC.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 ESX_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 ESX_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:

```
ESS_INIT_T InitStruct;   InitStruct.HelpFile = "C:\PRODUCT\HELP\PRODUCT.HLP";
```

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.

- **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.sl`
  - `libessotlnu.sl`
  - `libessgapinu.sl`
- **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`

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:

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,

```
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:

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

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

```
/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:

```
echo 30 > /proc/sys/net/ipv4/tcp_fin_timeout
```

4 Building a Simple API Program

In This Chapter

Introduction....47
Design Environment Setup Issues....47
Basic Requirements....48
Assembling a Program....55

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 prefic "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):

```
#pragma pack(push,localid,1)
#include <essapi.h>
#pragma pack(pop,localid)
```

- 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:

```
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 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 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:

```
allocate resource 1
begin action 1
        allocate resource 2
        begin action 2
                action 2
        end action 2
```

```
        free resource 2
end action 1
free resource 1
```

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, **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 (`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, ESX_DBINFO_T) is passed in at the end:

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

' 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
```

Note that in the C example, the returned argument (`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 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
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 `EssInit` 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 Example of a login using the EssLogin function */
ESS_STS_T       sts;
```

```
ESS_HINST_T     hInst;
ESS_SVRNAME_T   Server = "Larch";
ESS_USERNAME_T  Username = "Joe User";
ESS_PASSWORD_T  Password = "secret";
ESS_ACCESS_T    Access;
ESS_HCTX_T      hCtx = ESS_INVALID_HCTX;
sts = EssLogin (hInst, Server, Username, Password, &Access, &hCtx);

' 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
Server = "Larch"
Username = "Joe User"
Password = "secret"
hCtx = ESB_INVALID_HCTX
sts = EsbLogin (hInst, Server, Username, Password, Access, hCtx)
```

The following is a similar example of logging in, this time using **EsxAutoLogin**(). When using this function, the user supplies all the relevant information (server name, user name, password, application, and database names) by entering the information into the appropriate fields of the dialog box:

```
/* C Example of a login using the EssAutoLogin function */
ESS_STS_T               sts;
ESS_HINST_T             hInst;
ESS_ACCESS_T            Access;
ESS_HCTX_T              hCtx = ESS_INVALID_HCTX;
sts = EssAutoLogin (hInst, ESS_NULL, ESS_NULL, ESS_NULL, ESS_NULL,
        ESS_NULL, AUTO_DEFAULT, &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
```

```
Dim     hCtx as ESB_HCTX_T
Dim     AppName as ESB_APPNAME_T
Dim     DbName as ESB_DBNAME_T
Dim     Access as ESB_ACCESS_T
AppName = "Sample"
DbName = "Basic"
sts = EsbSetActive (hCtx, AppName, DbName, Access)
```

The user's access level to the selected database is returned by EssSetActive (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 Example of logging a user out */
ESS_STS_T       sts;
ESS_HCTX_T      hCtx;
sts = EssLogout (hCtx);
hCtx = ESS_INVALID_HCTX;

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

' 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" on page 56
- "Editing the Outline" on page 56
- "Loading Data" on page 58
- "Reporting" on page 58
- "Updating Data" on page 65
- "Calculating Data" on page 66
- "Using Security" on page 67
- "Maintaining Applications and Databases" on page 68
- "Handling Messages" on page 69
- "Managing Memory" on page 71

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.

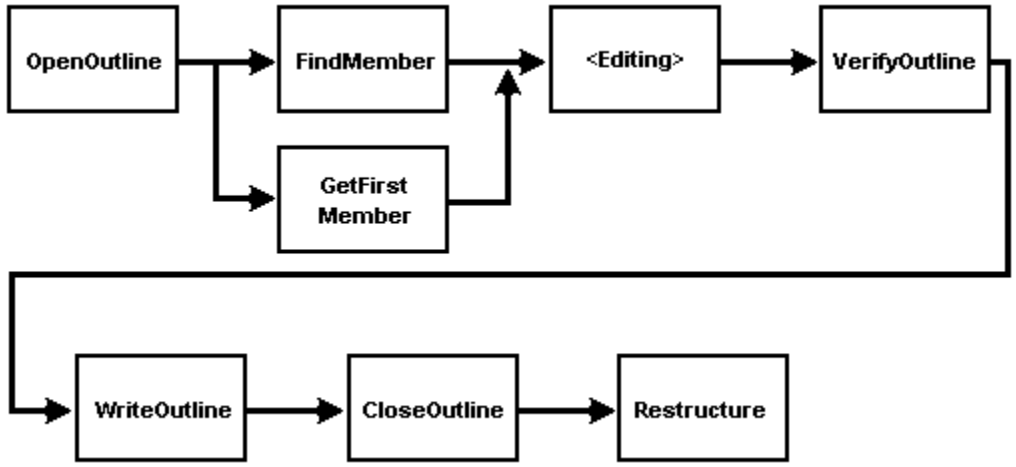
The API can automate the process of rebuilding dimensions dynamically from a data file or SQL source. To automate the process you must first create rules files by using Administration Services Console and then use the rules files to build the dimensions by calling `EssBuildDimension` or `EsbBuildDimension`.

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     794
```

- {**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:

```
        Actual  Sales   Ohio
        Jan     Feb     Mar
Product 287     217     290
```

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

  - <CHILDREN / <ICHILDREN
  - <DESCENDANTS / <IDESCENDANTS
  - <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 <IDESC) 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:

```
        Actual  Sales   Ohio
        Jan     Feb     Mar     Qtr1
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
300-10  30      19      32      81
300-20  24      16      25      65
300-30  12      7       11      30
 300    66      42      68      176
400-10  30      27      32      89
400-20  14      10      12      36
400-30  5       3       4       12
 400    49      40      48      137
100-20  44      38      49      131
200-20  18      14      18      50
300-30  12      7       11      30
 Diet   74      59      78      211
```

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).

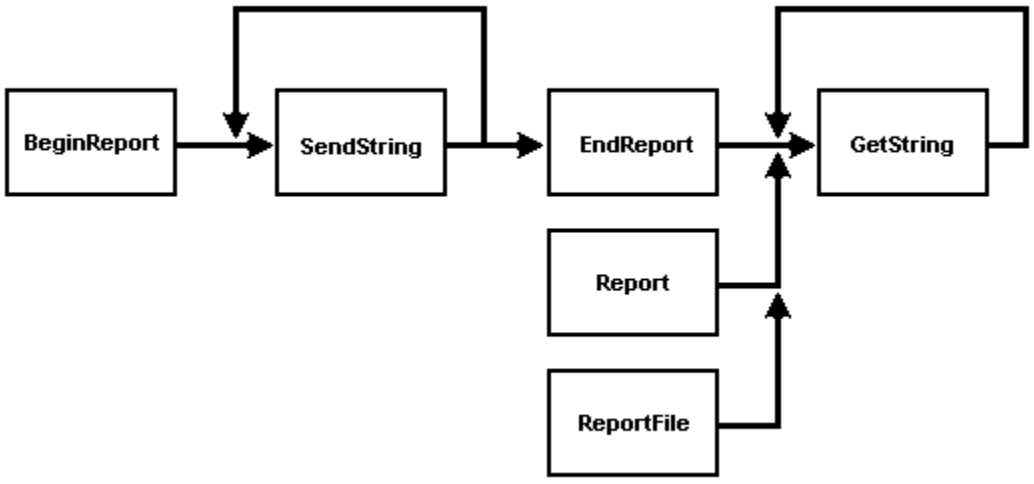
See "API Libraries" on page 42.

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).

For more detailed descriptions of all these functions, see `EssReport`, `EssReportFile`, `EssBeginReport`, `EsbReport`, `EsbReportFile`, and `EsbBeginReport`

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:

```
        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
```

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

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

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:

```
              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
```

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.

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

- {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
```

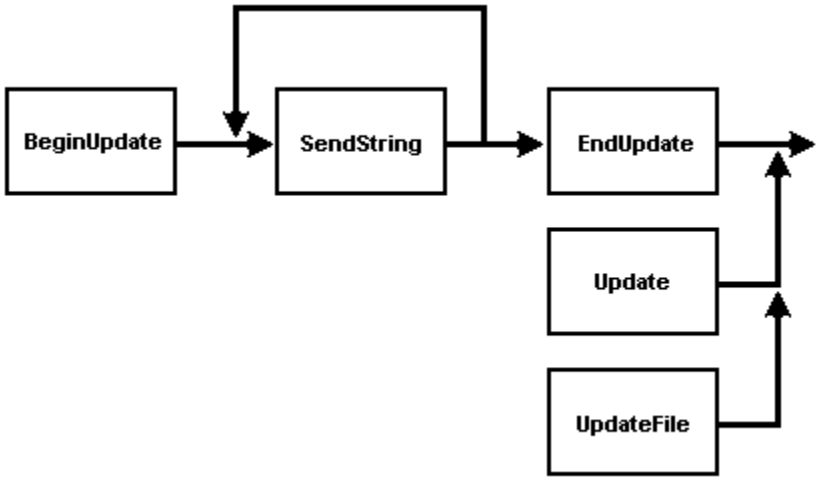
Also, all of the dimensions (or all but one) need to be included in the <ROW command in the report, to ensure that the data is returned in a fully normalized form.

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.

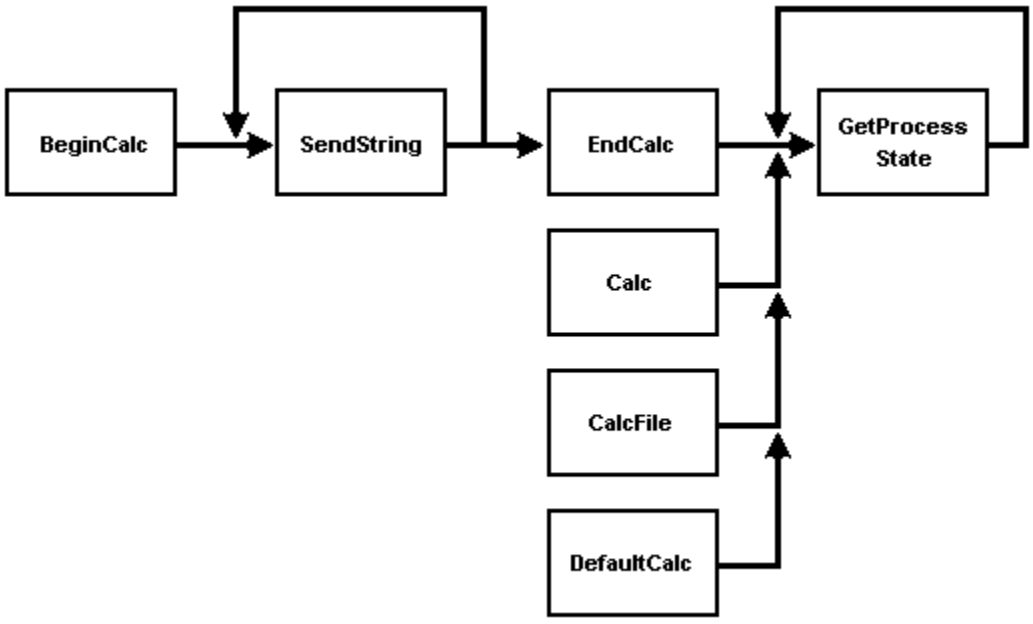
To execute an update from a file, first lock the blocks as described above and then call **EsxUpdateFile**().

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 **EsxDefaultCalc**(). To get and set the script used for a default calculation, use **EsxGetDefaultCalc**() and either **EsxSetDefaultCalc**() or **EsxSetDefaultCalcFile**().

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

- By passing the calc specification as a string to **EsxCalc**()
- By passing a set of strings with **EsxBeginCalc**(), **EsxSendString**(), and **EsxEndCalc**()
- By specifying a calc script file with **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 **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 `EssCalc`, `EssBeginCalc`, `EssCalcFile`, `EsbCalc`, `EsbBeginCalc`, and `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 **EsxCreateUser**() and **EsxDeleteUser**(). To set a user's password, use **EsxSetPassword**(). To get a list of users on a server, use **EsxListUsers**().

To get and set a user's or a group's security information, call **EsxGetUser**() and **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 **EsxGetGroupList**() and **EsxSetGroupList**().

To get user access privileges to an application, call **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 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);
```

```
                        GlobalUnlock (hMem);
                        GlobalFree (hMem);
                }
        }
        return (sts);
}
```

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:

```
' 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
```

```
  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.

5 Unicode Issues in Essbase API Programs

In This Chapter

General Programming Considerations 73
Defining Unicode Mode Client Programs 73
Specifying the Byte Order Encoding....75
Unicode Mode and Essbase Server 76
Unicode Outlines....77
Grid API 77

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" on page 74
- "Unicode-enabled Clients in Non-Unicode Mode" on page 74
- "Unicode-enabled Clients in Unicode Mode" on page 75
- "Specifying Unicode Mode" on page 75

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:

```
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

6 Using the C Main API

In This Chapter

C Main API Instance Handles 81
C Main API Context Handles 82
C Main API File Objects....82
Using Memory in C Programs 84
C Main API Message Handling....85
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

C Main API 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 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.

Using the Instance Handle in an Application

An instance handle is declared as type **ESS_HINST_T** in C programs.

The instance handle must be passed to the **EssLogin**() call, which returns a context handle, and also to the API terminate function **EssTerm**() 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 can terminate the API.

Note: A thread may require its own instance handle (*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 **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 **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 **EssLogin**(), using the same user name or different user names on one or more Essbase servers. Each call to **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 **EssSetActive**() function or the **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 alphanumberic 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 alocal 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:

```
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:

```
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 :
  [46] seconds
  ```

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 hadling. If an error event is incountered 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.

Problem	Solution
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.

7 C Main API Declarations

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)

Data Type	Essbase Type
typedef char	ESS_CHAR_T
typedef short	ESS_SHORT_T
typedef long	ESS_LONG_T
typedef unsigned char	ESS_UCHAR_T
typedef unsigned short	ESS_USHORT_T
typedef unsigned long	ESS_ULONG_T

Data Type	Essbase Type
typedef float	ESS_FLOAT_T
typedef double	ESS_DOUBLE_T
If win32 && _USE_32BIT_TIME_T defined: typedef __time32_t Otherwise, typedef time_t	ESS_TIME_T *
typedef unsigned short	ESS_DATE_T
If win32 && _USE_32BIT_TIME_T defined: typedef __time32_t Otherwise, typedef time_t	ESS_DATETIME_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)

Data Type	Essbase Type	Description
typedef void	*ESS_HCTX_T	API context handle
typedef void	*ESS_HINST_T	API instance handle
typedef unsigned char	ESS_BOOL_T	boolean
typedef size_t	ESS_SIZE_T	size of a memory block
typedef char	*ESS_STR_T	string (array of char)
typedef void	ESS_VOID_T	void

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**.

Data Type	Essbase Type	Description
typedef unsigned short	ESS_ ACCESS_T	Security access level. Possible bit values are: ● ESS_PRIV_NONE - 0x0000 - no privilege ● ESS_PRIV_READ - 0x0001 - read data ● ESS_PRIV_WRITE - 0x0002 - write data ● ESS_PRIV_CALC - 0x0004 - calculate data ● ESS_PRIV_DBLOAD - 0x0010 - load and unload databases ● ESS_PRIV_DBDESIGN - 0x0020 - manage databases ● ESS_PRIV_DBCREATE - 0x0040 - create, delete, and edit databases ● ESS_PRIV_APPLOAD - 0x0100 - load and unload applications ● ESS_PRIV_APPDESIGN - 0x0200 - manage applications ● ESS_PRIV_APPCREATE - 0x0400 - create, delete, and edit applications ● ESS_PRIV_USERCREATE - 0x1000 - create, delete, and edit users 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 - 0x00ff - full database access ● ESS_ACCESS_APPALL - 0x0fff - 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	Essbase Type	Description
typedef unsigned long	ESS_ OBJTYPE_T	File object type. 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)

Data Type	Essbase Type	Description
char	*ESS_PCHAR_T	pointer to char
unsigned char	*ESS_PUCHAR_T	pointer to unsigned char
short	*ESS_PSHORT_T	pointer to short
unsigned short	*ESS_PUSHORT_T	pointer to unsigned short
long	*ESS_PLONG_T	pointer to long

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_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_PHCTX_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

Miscellaneous Types (C)

Data Type	Essbase Type	Description
typedef long	ESS_STS_T	return value from API functions
typedef ESS_STS_T	(*ESS_FUNC_T)()	pointer to function

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**].

Data Type	Essbase Type	Description
typedef char	ESS_USERNAME_T	user name
typedef char	ESS_PASSWORD_T	password
typedef char	ESS_SVRNAME_T	server name
typedef char	ESS_APPNAME_T	application name
typedef char	ESS_DBNAME_T	database name
typedef char	ESS_OBJNAME_T	object name

Data Type	Essbase Type	Description
typedef char	ESS_MBRNAME_T	member name
typedef char	ESS_FTRNAME_T	filter name
typedef char	ESS_ALIASNAME_T	alias table name
typedef char	ESS_PATH_T	file path name
typedef char	ESS_DESC_T	app/database description

API Definitions (C)

Essbase Type	Value
#define ESS_TRUE	1
#define ESS_FALSE	0
#define ESS_NULL	NULL
#define ESS_NATIVE_SECURITY	1
#define ESS_SS_SECURITY	2

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.

Value	Definition
ESS_BASE_DIMENSION	A dimension that is not an attribute dimension
ESS_BASE_MEMBER	A member that is not an attribute member
ESS_ATTRIBUTE_DIMENSION	An attribute dimension
ESS_ATTRIBUTE_MEMBER	An attribute member
ESS_ATTRIBUTED_MEMBER	A base member or dimension that has attributes associated with it. Also called a standard member or dimension.

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

Value	Definition
ESS_MBRSTS_ATTRIBUTE	Attribute member status

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

Value	Definition
ESS_TTYPE_ATTRIBUTE	Attribute tag
ESS_TTYPE_ATTRCALC	Attribute calculation tag. Used internally for aggregation.

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.

Value	Definition
ESS_ATTRMBRDT_BOOL	Boolean data type
ESS_ATTRMBRDT_DATETIME	Datetime data type
ESS_ATTRMBRDT_DOUBLE	Double data type
ESS_ATTRMBRDT_STRING	String data type
ESS_ATTRMBRDT_NONE	No data type

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

Value	Definition
ESS_EQ	Equal to

Value	Definition
ESS_NEQ	Not equal to
ESS_GT	Greater than
ESS_LT	Less than
ESS_GTE	Greater than or equal to
ESS_LTE	Lesser than or equal to
ESS_TYPEOF	Type of
ESS_ALL	All

Table 6 C API Attributes Terminology

Term	Definition
bucketing type	When building a dimension, you can associate a zero-level attribute member of type ESS_ATTRMBRDT_DOUBLE with a range of data in a relational source. Bucketing type determines the upper or lower limit of the data range. See usBucketingType.
ESS_ATTRIBUTE_DIMENSION ESS_ATTRIBUTE_MEMBER	ESS_ATTRIBUTE_DIMENSION is an attribute dimension. ESS_ATTRIBUTE_MEMBER is a member of an attribute dimension. See "ESS_ATTRIBUTEQUERY_T" on page 697. Also see EssCheckAttributes.
ESS_ATTRIBUTED_MEMBER	ESS_ATTRIBUTED_MEMBER is a member (of a base dimension) which has an attribute member associated with it. See "ESS_ATTRIBUTEQUERY_T" on page 697. Also see EssCheckAttributes.
ESS_BASE_DIMENSION ESS_BASE_MEMBER	ESS_BASE_DIMENSION is a standard dimension that has an attribute dimension associated with it. ESS_BASE_MEMBER is a member of a base dimension. See "ESS_ATTRIBUTEQUERY_T" on page 697. Also see EssCheckAttributes.
ESS_STANDARD_DIMENSION ESS_STANDARD_MEMBER	ESS_STANDARD_DIMENSION is any dimension that is not an attribute dimension. ESS_STANDARD_MEMBER is a member of a standard dimension. See "ESS_ATTRIBUTEQUERY_T" on page 697. Also see EssCheckAttributes.

Term	Definition
long name	A zero-level attribute member that is not of type ESS_ATTRMBRDT_STRING is uniquely identified by a long name. A zero-level attribute member of type ESS_ATTRMBRDT_STRING must itself be unique. See the following structures: ● "ESS_ATTRSPECS_T" on page 119 ● "ESS_ATTRIBUTEINFO_T" on page 118 Also see the following functions: ● EssGetAttributeSpecifications ● EssOtlGetAttributeSpecifications ● EssOtlSetAttributeSpecifications And, see Notes on Adding an Attribute Member.
short name	A zero-level attribute member that is not of type ESS_ATTRMBRDT_STRING is called a short name. It is provided to a function as a parameter of type ESS_STR_T. See EssOtlFindAttributeMembers.

Dimension Tag Constants (C)

The following constants define the available information flags used in the *DimTag* field of the "ESS_DIMENSIONINFO_T" on page 132 structure.

Constant	Definition
ESS_TTYPE_NONE	No dimension type. Value for *DimTag* field of ESS_DIMENSIONINFO_T.
ESS_TTYPE_CCATEGORY	Accounts: Currency ACCOUNTS tag. Value for *DimTag* field of ESS_DIMENSIONINFO_T
ESS_TTYPE_CNAME	Country: Currency COUNTRY tag. Value for *DimTag* field of ESS_DIMENSIONINFO_T
ESS_TTYPE_CTIME	Time: Currency TIME tag. Value for *DimTag* field of ESS_DIMENSIONINFO_T
ESS_TTYPE_CTYPE	Type: Currency TYPE tag. Value for *DimTag* field of ESS_DIMENSIONINFO_T
ESS_TTYPE_CPARTITION	Currency PARTITION tag. Value *DimTag* field of ESS_DIMENSIONINFO_T

Implied Share Setting (C)

Implied Share settings can apply to a specific outline, using the EssOtlGetImpliedShare and 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:

Value	Description
ESS_IMPLIEDSHARE_DEFAULT	Can be set using `EssOtlSetImpliedShare`. When set, immediately gets converted to either _ON or _OFF. If returned: ● Outline has no Implied Share setting ● Implied Share is ON
ESS_IMPLIEDSHARE_DEFAULT_ON	Return value only available with `EssOtlGetImpliedShare`. If returned: ● Outline uses Implied Share default setting in Essbase.cfg ● Essbase.cfg might contain an Implied Share entry (ON) ● Essbase.cfg should contain no entry ● Implied Share is ON
ESS_IMPLIEDSHARE_DEFAULT_OFF	Return value only available with `EssOtlGetImpliedShare`. If returned: ● Outline uses Implied Share default setting in Essbase.cfg ● Essbase.cfg contains an Implied Share entry (OFF) ● Implied Share is OFF
ESS_IMPLIEDSHARE _FORCE_ON	Can be set using `EssOtlSetImpliedShare`. An explicit setting indicating the outline always has Implied Share ON.
ESS_IMPLIEDSHARE _FORCE_OFF	Can be set using `EssOtlSetImpliedShare`. An explicit setting indicating the outline always has Implied Share OFF.

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.

Constant	Definition
ESS_DBREQFLAG_CALCDEF	Default flag for `DbReqFlags` field. Used the default calc script. Value: 0x00000001.
ESS_DBREQFLAG_CALCDSCR	Custom calc script flag for `DbReqFlags` field. Used a custom calc script. Value: 0x00000002.

List Option Constants (C)

The following constants define request types used by the *ListOption* field of the `EssListTransactions` function.

Constant	Definition
ESS_LIST_TRANSACTIONS_TOCLIENT	Write the output to the screen.
LIST_TRANSACTIONS_TOFILE	• Write output to a CSV file. • Output is not returned in ppResults. • pCount and ppResults will be NULL. • Content is written to the FileName as comma separated file. • If the specified file name exists, the command fails.
ESS_LIST_TRANSACTIONS_FORCETOFILE	• Write output to a CSV file. • Output is not returned in ppResults. • pCount and ppResults will be NULL. • Content is written to the FileName as comma separated file. • If the specified file name exists, it is overwritten with the new output.

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:

Constant	Definition
ESS_ALIASNAMELEN	Maximum length of an alias table name
ESS_APPNAMELEN	Maximum length of an application name
ESS_CRDB_MAXIMUM	Maximum dimension number for a Currency database
ESS_DBNAMELEN	Maximum length of a database name
ESS_DESCLEN	Maximum length of an application or database description
ESS_FTRNAMELEN	Maximum length of a filter name
ESS_LINELEN	Maximum length of a line in a report
ESS_MBRCOMMENTEXLEN	Maximum length of an extended member comment
ESS_MBRNAMELEN	Maximum length of a member name
ESS_NAMELEN	Maximum length of a general name
ESS_PASSWORDLEN	Maximum length of a user password
ESS_PATHLEN	Maximum length of a file path name

Constant	Definition
ESS_OBJNAMELEN	Maximum length of an object name
ESS_SVRNAMELEN	Maximum length of a server name
ESS_USERNAMELEN	Maximum length of a user or group name

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.

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

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.

Constant	Definition
ESS_INDEXCACHEMIN_SIZE	Minimum index cache size for the *MaxMemIndex* field of the ESS_DBSTATE_T structure. Value: 1048576. No maximum value is defined.
ESS_INDEXPAGEMAX_SIZE	Maximum index page size for the *IndexPageSize* field of the ESS_DBSTATE_T structure. Value: 8192
ESS_INDEXPAGEMIN_SIZE	Minimum index page size for the *IndexPageSizeMin* field of the ESS_DBSTATE_T structure. Value: 1024

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.

Constant	Definition	Description
ESS_API_NONUNICODE	0x0002	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.
ESS_API_UTF8	0x0003	The program is a Unicode mode client program. The client program is passing a UTF-8 encoded argument to the API.

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_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.

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

Data Type	Field	Description
ESS_LROTYPE_URL_API	2	Value specifying that a linked object is a URL

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:

Data Type	Field	Description
ESS_ULONG_T	*cellOffset*	Cell offset within a data block
ESS_SECPART_T	*blkOffset*	Block offset
ESS_SECPART_T	*segment*	Segment number

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:

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

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:

Data Type	Field	Description
ESS_CELLADDR_API_T	*cellKey*	Cell address
ESS_LONG_T	*hObject*	Internal object handle

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:

Data Type	Field	Description
ESS_CHAR_T	*objName[ESS_ONAMELEN_API]*	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.
ESS_CHAR_T	*objDesc[ESSW_LRODESCLEN_API]*	Description of an object linked to a data cell. ESS_LRODESCLEN_API specifies the maximum length of the description; the default value is 79.

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

"ESS_PARTOTL_MBRASSOCCHG_API_T" on page 164

"ESS_PARTOTL_MBRATTR_API_T" on page 164

"ESS_PARTOTL_MBRCHG_API_T" on page 165

"ESS_PARTOTL_MBR_RSRVD_API_T" on page 164

"ESS_PARTOTL_CHG_FILE_T" on page 160

"ESS_PARTOTL_QRY_FILTER_T" on page 170"ESS_PARTOTL_QUERY_T" on page 169

"ESS_PARTOTL_READ_T" on page 170

"ESS_PARTOTL_NAMECHG_API_T" on page 167

"ESS_PARTOTL_NAMED_GENLEV_API_T" on page 167

"ESS_PARTOTL_NAMEMAP_API_T" on page 167

"ESS_PARTOTL_CHANGE_API_T" on page 160

"ESS_PARTOTL_DIMCHG_API_T" on page 162

"ESS_PARTOTL_SELECT_APPLY_T" on page 171

"ESS_PARTOTL_SELECT_CHG_T" on page 171

"ESS_PARTSLCT_T" on page 171

"ESS_PARTSLCT_VALIDATE_T" on page 172

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:

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

Pointer Types

Data Type	Essbase Type	Description
char	*ESS_PCHAR_T	pointer to char
unsigned char	*ESS_PUCHAR_T	pointer to unsigned char
short	*ESS_PSHORT_T	pointer to short
unsigned short	*ESS_PUSHORT_T	pointer to unsigned short
long	*ESS_PLONG_T	pointer to long
unsigned long	*ESS_PULONG_T	pointer to unsigned long
double	*ESS_PDOUBLE_T	pointer to double

Data Type	Essbase Type	Description
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_PHCTX_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 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:

Constant	Definition
ESS_ALIASNAMELEN	Maximum length of an alias table name
ESS_APPNAMELEN	Maximum length of an application name
ESS_CRDB_MAXIMUM	Maximum dimension number for a Currency database
ESS_DBNAMELEN	Maximum length of a database name
ESS_DESCLEN	Maximum length of an application or database description
ESS_DESCRIPTION_LEN	Maximum string length (255) used for drill-through

Constant	Definition
ESS_DTREPORT_NAME	Maximum string length (80) used for drill-through
ESS_FTRNAMELEN	Maximum length of a filter name
ESS_LINELEN	Maximum length of a line in a report
ESS_MAX_DATALEN	Maximum string length (255) used for drill-through
ESS_MAX_NAME	Maximum string length (30) used for drill-through
ESS_MBRCOMMENTEXLEN	Maximum length of an extended member comment
ESS_MBRNAMELEN	Maximum length of a member name
ESS_NAMELEN	Maximum length of a general name
ESS_PASSWORDLEN	Maximum length of a user password
ESS_PATHLEN	Maximum length of a file path name
ESS_OBJNAMELEN	Maximum length of an object name
ESS_SVRNAMELEN	Maximum length of a server name
ESS_USERNAMELEN	Maximum length of a user or group name

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.

Constant	Definition
ESS_DTAPI_PROMPT_HISNAME	A value for *uInputOption* which means that the user can connect to Essbase Studio to perform a drill-through
ESS_DTAPI_PROMPT_LOGIN	A value for *uInputOption* which means that a password is required to connect to Essbase Studio to perform a drill-through

Pointer Types

Data Type	Essbase Type	Description
char	*ESS_PCHAR_T	pointer to char
unsigned char	*ESS_PUCHAR_T	pointer to unsigned char
short	*ESS_PSHORT_T	pointer to short
unsigned short	*ESS_PUSHORT_T	pointer to unsigned short
long	*ESS_PLONG_T	pointer to long

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_PHCTX_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:

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

Data Type	Field	Description
ESS_APPNAME_T	*AppName*	The application name
ESS_DBNAME_T	*DbName*	The database name

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 117structure, 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;
```

Data Type	Field	Description
ESS_APPNAME_T	*Name*	The application name
ESS_SVRNAME_T	*Server*	The server name
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	AppType	The type of application. Valid values are: ● 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. ● ESS_APP_NONUNICODE - 0x0002 - The program is a non-Unicode mode client program.
ESS_CHAR_T	AppLocale	The application locale description, of type ESS_LOCALESTRING_LENGTH.
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_USHORT_T	*nDbs*	The number of databases in this application

Data Type	Field	Description
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_DBNAME_T	*DbNames [1]*	A dynamic array (with *nDb* elements) of database name strings listing all the databases in the application.

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:

```
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	Field	Description
ESS_APPNAME_T	*Name*	Application name
ESS_SVRNAME_T	*Server*	Server name
ESS_USHORT_T	AppType	The type of application. Valid values are: ● 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. ● ESS_APP_NONUNICODE - 0x0002 - The program is a non-Unicode mode client program.
ESS_CHAR_T	AppLocale	The application locale description, of type ESS_LOCALESTRING_LENGTH.

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:

```
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)

Data Type	Field	Description
ESS_ACCESS_T	*Access*	The default access to databases in the application (the lowest possible level of access for all users). This field can contain the following values: ● ESS_PRIV_NONE ● ESS_PRIV_DBDESIGN ● ESS_PRIV_CALC ● ESS_PRIV_WRITE ● ESS_PRIV_READ
ESS_BOOL_T	*Connects*	Flag to indicate whether users can connect to the application (ESS_TRUE if users can connect).
ESS_BOOL_T	*Commands*	Flag to indicate whether users can issue commands to the application (ESS_TRUE if the application is accepting user commands).
ESS_BOOL_T	*Updates*	Flag to indicate whether users can update data in the application (ESS_TRUE if the application is accepting user update commands).
ESS_BOOL_T	*Security*	Flag to indicate whether application security is enabled (ESS_TRUE if security is enabled).
ESS_ULONG_T	*LockTimeout*	Timeout period (in seconds) after which block-level locks are automatically removed. This field does not apply to aggregate storage databases.
ESS_ULONG_T	*lroSizeLimit*	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.

ESS_ATTRIBUTEINFO_T

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

```
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;
```

Data Type	Field	Description
ESS_MBRNAME_T	*MbrName*	Attribute member name from "ESS_MEMBERINFO_T" on page 149 or "ESS_MBRINFO_T" on page 699, including a long name
ESS_MBRNAME_T	*DimName*	Attribute dimension name
"ESS_ATTRIBUTEVALUE_T" on page 118	*Attribute*	Attribute value

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;
```

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

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;
```

```
  ESS_STR_T     pszDefaultMaxMbrName;
} ESS_ATTRSPECS_T, *ESS_PATTRSPECS_T, **ESS_PPATTRSPECS_T;
```

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_CHAR_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").

Data Type	Field	Description
ESS_STR_T	*pszDefaultMinMbrName*	The name used with the attribute calculations (aggregate) dimension to indicate MINIMUM. The default value is ESS_DEFAULT_MIN ("Min").
ESS_STR_T	*pszDefaultMaxMbrName*	The name used with the attribute calculations (aggregate) dimension to indicate MAXIMUM. The default value is ESS_DEFAULT_MAX ("Max").

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;
```

Data Type	Field	Description
ESS_USHORT_T	*usProcessState*	The state of dimension build/data load process: whether it is in progress, in the final stages, or completed. For values, see "Constant Values for *usProcessState*.
ESS_USHORT_T	*usProcessStage*	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 "Constant Values for *usProcessStage*."
ESS_LONG_T	*ilProcessStatus*	The status of the dimension build/data load process (same as function return status)
ESS_ULONG_T	*ulRecordsProcessed*	The number of data records processed so far
ESS_ULONG_T	*ulRecordsRejected*	The number of data records rejected so far

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;
```

Data Type	Field	Description
ESS_USERNAME_T	*Name*	The name of the logged in user.
ESS_APPNAME_T	*AppName*	Name of currently connected application (if applicable).
ESS_DBNAME_T	*DbName*	Name of the currently connected database(if applicable).
ESS_SVRNAME_T	*LoginMachine*	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.
ESS_ULONG_T	*LoginIP*	The IP address of the logged in machine.
ESS_TIME_T	*LastLogin*	The time of the last login.

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;
```

Data Type	Field	Description
ESS_USERNAME_T	*Name*	Name of the logged in user
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_APPNAME_T	*AppName*	Name of the currently connected application (if applicable)
ESS_DBNAME_T	*DbName*	Database name
ESS_SVRNAME_T	*LoginMachine*	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`.
ESS_ULONG_T	*LoginIP*	IP address of the logged in machine
ESS_TIME_T	*LastLogin*	Time of the last login

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;
```

Data Type	Field	Description
ESS_APPNAME_T	*AppName*	Application name
ESS_DBNAME_T	*DbName*	Database name
ESS_FILENAME_T	*FilePath*	File path
ESS_SIZE_T	*FileSize*	File size in bytes
ESS_USHORT_T	*FileSequenceNum*	The 1-based sequence number of the file within the set of files of its *FileType* for the specified database
ESS_USHORT_T	*FileCount*	Number of files of its *FileType* returned

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 127structure, which contains additional database state parameters that can be modified, and the "ESS_DBSTATS_T" on page 130 structure. The fields are:

```
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_ULONG_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_ULONG_T        DataFileCacheSetting;
   ESS_ULONG_T        DataFileCacheSize;
   ESS_ULONG_T        DataCacheSetting;
   ESS_ULONG_T        DataCacheSize;
   ESS_ULONG_T        IndexCacheSetting;
   ESS_ULONG_T        IndexCacheSize;
   ESS_ULONG_T        IndexPageSetting;
   ESS_ULONG_T        IndexPageSize;
   ESS_DBREQINFO_T    DbReqInfoAry[ESS_DBREQNUM];
   ESS_BOOL_T         bDbReadOnly;
   ESS_BOOL_T         bDataCompress;
   ESS_USHORT_T       usDataCompressType;
   ESS_ULONG_T        ulRetrievalBuffer;
   ESS_ULONG_T        ulRetrievalSortBuffer;
   ESS_BOOL_T         bCacheMemLocking;
   ESS_BOOL_T         bPreImage;
   ESS_USHORT_T       usIsolationLevel;
   ESS_LONG_T         lTimeOut;
   ESS_ULONG_T        ulCommitBlocks;
   ESS_ULONG_T        ulCommitRows;
   ESS_ULONG_T        ulDiskVolumeCount;
   ESS_DISKVOLUME_T   aDiskVolume[1];
```

```
} ESS_DBINFO_T, *ESS_PDBINFO_T, **ESS_PPDBINFO_T;
```

Data Type	Field	Description
ESS_APPNAME_T	*AppName*	The associated application name
ESS_DBNAME_T	*Name*	The database name
ESS_USHORT_T	*DbType*	Database type. Values: ● ESS_DBTYPE_NORMAL ● ESS_DBTYTPE_CURRENCY
ESS_USHORT_T	*Status*	DatabaseLoad status (loaded or not loaded . Values: ● ESS_STATUS_NOTLOADED ● ESS_STATUS_LOADING ● ESS_STATUS_LOADED ● ESS_STATUS_UNLOADING
ESS_USHORT_T	*nConnects*	Number of users currently connected to the database
ESS_USHORT_T	*nLocks*	Number of data blocks currently exclusively locked
ESS_ULONG_T	*nDims*	Number of dimensions in database
ESS_USHORT_T	*Data*	Flag indicating loading state of the data in the database. Values: ESS_DBDATA_NONE: no data loaded ESS_DBDATA_LOADNOCALC: data loaded but not calculated ESS_DBDATA_CLEAN: data loaded and calculated
ESS_MBRNAME_T	*Country*	The currency country dimension member, if any. If none, the first byte is NULL.
ESS_MBRNAME_T	*Time*	Currency time dimension member, if any. If none, the first byte is NULL.
ESS_MBRNAME_T	*Category*	The currency category dimension member, if any. If none, the first byte is NULL.
ESS_MBRNAME_T	*Type*	Currency type dimension member (currency databases only). If none exists, the first byte is NULL.
ESS_MBRNAME_T	*CrPartition*	The currency partition member (non-currency databases only)
ESS_TIME_T	*ElapsedDbTime*	Number of seconds the database has been loaded
ESS_ULONG_T	*DataFileCacheSetting*	The Data File Cache Size setting value currently in effect.
ESS_ULONG_T	*DataFileCacheSize*	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.

Data Type	Field	Description
ESS_ULONG_T	*DataCacheSetting*	The Data Cache Size setting value (in KB) currently in effect.
ESS_ULONG_T	*DataCacheSize*	The run-time size (in KB) of the Data Cache/
ESS_ULONG_T	*IndexCacheSetting*	The Index Cache Size (in KB) setting value currently in effect.
ESS_ULONG_T	*IndexCacheSize*	Run-time size (in KB) of the Index Cache.
ESS_ULONG_T	*IndexPageSetting*	The Index Page Size setting (in KB) currently in effect.
ESS_ULONG_T	*IndexPageSize*	Run-time size (in KB) of an Index Page.
"ESS_DBREQINFO_T" on page 126	*DbReqInfo, Ary[ESS_DBREQNUM]*	Array for request information, including last calc, dataLoad, and outline update
ESS_BOOL_T	*bDbReadOnly*	TRUE if the database is in read-only mode; FALSE otherwise.
ESS_BOOL_T	*bDataCompress*	Optional compression flag (the default is YES).
ESS_USHORT_T	*usDataCompressType*	The data compression type if the optional compression flag is set (the default is BitMap).
ESS_ULONG_T	*ulRetrievalBuffer*	Retrieval buffer size allocated per retrieval request (the default is 2048 bytes).
ESS_ULONG_T	*ulRetrievalSortBuffer*	Retrieval sort buffer size allocated per retrieval request (the default is 10240 bytes).
ESS_BOOL_T	*bCacheMemLocking*	TRUE if index and data cache memory pages are locked into physical memory.
ESS_BOOL_T	*bPreImage*	Flag to read previously committed.
ESS_USHORT_T	*usIsolationLevel*	Isolation level (the default is UNCOMMITTED).
ESS_LONG_T	*lTimeOut*	Time out set in seconds for COMMITTED access only.
ESS_ULONG_T	*ulCommitBlocks*	The number of data blocks updated before the explicit commit is performed (during calculation and spreadsheet updates).
ESS_ULONG_T	*ulCommitRows*	The number of rows of the input file processed before the explicit commit is performed during the dataload.
ESS_ULONG_T	*ulDiskVolumeCount*	The number of disk volume settings for this database.
ESS_DISKVOLUME_T	*aDiskVolume[1]*	an array of disk volume settings

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;
```

Data Type	Value	Description
ESS_DBREQTYPE_DATLOAD	0	Data Load
ESS_DBREQTYPE_CALC	1	Calculation
ESS_DBREQTYPE_OTLUPD	2	Outline Update

The fields are:

Data Type	Field	Description
ESS_ULONG_T	*DbReqType*	Type of database request
ESS_USERNAME_T	*User*	User name
"ESS_TIMERECORD_T" on page 186	*StartTimeRec*	Request start time
"ESS_TIMERECORD_T" on page 186	*EndTimeRec*	Request end time
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_CALCDSCR (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;
```

```
   ESS_BOOL_T        CalcNoAggMissing;
   ESS_BOOL_T        CalcNoAvgMissing;
   ESS_BOOL_T        CalcTwoPass;
   ESS_BOOL_T        CalcCreateBlock;
   ESS_DBNAME_T      CrDbName;
   ESS_MBRNAME_T     CrTypeMember;
   ESS_USHORT_T      CrConvType;
   ESS_ULONG_T       MaxMemIndex;
   ESS_ULONG_T       IndexPageSize;
   ESS_BOOL_T        DataCompress;
   ESS_USHORT_T      DataCompressType;
   ESS_ULONG_T       RetrievalBuffer;
   ESS_ULONG_T       RetrievalSortBuffer;
   ESS_BYTE_T        cIOAccessFlagInUse;      /* new for 6.5
   ESS_BOOL_T        bNoWaitIO;               /* new for 6.5
   ESS_USHORT_T      IsolationLevel;
   ESS_BOOL_T        PreImage;
   ESS_BYTE_T        cIOAccessFlagPending;    /* new for 6.5
   ESS_LONG_T        TimeOut;
   ESS_ULONG_T       CommitBlocks;
   ESS_ULONG_T       CommitRows;
   ESS_ULONG_T       nVolumes;
   ESS_DISKVOLUME_T  DiskVolume[1];
} ESS_DBSTATE_T, *ESS_PDBSTATE_T, **ESS_PPDBSTATE_T;
```

The fields are:

Data Type	Field	Description
ESS_DESC_T	*Description*	The database description (up to 80 characters)
ESS_BOOL_T	*Loadable*	Flag to indicate whether the database can be loaded (ESS_TRUE if the database is loadable)
ESS_BOOL_T	*Autoload*	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)
ESS_ACCESS_T	*Access*	The default access level to the database. See “Bitmask Data Types (C)” on page 96 for a list of values this field can contain.
ESS_USHORT_T	*IndexType*	The database index type (array or tree). This field can contain the following values: ● ESS_INDEXTYPE_ARRAY ● ESS_INDEXTYPE_AVL For API releases 4 and later, the *IndexType* field is obsolete.
ESS_ULONG_T	*MaxMem*	The maximum memory reserved for non-compressed data blocks in the database(in bytes)
ESS_ULONG_T	*MaxCompMem*	The maximum memory reserved for compressed data blocks in database(in bytes)
ESS_BOOL_T	*CalcNoAggMissing*	Flag to suppress aggregation of members if all their children are missing (ESS_TRUE if missing values are not aggregated)
ESS_BOOL_T	*CalcNoAvgMissing*	Flag to suppress inclusion of missing members in calculating averages (ESS_TRUE if missing values are not included)

Data Type	Field	Description
ESS_BOOL_T	*CalcTwoPass*	Flag to force two pass calculation when running full calculation of database (ESS_TRUE if two pass calculation is enabled)
ESS_BOOL_T	*CalcCreateBlock*	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.
ESS_DBNAME_T	*CrDbName*	The name of associated currency database (valid in non-currency databases).
ESS_MBRNAME_T	*CrTypeMember*	The name of Currency Conversion type member (valid in non-currency databases)
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.
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	*cIOAccessFlagInUse*	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 cIOAccessFlagPending 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 cIOAccessFlag is ESS_IO_ACCESS_DIRECT. This field is read only. The default is TRUE.

Data Type	Field	Description
ESS_USHORT_T	*IsolationLevel*	The isolation level: ● COMMITTED—Write locks on all affected data blocks restrict access until the transaction commits. ● UNCOMMITTED(default)—Write locks are acquired and released as needed during the transaction.
ESS_BOOL_T	*PreImage*	The flag to read previously committed data during read-only requests. This flag can only be set for COMMITTED access. The default is YES.
ESS_BYTE_T	*cIOAccessFlagPending*	The type of I/O Access (direct or buffered) that Essbase will use. This setting takes effect after the next DBLoad (open operation).
ESS_LONG_T	*TimeOut*	The timeout interval in seconds. This can only be set for COMMITTED access. -1 is Indefinite wait. 0 is Immediate access, no wait (the default). *n* is the specified interval in seconds.
ESS_ULONG_T	*CommitBlocks*	The number of data blocks modified before performing the explicit commit (only used when isolation level is UNCOMMITTED).
ESS_ULONG_T	*CommitRows*	The number of rows of the input file to data load before performing the explicit commit (only used when isolation level is UNCOMMITTED).
ESS_ULONG_T	*nVolumes*	The number of disk volume settings for this database.
ESS_DISKVOLUME_T	*DiskVolume[1]*	An array of disk volume settings.

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:

```
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_DOUBLE_T      TotMemPagedInBlocks;
   ESS_DOUBLE_T      TotMemBlocks;
   ESS_DOUBLE_T      TotMemIndex;
   ESS_DOUBLE_T      TotMemInMemCompBlocks;
   ESS_DOUBLE_T      BlockDensity;
   ESS_DOUBLE_T      SparseDensity;
   ESS_DOUBLE_T      CompressionRatio;
   ESS_DOUBLE_T      ClusterRatio;
   ESS_DIMSTATS_T    DimStatsAry[1];
} ESS_DBSTATS_T, *ESS_PDBSTATS_T, **ESS_PPDBSTATS_T;
```

Data Type	Field	Description
ESS_USHORT_T	*IndexType*	The database index type (array or tree). This field can contain the following values: ● ESS_INDEXTYPE_ARRAY ● ESS_INDEXTYPE_AVL
ESS_ULONG_T	*nDims*	The number of dimensions in database.
ESS_ULONG_T	*DeclaredBlockSize*	The declared data block size.
ESS_ULONG_T	*ActualBlockSize*	The actual data block size
ESS_DOUBLE_T	*DeclaredMaxBlocks*	The declared maximum number of blocks in the database.
ESS_DOUBLE_T	*ActualMaxBlocks*	The actual maximum number of blocks in the database.
ESS_DOUBLE_T	*NonMissingLeafBlocks*	The number of non-missing leaf (lowest level) blocks in the database.
ESS_DOUBLE_T	*NonMissingNonLeafBlocks*	The number of non-missing, non-leaf (upper level) blocks in the database.
ESS_DOUBLE_T	*NonMissingBlocks*	Obsolete. Returns zero.
ESS_DOUBLE_T	*PagedOutBlocks*	The number of database blocks currently paged out to disk.
ESS_DOUBLE_T	*PagedInBlocks*	The total number of database blocks currently paged into memory.
ESS_DOUBLE_T	*InMemCompBlocks*	The number of database blocks currently paged into compressed memory.
ESS_DOUBLE_T	*TotalBlocks*	Total number of existing data blocks (not the maximum).
ESS_DOUBLE_T	*AverageFragmentationQuotient*	Percentage of space within the data file that is free space or not used by Essbase.
ESS_DOUBLE_T	*BytesOfRecoverableFreeSpace*	● Estimated bytes of recoverable free space ● -1 if free space recovery is not necessary
ESS_DOUBLE_T	*TotMemPagedIn-Blocks*	The total memory used for all paged-in (uncompressed) database blocks.
ESS_DOUBLE_T	*TotMemBlocks*	The total memory used for all database blocks.
ESS_DOUBLE_T	*TotMemIndex*	The total memory used for the database index.

Data Type	Field	Description
ESS_DOUBLE_T	*TotMemInMemCompBlocks*	The total memory used for database blocks currently paged into compressed memory.
ESS_DOUBLE_T	*BlockDensity*	The average database block density (calculated using all currently loaded blocks).
ESS_DOUBLE_T	*SparseDensity*	Average density of the sparse dimensions in the database.
ESS_DOUBLE_T	*CompressionRatio*	Average data block compression ratio on the disk.
ESS_DOUBLE_T	*ClusterRatio*	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.
"ESS_DIMSTATS_T" on page 133	*DimStatsAry [1]*	Dynamic array (with nDim elements) of Dimension Statistical Structures, of type ESS_DIMSTATS_T. See the structure definition.

ESS_DIMENSIONINFO_T

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

```
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;
```

Data Type	Field	Description
ESS_MBRNAME_T	*DimName*	Dimension name
ESS_DIMNUM_T	*DimNumber*	Dimension number
ESS_USHORT_T	*DimType*	Dimension type. Values: ● ESS_DIMTYPE_DENSE ● ESS_DIMTYPE_SPARSE

Data Type	Field	Description
ESS_USHORT_T	*DimTag*	Dimension tag type. Values: ● ESS_TTYPE_ATTRCALC ● ESS_TTYPE_ATTRIBUTE ● ESS_TTYPE_CCATEGORY ● ESS_TTYPE_CNAME ● ESS_TTYPE_CTIME ● ESS_TTYPE_NONE ● ESS_TTYPE_CPARTITION ● ESS_TTYPE_CTYPE
ESS_ULONG_T	*DeclaredDimSize*	Declared dimension size
ESS_ULONG_T	*ActualDimSize*	Actual dimension size
ESS_DESC_T	*Description*	Reserved (not currently supported)
ESS_USHORT_T	*DimDataType*	Attribute dimension data type. Values: ● ESS_ATTRMBRDT_BOOL ● ESS_ATTRMBRDT_DATETIME ● ESS_ATTRMBRDT_DOUBLE ● ESS_ATTRMBRDT_STRING

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:

```
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;
```

Data Type	Field	Description
ESS_MBRNAME_T	*DimName*	The dimension member name
ESS_USHORT_T	*DimType*	The dimension type (sparse or dense). This field can contain the following values: ● ESS_DIMTYPE_SPARSE ● ESS_DIMTYPE_DENSE
ESS_ULONG_T	*DeclaredDimSize*	The declared dimension size (the number of members declared in the specified dimension, including any label only or shared members in that dimension)

Data Type	Field	Description
ESS_ULONG_T	*ActualDimSize*	The actual dimension size (the number of members in the specified dimension, excluding any label only or shared members in that dimension)

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;
```

Data Type	Field	Description
ESS_LONG_T	*nColumnIdx*	0-based index of the column position (read only)
ESS_LONG_T	*nDisplayOrder*	The order in which columns are displayed
ESS_CHAR_T	*sViewName* [ESS_MBRNAMELEN]	(read only)
ESS_CHAR_T	*sColumnName* [ESS_MBRNAMELEN]	Heading text for the given column of data (read only)
ESS_USHORT_T	*uDataType*	Data type of the given column of data • ESS_DT_STRING • ESS_DT_DATETIME • ESS_DT_DOUBLE
ESS_LONG_T	*nSortOrder*	
ESS_LONG_T	*nSortSequence*	
ESS_BOOL_T	*bFilterOnly*	ESS_TRUE = filter only.
ESS_CHAR_T	*sFilter* [ESS_MAX_DATALEN + 1]	

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;
```

Data Type	Field	Description
ESS_ULONG_T	*nRowIdx*	0-indexed row number for the given data block
ESS_ULONG_T	*nColumnIdx*	0-indexed column number for the given data block
ESS_CHAR_T	*sData* [ESS_MAX_DATALEN + 1]	Data value for the given data block

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 ;
```

Data Type	Field	Description
ESS_ULONG_T	*nColumnIdx*	0-indexed column number for the given data block.
ESS_CHAR_T	*sViewName* [ESS_MBRNAMELEN]	
ESS_CHAR_T	*sColumnName* [ESS_DESCRIPTION_LEN + 1]	Data value for the given data block.
ESS_USHORT_T	*uDatatype*	

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;
```

Data Type	Field	Description
ESS_CHAR_T	*sHisName* [ESS_MAX_NAME + 1]	
ESS_CHAR_T	*sUsername* [ESS_MAX_NAME + 1]	
ESS_CHAR_T	*sPassword* [ESS_MAX_NAME + 1]	(write only)

Data Type	Field	Description
ESS_USHORT_T	*uInputOption*	(read only)

ESS_DTAPIREPORT_T

Defines the header information for a specific column.

```
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;
```

Data Type	Field	Description
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_DTBUFFER_T

Defines a report data cell.

```
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;
```

Data Type	Field	Description
ESS_ULONG_T	*row*	0-indexed row number for the given data block.
ESS_ULONG_T	*column*	0-indexed column number for the given data block.
ESS_CHAR_T	*data* [ESS_DESCRIPTION_LEN + 1]	Data value for the given data block.

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	Field	Description
ESS_ULONG_T	*row*	0-indexed row number for the given data block.
ESS_ULONG_T	*column*	0-indexed column number for the given data block.
ESS_CHAR_T	*data* [ESS_DESCRIPTION_LEN + 1]	Data value for the given data block.

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	Field	Description
ESS_ULONG_T	*colIndex*	0-based index of the column position.
ESSDTREPORTDATATYPE	*dataType*	Data type of the given column of data.
ESS_CHAR_T	*data* [ESS_DESCRIPTION_LEN + 1]	Heading text for the given column of data.

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	Field	Description
ESS_FILENAME_T	*szPartition_Src*	Name of disk partition to be replaced.
ESS_FILENAME_T	*szPartition_Dest*	Name of disk partition with which to replace `szPartition_Src`

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;
```

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

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,;
```

Data Type	Field	Description
ESS_USERNAME_T	*Name*	User name
ESS_APPNAME_T	*AppName*	Name of currently connected application (if applicable)
ESS_DBNAME_T	*DbName*	Name of the currently connected database (if applicable)
ESS_BOOL_T	*Login*	Flag to indicate login status.
ESS_USHORT_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_USHORT_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_BOOL_T	*PwdChgNow*	Flag that user must change password.
ESS_USHORT_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   PwStoredNum;

} 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.

Data Type	Field	Description
ESS_USHORT_T	*PwMin*	The minimum permitted password length (default is 6 characters).
ESS_TIME_T	*InactivityTime*	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.
ESS_TIME_T	*InactivityCheck*	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.
ESS_USHORT_T	*InvalidAttempts*	The number of invalid attempts allowed by a user before the system administrator is warned and the user is locked out.
ESS_USHORT_T	*InactivityLockout*	The duration of a period of inactivity (betweeen logins) for any user before that user is locked out (default is 365 days)
ESS_USHORT_T	*NumPwExpWarn*	The number of expired password warnings issued to a user before that user is locked out.
ESS_USHORT_T	*PwStoredNum*	The number of passwords stored for any user.

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_PCATCHFUNC_T,  CatchFunc;
   ESS_PCATCH_INIT_FUNC_T,  CatchInitFunc;
   ESS_PCATCH_TERM_FUNC_T,  CatchTermFunc;
   ESS_PCOOKIE_CREATE_FUNC_T, CookieCreateFunc;
   ESS_PCOOKIE_DELETE_FUNC_T, CookieDeleteFunc;
} ESS_INIT_T, *ESS_PINIT_T;
```

Data Type	Field	Description
ESS_ULONG_T	*Version*	Version of Essbase API used to compile the application. Should be set to ESS_API_VERSION. Used for backward compatibility.
ESS_PVOID_T	*UserContext*	An optional pointer to a user-defined message context (passed as argument to a user-defined MessageFunction)
ESS_USHORT_T	*MaxHandles*	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.
ESS_SIZE_T	*MaxBuffer*	The maximum size buffer that can be allocated in the client program (typically 64 KB). The default is 64 KB.
ESS_STR_T	*LocalPath*	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. .
ESS_STR_T	*MessageFile*	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.
ESS_PFUNC_T	*AllocFunc*	Pointer to the user-defined memory allocation function. All platforms: memory allocation functions use the malloc() function.
ESS_PFUNC_T	*ReallocFunc*	Pointer to the user-defined memory reallocation function. All platforms: memory allocation functions use the realloc() function.
ESS_PFUNC_T	*FreeFunc*	A pointer to the user-defined memory free function. All platforms: memory allocation functions use the free() function.
ESS_PFUNC_T	*MessageFunc*	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.
ESS_STR_T	*HelpFile*	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. See Chapter 3, "Integrating Essbase With Your Product." 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.
ESS_ULONG_T	*Ess_System*	Reserved for internal use. Set to NULL
ESS_USHORT_T	*usApiType*	Required. Defines whether the program is in Unicode or non-Unicode mode. For valid values, see "Unicode Mode Constants (C)" on page 107.
typedef ESS_ BOOL_T (*ESS_ PCATCHFUNC_T) (ESS_HCTX_T);	*CatchFunc*	If implemented by the client, Essbase calls this function intermittently (every few seconds) during queries. If the routine retuns TRUE, the API call gets canceled.

Data Type	Field	Description
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`.

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

ESS_BOOL_T KillReqCatcher(ESS_HCTX_T hCtx)
{
  KBDINFO_T      kbinfo;
   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*.

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

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.

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

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;
```

Data Type	Field	Description
ESS_ ULONG_T	*ulBufferId*	ID of a data load buffer (a number between 1 and 4294967296).
ESS_ ULONG_T	*ulDuplicateAggregationMethod*	One of the following constants describing how to combine 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. ● 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. ● ESS_ASO_DATA_LOAD_BUFFER_DUPLICATES_USE_LAST: Use the last value loaded into the buffer as the final value for the cell.
ESS_ ULONG_T	*ulOptionFlags*	Either (or a combination) of the following load buffer options: ● ESS_ASO_DATA_LOAD_BUFFER_IGNORE_MISSING_VALUES ● ESS_ASO_DATA_LOAD_BUFFER_IGNORE_ZERO_VALUES
ESS_ ULONG_T	*ulSize*	The percentage of the aggregate storage cache that the data load buffer is allowed to use (a number between 1 and 100, inclusive)
ESS_BOOL_T	*bInternal*	ESS_TRUE if the buffer was created by Essbase; ESS_FALSE for user-created buffers
ESS_BOOL_T	*bActive*	ESS_TRUE if the buffer is currently in use by a data load
ESS_BOOL_T	*bReserved01*	Not used
ESS_BOOL_T	*bReserved02*	Not used

Data Type	Field	Description
ESS_ ULONG_T	*ulReserved01*	Not used
ESS_ ULONG_T	*ulReserved02*	Not used
ESS_ ULONG_T	*ulReserved03*	Not used

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.

```
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;
```

Data Type	Field	Description
ESS_USERNAME_T	*UserName*	The user name
ESS_USHORT_T	*nLocks*	The number of blocks exclusively locked by this user
ESS_TIME_T	*Time*	The maximum time (in seconds) that blocks have been exclusively locked
ESS_LOGINID_T	*LoginId*	The user login identification tag

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.

```
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

Data Type	Field	Description
ESS_ULONG_T	*reserved0–2*	Reserved for future use

ESS_MBRALT_T

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

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

Data Type	Field	Description
ESS_MBRNAME_T	*MbrName*	The member name
ESS_MBRNAME_T	*AltName*	The associated alias name

ESS_MBRERR_T

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

```
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;
```

Data Type	Field	Description
struct ESS_MBRERR_T	**pNext*	Pointer to next structure in list
ESS_USHORT_T	*ErrType*	The type of error
ESS_STR_T	*Name*	The member name
ESS_STR_T	*Record*	The file record containing the error

ESS_MBRUSER_T

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

```
typedef struct ESS_MBRUSER_T
{
   ESS_STR_T User;
```

```
   ESS_STR_T Password;
} ESS_MBRUSER_T, *ESS_PMBRUSER_T;
```

Data Type	Field	Description
ESS_STR_T	*User*	The external data source user name
ESS_STR_T	*Password*	The external data source password

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            fHasHAEnabled;
} ESS_MEMBERINFO_T, *ESS_PMEMBERINFO_T, **ESS_PPMEMBERINFO_T;
```

Data Type	Field	Description
ESS_MBRNAME_T	*MbrName*	The member name
ESS_MEMNUM_T	*MbrNumber*	The member number in the database outline
ESS_MBRNAME_T	*DimName*	The member's dimension name
ESS_DIMNUM_T	*DimNumber*	The member's dimension number

Data Type	Field	Description
ESS_USHORT_T	*Status*	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 ● ESS_MBRSTS_xxx: ● ESS_MBRSTS_NOTSET ● ESS_MBRSTS_NEVER ● ESS_MBRSTS_LABEL ● ESS_MBRSTS_REFER ● ESS_MBRSTS_REFNME ● ESS_MBRSTS_SHARE ● ESS_MBRSTS_VIRTSTORE ● ESS_MBRSTS_VIRTNOSTORE
ESS_SHORT_T	*Level*	The member level number (zero-based), counting up from the lowest descendent of the specified member
ESS_SHORT_T	*Generation*	The member generation number (one-based), counting down from the specified member's dimension member

ESS_SHORT_T	*UnaryCalc*	The default unary rollup for this member. A value of the form ESS_UCALC_xxx (add, subtract, multiply, divide, percent, none, or never).
		`ESS_UCALC_ADD`
		`ESS_UCALC_SUB`
		`ESS_UCALC_MULT`
		`ESS_UCALC_DIV`
		`ESS_UCALC_PERCENT`
		`ESS_UCALC_NOOP`
		`ESS_UCALC_NEVER`

ESS_USHORT_T	*MbrTagType*	A 16 bit mask for the member's tagged types. A value of the form ESS_ATYPE_xxx.
ESS_BOOL_T	*CurrConvert*	Currency Conversion. Values: ESS_TRUE and ESS_FALSE
ESS_MBRNAME_T	*CrMbrName*	Name of the tagged currency database member. ● For Time dimension, gives the name of the tagged time member. ● For Country dimension, gives the name of the tagged currency member. ● For Accounts dimension, gives the name of the tagged category member.
ESS_DESC_T	*Description*	Member description
ESS_MBRNAME_T	*ParentMbrName*	Specified member's parent member name or empty string if member has no parent
ESS_MBRNAME_T	*ChildMbrName*	Specified member's first child member name
ESS_MBRNAME_T	*PrevMbrName*	Specified member's previous sibling member name
ESS_MBRNAME_T	*NextMbrName*	Specified member's next sibling member name
ESS_BOOL_T	*fAttributed*	Indicates whether the member has attributes associated with it. Values: ESS_TRUE and ESS_FALSE.

Data Type	Field	Description
"ESS_ATTRIBUTEVALUE_T" on page 118	*Attribute*	Attribute value
ESS_BOOL_T	*fHasRelDesc*	The member has one or more relational children.
ESS_BOOL_T	*fHasHAEnabled*	The dimension has Hybrid Analysis relational partitions enabled. Valid only for Dimension members.

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:

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

Data Type	Field	Description
ESS_USERNAME_T	*Name*	User or group name.
ESS_PASSWORD_T	*Password*	User password.

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.

```
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;
```

Data Type	Field	Description
ESS_HCTX_T	*hCtx*	Object context handle
ESS_OBJTYPE_T	*ObjType*	Object type. See "Bitmask Data Types (C)" on page 96 for a list of object types.
ESS_STR_T	*AppName*	Application name

Data Type	Field	Description
ESS_STR_T	*DbName*	Database name
ESS_STR_T	*FileName*	8-character object file name with no extension. This name is a local file name when all of the following apply: ● *hCtx* is a local context handle ● *AppName* and *DbName* are NULL ● *FileName* points to the full path name of a local file

ESS_OBJINFO_T

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

```
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_TIMERECORD_T TimeModified;
} ESS_OBJINFO_T, *ESS_POBJINFO_T, **ESS_PPOBJINFO_T;
```

Data Type	Field	Description
ESS_OBJNAME_T	*Name*	Object name
ESS_OBJTYPE_T	*Type*	Object type. See "Bitmask Data Types (C)" on page 96 for a list of object types.
ESS_APPNAME_T	*AppName*	Application name
ESS_DBNAME_T	*DbName*	Database name
ESS_ULONG_T	*FileSize*	Object's allocated file size on disk (in bytes)
ESS_BOOL_T	*Locked*	Flag to indicate whether object is locked on the server (ESS_TRUE indicates the object is locked)
ESS_USERNAME_T	*User*	Name of the user who has the object locked (if locked), otherwise undefined
ESS_TIME_T	*TimeStamp*	Date and time object was locked (if locked), otherwise undefined
"ESS_TIMERECORD_T" on page 186	*TimeModified*	Date and time of last modification

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;
```

Data Type	Field	Description
"ESS_PARTHDR_T" on page 159	*file_header*	File header.
ESS_USHORT_T	*partition_count*	Number of shared partitions.
"ESS_PARTDEF_T" on page 157	*partitions*	Array of shared partition definitions.
ESS_ULONG_T	*maxserialno*	High water mark for serial number.

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;
```

Data Type	Field	Description
ESS_STR_T	*pszHostName*	Host name.
ESS_STR_T	*pszAppName*	Application name.
ESS_STR_T	*pszDbName*	Database name.

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;
```

Data Type	Field	Description
ESS_USHORT_T	*usType*	One of the Operation Type constants listed below.
ESS_USHORT_T	*usDirection*	One of the Directions constants listed below.

Data Type	Field	Description
"ESS_PART_CONNECT_INFO_T" on page 153	*HostDatabase*	The host server.

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_INFO_T

Holds the multicube shared partition information.

```
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.

Data Type	Field	Description
The following fields only apply to replication data targets		
ESS_TIME_T	*LastRefreshTime*	Last time data at target was refreshed.
ESS_BOOL_T	*partitionUpdatable*	Are changes allowed to replicated data?
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.

```
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;
```

Data Type	Field	Description
ESS_LONG_T	*lPartitionCount*	Number of partitions to refresh from (-1 == ALL)
ESS_BOOL_T	*bUpdatedOnly*	Refreshes only the cells modified at the source since the last refresh operation.
"ESS_PART_CONNECT_INFO_T" on page 153	*pHostDatabase*	Array of partition specifications.

ESS_PARTDEF_INVALID_T

This is the shared partition verification structure.

```
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;
```

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

Error Constants

```
define ESS_PARTITION_DEF_ERROR           = 1
define ESS_PARTITION_GLOBAL_MAP_ERROR    = 2
define ESS_PARTITION_AREA_MAP_ERROR      = 3
define ESS_PARITITON_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;
```

Data Type	Field	Description
ESS_CHAR_T	*svrname [ESS_SVRNAMELEN]*	Server name.

Data Type	Field	Description
ESS_CHAR_T	*appname [ESS_APPNAMELEN]*	Application name.
ESS_CHAR_T	*dbname [ESS_DBNAMELEN]*	Database name.
ESS_CHAR_T	*username [ESS_USERNAMELEN]*	Administrator username.
ESS_CHAR_T	*password [ESS_PASSWORDLEN]*	Administrator password.

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;
```

Data Type	Field	Description
ESS_ULONG_T	*mbr_count*	Size of remapping arrays.
ESS_STR_T	*src_mbrs*	Array of member names at src.
ESS_STR_T	*dest_mbrs*	Array of member names at target.

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;
```

Data Type	Field	Description
"ESS_PARTDEF_CONNECT_T" on page 156	*connection*	Connection information.
ESS_STR_T	*description*	User's description of partition.
"ESS_PARTDEF_AREAS_T" on page 158	*shape_defn*	Shape definition.
"ESS_PARTDEF_TYPE_T" on page 158	*typedata*	Type-specific data.

Data Type	Field	Description
ESS_ULONG_T	*serialno*	1-based ID for shared partitions.
ESS_TIME_T	*meta_last_updated*	Last restructure affecting this partition.

ESS_PARTDEF_AREAS_T

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

```
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;
```

Data Type	Field	Description
ESS_USHORT_T	*slice_count*	Number of slices.
ESS_STR_T	*slices*	Array of slice definition strings.

ESS_PARTDEF_TYPE_T

Holds partition type-specific information.

```
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;
```

Data Type	Field	Description
ESS_USHORT_T	*operation_type*	One of the Operation Type constants listed below.
ESS_USHORT_T	*direction_type*	One of the Direction constants listed below. **Note:** Fields marked as SVR: should only be modified by server code.
ESS_USHORT_T	*meta_direction_type*	Source of metadata identified by one of the Direction constants listed below.

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

```
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_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;
```

Data Type	Field	Description
ESS_SVRNAME_T	*zServer*	The server name.
ESS_APPNAME_T	*zApplication*	The application name.

Data Type	Field	Description
ESS_DBNAME_T	*zDatabase*	The database name.
ESS_USERNAME_T	*zUser*	The user name.
ESS_TIME_T	*tTime*	Last restructure affecting this partition.

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;
```

Data Type	Field	Description
ESS_ULONG_T	*ulDimensionCount*	Number of dimension changes.
"ESS_PARTOTL_DIMCHG_API_T" on page 162	*pDimchg*	Pointer to a link list of dimension changes.
ESS_ULONG_T	*ulAliasTableCount*	Count of alias table changes.
"ESS_PARTOTL_NAMEMAP_API_T" on page 167	*pAliasTableChg*	Linked list of table changes.

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;
```

Data Type	Field	Description
ESS_USHORT_T	*usFileNum*	Number of meta change files.
ESS_PSTR_T	*ppszFileName*	Array of meta change file names.

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;
```

Data Type	Field	Description
ESS_USHORT_T	*usDimType*	One of the Dimension Type constants listed below.
ESS_USHORT_T	*usDimTag*	One of the Dimension Tag constants listed as type ESS_TTYPE_XXX below.
ESS_ULONG_T	*ulOldDimNo*	The dimension number in the old outline.
ESS_ULONG_T	*ulNewDimNo*	The dimension number in the new outline.
ESS_ULONG_T	*ulNamedLevNum*	The number of named levels.
"ESS_PARTOTL_NAMED_GENLEV_API_T" on page 167	*pNamedLev*	The pointer to an array of named level structures.
ESS_ULONG_T	*ulNamedGenNum*	The number of named generations.
"ESS_PARTOTL_NAMED_GENLEV_API_T" on page 167	*pNamedGen*	The pointer to an array of named generations structures.
ESS_STR_T	*pszBasememberName*	The base member name for the add and delete dimensions.

Data Type	Field	Description
ESS_STR_T	*pszOldName*	The old dimension name.
ESS_STR_T	*pszNewName*	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

Dimension Type Constants (usDimType)

```
define ESS_DIMTYPE_DENSE  0
define ESS_DIMTYPE_SPARSE 1
```

Dimension Tag Constants (usDimTag)

```
#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.

```
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;
```

Data Type	Field	Description
ESS_SHORT_T	*usDimAssocChgType*	Dimension association change type
ESS_CHAR_T	*pszAttrDimName*	Attribute dimension name
ESS_SHORT_T	*usLevel*	Dimension association level
ESS_PARTOTL_DIMASSOCCHG_API_T	*pNext*	Pointer to the next structure

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;
```

Data Type	Field	Description
ESS_USHORT_T	*usDimChgType*	One of the dimension change (ESS_OTL_DIMCHG_T) constants listed below
"ESS_PARTOTL_DIM_ATTRIB_API_T" on page 161	*DimAttribute*	Dimension attributes
"ESS_PARTOTL_MBR_RSRVD_API_T" on page 164	*MemberReserved*	Reserved
The following two fields are only valid when ESS_PARTITION_OTLDIM_MBRCHG is one of the dimension change types.		
ESS_ULONG_T	*ulMemberChanges*	Number of member changes
"ESS_PARTOTL_MBRCHG_API_T" on page 165	*pMemberChange*	Pointer to the linked list of member changes
ESS_USHORT_T	*usAttrType*	Attribute type
ESS_USHORT_T	*usDimAssocChgCnt*	Number of dimension associations
"ESS_PARTOTL_DIMASSOCCHG_API_T" on page 162	*pDimAssocChg*	Linked list of dimension associations
ESS_PARTOTL_DIMCHG_API_T	*pNext*	Pointer to the next dimension change

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_OTLDIM_ADD     /* Add dimensions    */
ESS_PARTITION_OTLDIM_DELETE  /* Delete dimensions */
ESS_PARTITION_OTLDIM_UPDATE  /* Update dimensions */
ESS_PARTITION_OTLDIM_MOVE    /* Move dimensions   */
ESS_PARTITION_OTLDIM_RENAME  /* Rename dimensions */
ESS_PARTITION_OTLDIM_MBRCHG  /*                   */
ESS_PARTITION_OTLDIM_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;
```

Data Type	Field	Description
ESS_BOOL_T	*breject*	TRUE rejects this record (for Outline Synchronization only).
"ESS_PARTOTL_OSN_RELATIVES_API_T" on page 168	**pSrcRelatives*	Source parent and sibling
"ESS_PARTOTL_OSN_RELATIVES_API_T" on page 168	**pDstRelatives*	Destination parent and sibling
ESS_VOID_T	**unused*	(Unused)

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;
```

Data Type	Field	Description
ESS_CHAR_T	*pszAttrDimName*	Attribute dimension name
ESS_CHAR_T	*pszAttrMbrName*	Attribute member name
ESS_CHAR_T	*pszAttrParName*	Attribute parent name
"ESS_ATTRIBUTEVALUE_T" on page 118	*AttrValue*	Attribute value
ESS_PARTOTL_MBRASSOCCHG_API_T	*pNext*	Pointer to the next structure

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;
```

Data Type	Field	Description
ESS_STS_T	*status*	Member status.
ESS_SHORT_T	*level*	Level number.
ESS_SHORT_T	*generation*	Generation.
ESS_CHAR_T	*calc*	Calculation equation.
ESS_SHORT_T	*ucalc*	Unary calculation symbol for this member.
ESS_USHORT_T	*atype*	A 16 bit mask for members of the dimension tagged as ACCOUNT. This is not used elsewhere. By default, they are all OFF.
ESS_BOOL_T	*nocconvert*	Default to FALSE, do currency conversion.
ESS_CHAR_T	*crMbrName*	The name of the tagged currency database member FOR TIME -- tagged Time Member FOR COUNTRY -- tagged currency Member FOR ACCOUNTS -- tagged category Member
"ESS_PARTOTL_NAMECHG_API_T" on page 167	*pUdaChange*	User defined attributes changes.
"ESS_PARTOTL_NAMECHG_API_T" on page 167	*pAliasChange*	Alias changes.

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;
```

Data Type	Field	Description
ESS_ULONG_T	*ulOperator*	One of the member change (ESS_MBR_CHANGE_T) constants listed below
ESS_CHAR_T	*pszOperand1*	Alphabetic operand 1
ESS_CHAR_T	*pszOperand2*	Alphabetic operand 2
ESS_CHAR_T	*pszOperand3*	Alphabetic operand 3
ESS_CHAR_T	*pszOperand4*	Alphabetic operand 4
ESS_ULONG_T	*ulOperand1*	A bit-field operand that indicates updated attributes of the given member. This field is only used when the member change operator is ESS_PARTITION_OTLMBR_UPDATE.
"ESS_PARTOTL_MBRATTR_API_T" on page 164	*pMemberAttribute*	The pointer to a member attribute structure. The value is null for delete and rename.
"ESS_PARTOTL_MBR_RSRVD_API_T" on page 164	*MemberReserved*	Reserved
ESS_ULONG_T	*ulMbrAssocChgCnt*	Number of member associations
"ESS_PARTOTL_MBRASSOCCHG_API_T" on page 164	*pMbrAssocChg*	Linked list of member associations
ESS_PARTOTL_MBRCHG_API_T	*pNext*	Pointer to the next structure

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_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_CCONVERT     /* Currency conversion flag        */
ESS_PARTITION_OTLMBRATTR_CRMBRNAME    /* Tagged currency database member */
ESS_PARTITION_OTLMBRATTR_UDA          /* 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                */
```

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;
```

Data Type	Field	Description
ESS_USHORT_T	*usCount*	The number of changes.
"ESS_PARTOTL_NAMEMAP_API_T" on page 167	*pNameMap*	Array of name maps.

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;
```

Data Type	Field	Description
ESS_USHORT_T	*usOperator*	One of the Name Operation Type constants listed below.
ESS_SHORT_T	*sGenLev*	Generation or Level number.
ESS_STR_T	*pszName*	Generation or Level name.
ESS_PARTOTL_NAMED_GENLEV_API_T	*pNext*	Pointer to the next structure.

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;
```

Data Type	Field	Description
ESS_USHORT_T	*usOperator*	One of the Name Operation Type constants listed below.
ESS_CHAR_T	*name*	Name of uda, for alias changes, the alias table name.
ESS_CHAR_T	*name2*	Not used for uda changes, for alias changes, use the alias name.
ESS__PARTOTL_NAMEMAP_API_T	*pNext*	Pointer to the next structure.

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;
```

Data Type	Field	Description
ESS_UCHAR_T	*statuses*	An array containing the status of each relative
ESS_PCHAR_T	*names*	An array containing the name of each relative
"ESS_ATTRIBUTEVALUE_T" on page 118	*values*	An array containing the attribute value structure for each relative

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
```

```
ESS_PARTOTL_OSN_RESERVED2
ESS_PARTOTL_OSN_NUM_RELATIVES
}
#define ESS_PARTOTL_OSN_REGION_LSIBLING ESS_PARTOTL_OSN_GENER_REGION_LSIBLING
#define ESS_PARTOTL_OSN_REGION_RSIBLING ESS_PARTOTL_OSN_GENER_REGION_RSIBLING


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;
```

Data Type	Field	Description
"ESS_PART_CONNECT_INFO_T" on page 153	*HostDatabase*	The host database.
ESS_USHORT_T	*usOperationType*	One of the Operation Type constants listed below.
ESS_USHORT_T	*usDataDirectionType*	One of the Direction Type constants listed below.
"ESS_PARTOTL_QRY_FILTER_T" on page 170	*MetaFilter*	Criteria to further define names.

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.

```
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;
```

Data Type	Field	Description
ESS_TIME_T	*TimeStamp*	Query meta change happens after this time.
ESS_ULONG_T	*ulDimFilter*	Bitfield to select dimension changes.
ESS_ULONG_T	*ulMbrFilter*	Bitfield to select member changes.
ESS_ULONG_T	*ulMbrAttrFilter*	Bitfield to select member attribute changes.

Member Attribute Change Constants

```
#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)
#define ESS_ALLCHG      (ESS_PARTITION_OTLMBR_ALL | ESS_DIMCHG_ALL)
```

ESS_PARTOTL_READ_T

Reads metadata changes.

```
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;
```

Data Type	Field	Description
"ESS_PARTOTL_CHANGE_API_T" on page 160	*pOtlChg*	Meta change records.
ESS_TIME_T	*SourceTime*	Time when source outline is changed.

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;
```

Data Type	Field	Description
ESS_STR_T	*pszFileName*	Outline change file name.
"ESS_PARTOTL_CHANGE_API_T" on page 160	*pOtlChg*	Outline change records (from `EssPartitionReadOtlChangeFile`).
ESS_TIME_T	*SourceTime*	Timestamp from outline change source.

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;
```

Data Type	Field	Description
ESS_STR_T	*pszFileName*	Meta change file name.
"ESS_PARTOTL_QRY_FILTER_T" on page 170	*QueryFilter*	Only reads records which satisfy the criteria.

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       usMetaDirectionTypes;
} ESS_PARTSLCT_T, *ESS_PPARTSLCT_T, **ESS_PPPARTSLCT_T;
```

Data Type	Field	Description
ESS_USHORT_T	*usOperationTypes*	One of the Operation Type constants listed below.
ESS_USHORT_T	*usDirectionTypes*	One of the Direction constants listed below.

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_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;
```

Data Type	Field	Description
ESS_USHORT_T	*usLoc*	Either ESS_FILE_CLIENT or ESS_FILE_SERVER
ESS_STR_T	*pszFileName*	Partition definition file name.
"ESS_PART_DEFINED_T" on page 153	*Partition*	Partition to verify.

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,
```

```
   ESS_PERF_ALLOC_ARG_AMOUNTCONTEXT,3,
   ESS_PERF_ALLOC_ARG_AMOUNTTIMESPAN,4,
   ESS_PERF_ALLOC_ARG_TARGET,5,
   ESS_PERF_ALLOC_ARG_TARGETTIMESPAN,6,
   ESS_PERF_ALLOC_ARG_TARGETTIMESPANOPTION,7,
   ESS_PERF_ALLOC_ARG_OFFSET,8,
   ESS_PERF_ALLOC_ARG_DEBITMEMBER,9,
   ESS_PERF_ALLOC_ARG_CREDITMEMBER,10,
   ESS_PERF_ALLOC_ARG_RANGE,11,
   ESS_PERF_ALLOC_ARG_EXCLUDEDRANGE,12,
   ESS_PERF_ALLOC_ARG_BASIS,13,
   ESS_PERF_ALLOC_ARG_BASISTIMESPAN,14,
   ESS_PERF_ALLOC_ARG_BASISTIMESPANOPTION,15,
   ESS_PERF_ALLOC_ARG_ALLOCATIONMETHOD,16,
   ESS_PERF_ALLOC_ARG_SPREADSKIPOPTION,17,
   ESS_PERF_ALLOC_ARG_ZEROAMOUNTOPTION,18,
   ESS_PERF_ALLOC_ARG_ZEROBASISOPTION,19,
   ESS_PERF_ALLOC_ARG_NEGATIVEBASISOPTION,20,
   ESS_PERF_ALLOC_ARG_ROUNDMETHOD,21,
   ESS_PERF_ALLOC_ARG_ROUNDDIGITS,22,
   ESS_PERF_ALLOC_ARG_ROUNDTOLOCATION,23,
   ESS_PERF_ALLOC_ARG_SCRIPT,24,
   ESS_PERF_ALLOC_ARG_SOURCEREGION,25,
   ESS_PERF_ALLOC_ARG_GROUPID,26,
   ESS_PERF_ALLOC_ARG_RULEID,27
} ESS_PERF_ALLOC_ARG_T;
```

ESS_PERF_ALLOC_ERROR_T

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;
```

Data Type	Field	Description
ESS_PERF_ALLOC_ERROR_T	*nextError*	Pointer to the next error structure, if any
ESS_ULONG_T	*messageNumber*	The number of the corresponding error or warning message
ESS_PERF_ALLOC_ARG_T	*argument*	Indicates which parameter contains the error
ESS_ULONG_T	*lineNumber*	Indicates which line of the argument contains the error. If zero, this is not applicable.

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;
```

Data Type	Field	Description
ESS_STR_T	*pov*	MDX set expression specifying allocation area within the database
ESS_STR_T	*amount*	MDX tuple or numeric value expression specifying amount or amounts to be allocated
ESS_STR_T	*amountContext*	**Optional:** MDX tuple expression: ● If *amount* is a numeric value expression, specifies context for amount ● If *amount* is a tuple or constant, this argument is empty
ESS_STR_T	*amountTimeSpan*	**Optional:** MDX set expression of level 0 members specifying time periods from which *amount* is summed before allocation
ESS_STR_T	*target*	MDX tuple expression specifying target locations for allocation
ESS_STR_T	*targetTimeSpan*	**Optional:** MDX set expression specifying time periods for target; used with *targetTimeSpanOption*

Data Type	Field	Description
ESS_ALLOCATION_TARGETTIMESPAN_OPTION_T	*targetTimeSpanOption*	**Optional:** Specifies how values are allocated to *targetTimeSpan* members: ● ESS_ASO_ALLOCATION_TIMESPAN_DIVIDEAMT (divide) ● ESS_ASO_ALLOCATION_TIMESPAN_REPEATAMT (repeat) ● Ignored if empty
ESS_STR_T	*offset*	**Optional:** MDX tuple expression specifying location for offsetting entries
ESS_STR_T	*debitMember*	**Optional:** MDX member expression specifying where positive result values should be written. If empty, debit/credit processing is not performed.
ESS_STR_T	*creditMember*	**Optional:** MDX member expression specifying where negative result values should be written. If empty, debit/credit processing is not performed.
ESS_STR_T	*range*	MDX set expression specifying database region for allocation
ESS_STR_T	*excludedRange*	**Optional:** MDX set expression specifying a subset of *range*; a region included in the allocation but not written to
ESS_STR_T	*basis*	MDX tuple expression specifying the basis location. If *allocationMethod* = ESS_ASO_ALLOCATION_METHOD_SPREAD and *spreadSkipOptions* = 0, then *basis* must be empty.
ESS_STR_T	*basisTimeSpan*	**Optional:** MDX set expression specifying time periods to be considered with *basis*. With *basisTimeSpanOption*, determines basis for allocation.
ESS_ALLOCATION_BASISTIMESPAN_OPTION_T	*basisTimeSpanOption*	**Optional:** Specifies how basis is computed across time periods from the following options: ● ESS_ASO_ALLOCATION_TIMESPAN_SPLITBASIS—Process basis value for each time period individually ● ESS_ASO_ALLOCATION_TIMESPAN_COMBINEBASIS—Sum basis value across time periods in *basisTimeSpan* and use the combined basis for allocation
ESS_ALLOCATION_METHOD_OPTION_T	*allocationMethod*	Allocation method: ● ESS_ASO_ALLOCATION_METHOD_SHARE—allocate proportional to basis values ● ESS_ASO_ALLOCATION_METHOD_SPREAD—allocate evenly across the target region

Data Type	Field	Description
ESS_ULONG_T	*spreadSkipOption*	**Optional:** ● If *allocationMethod* = ESS_ASO_ALLOCATION_METHOD_SHARE, then this value equals 0. ● If *allocationMethod* = ESS_ASO_ALLOCATION_METHOD_SPREAD, specifies which basis values should be skipped. Select one or more of the following bitwise arguments: ❍ ESS_ASO_ALLOCATION_SPREAD_SKIPMISSING—Excludes all cells in *allocationRange* for which the basis member is #missing ❍ ESS_ASO_ALLOCATION_SPREAD_SKIPZERO—Excludes all cells in allocationRange for which the basisMbr is zero ❍ ESS_ASO_ALLOCATION_SPREAD_SKIPNEGATIVE—Excludes all cells in allocationRange for which the basisMbr is negative These arguments can be combined bitwise; for example ESS_ASO_ALLOCATION_SPREAD_SKIPZERO | ESS_ASO_ALLOCATION_SPREAD_SKIPNEGATIVE
ESS_ALLOCATION_ZEROAMT_OPTION_T	*zeroAmountOption*	Specifies what to do when an *amount* value is zero or #MISSING: ● ESS_ASO_ALLOCATION_ZEROAMT_DEFAULT—Allocate zero values ● ESS_ASO_ALLOCATION_ZEROAMT_NEXTAMT—Skip to the next *amount* value ● ESS_ASO_ALLOCATION_ZEROAMT_ABORT—Cancel the entire allocation
ESS_ALLOCATION_ZEROBASIS_OPTION_T	*zeroBasisOption*	● If *allocationMethod*=ESS_ASO_ALLOCATION_METHOD_SHARE—Tells Essbase what to do when the aggregate sum of *basis* is zero ● If *allocationMethod*=ESS_ASO_ALLOCATION_METHOD_SPREAD—Tells Essbase what to do when all *basis* values have been skipped Specify an option: ● ESS_ASO_ALLOCATION_ZEROBASIS_NEXTAMT—Skip to the next *amount* value ● ESS_ASO_ALLOCATION_ZEROBASIS_ABORT—Cancel the allocation
ESS_ALLOCATION_NEGBASIS_OPTION_T	*negativeBasisOption*	Tells Essbase what to do when a negative *basis* value is encountered: ● ESS_ASO_ALLOCATION_NEGBASIS_DEFAULT—Calculate as normal ● ESS_ASO_ALLOCATION_NEGBASIS_NEXTAMT—Skip to next *amount* value. No data is allocated for the current *amount* value. ● ESS_ASO_ALLOCATION_NEGBASIS_ABORT—Cancel the allocation; no data is written. The following values are only valid when *allocationMethod*==ESS_ASO_ALLOCATION_METHOD_SHARE ● ESS_ASO_ALLOCATION_NEGBASIS_ABS—Use the absolute value ● ESS_ASO_ALLOCATION_NEGBASIS_MISSING—Treat the basis as #missing ● ESS_ASO_ALLOCATION_NEGBASIS_ZERO—Treat the basis value as zero

Data Type	Field	Description
ESS_ALLOCATION_ROUND_OPTION_T	*roundMethod*	Rounding method for allocated values: ● ESS_ASO_ALLOCATION_ROUND_NONE—Perform no rounding ● ESS_ASO_ALLOCATION_ROUND_DISCARDERRORS—Round, discarding rounding errors ● ESS_ASO_ALLOCATION_ROUND_ERRORSTOHIGHEST—Round, adding rounding errors to the target cell with the greatest allocated value ● ESS_ASO_ALLOCATION_ROUND_ERRORSTOLOWEST—Round, adding rounding errors to the target cell with the lowest allocated value ● ESS_ASO_ALLOCATION_ROUND_ERRORSTOLOCATION—Round, adding rounding errors to *roundToLocation*
ESS_STR_T	*roundDigits*	Must be empty if *roundMethod*=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.
ESS_STR_T	*roundToLocation*	**Optional:** If *roundMethod*=ESS_ASO_ALLOCATION_ROUND_ERRORSTOLOCATION, this is an MDX tuple expression specifying a location within *range*; empty otherwise
ESS_ULONG64_T	*groupID*	Internal use only. Always enter 0.
ESS_ULONG64_T	*ruleID*	Internal use only. Always enter 0.

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;
```

Data Type	Field	Description
ESS_STR_T	*pov*	MDX set expression specifying script execution area within the database

Data Type	Field	Description
ESS_STR_T	*script*	Contents of the custom calculation script. Should include multiple assignments of the form `Target := Formula;` where *Target* is an MDX tuple expression and *Formula* is an MDX numeric value expression. The script can contain only MDX tuple expressions and arithmetic operators (+ , - , * , /). MDX functions are not supported.
ESS_STR_T	*target*	**Optional:** MDX tuple expression specifying location, in combination with *pov* and the left-hand side of the assignment statements in *script*, to which calculation results will be written. If dimensions overlap , the order for resolving conflicts is assignment statements first, then *target*, then *pov*.
ESS_STR_T	*debitMember*	**Optional:** MDX member expression specifying the debit member. Positive results are stored here. If empty, debit/credit processing is not performed.
ESS_STR_T	*creditMember*	**Optional:** MDX member expression specifying the credit member. Negative results will be stored here. If empty, debit/credit processing is not performed.
ESS_STR_T	*offset*	**Optional:** MDX tuple expression specifying the location for offset entries, if any, to be written
ESS_STR_T	*sourceRegion*	MDX set expression indicating the database region referred to by the right-hand sides of the formulas in the script
ESS_ULONG64_T	*groupID*	Internal use only. Always enter 0.
ESS_ULONG64_T	*ruleID*	Internal use only. Always enter 0.

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;
```

Data Type	Field	Description
ESS_USHORT_T	*Action*	Current process action (not used)

Data Type	Field	Description
ESS_USHORT_T	*Stat*	Current process state (either done or in progress). Values: ● ESS_STATE_DONE (0) ● ESS_STATE_INPROGRESS (1) ● ESS_STATE_FINALSTAGE (5)
ESS_USHORT_T	*Reserved1*	Reserved for future use
ESS_ULONG_T	*Reserved2*	Reserved for future use
ESS_ULONG_T	*Reserved3*	Reserved for future use

ESS_RATEINFO_T

This currency partition rate information structure is used by **EssGetCurrencyRateInfo**(). The fields in this structure cannot be modified by the API.

```
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;
```

Data Type	Field	Description
ESS_MBRNAME_T	*MbrName*	Member name
ESS_MBRNAME_T	*RateMbr[ESS_CRDB_MAXDIMNUM]*	Array of rate member names

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.

```
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;
```

Data Type	Field	Description
ESS_LOGINID_T	*LoginId*	A unique number assigned to the user when the user logs in.
ESS_USERNAME_T	*UserName*	The name of the requesting user.
ESS_SVRNAME_T	*LoginSourceMachine*	Server name from which the session or request is being made
ESS_APPNAME_T	*AppName*	The active application (if any) for the session or request
ESS_DBNAME_T	*DbName*	The active database (if any) for the session or request
ESS_USHORT_T	*DbRequestCode*	A positive integer representing an active session. Example: 774896669
ESS_DESC_T	*RequestString*	A string representing the type of request. For possible values, see Request Types below.
ESS_TIME_T	*TimeStarted*	how long the session or request has been in progress (in seconds)
"ESS_REQ_STATE_T" on page 185	*State*	The state of the current session or request: whether it is processing, terminating, or terminated.

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_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;
```

```
   ESS_TIME_T      TimeStarted;
   ESS_REQ_STATE_T State;
}  ESS_REQUESTINFOEX_T, *ESS_PREQUESTINFOEX_T, **ESS_PPREQUESTINFOEX_T;
```

Data Type	Field	Description
ESS_LOGINID_T	*LoginId*	A unique number assigned to the user when the user logs in
ESS_USERNAME_T	*UserName*	Name of the requesting user
ESS_USERNAME_T	*ProviderName*	Name of the user directory. Example: `@Native Directory`
ESS_CONNPARAM_T	*connparam*	Unique identity attribute identifying a user or group in a directory. Example: `native://nvid=f0ed2a6d7fb07688:5a342200:1265973105c:-7f46?USER`
ESS_SVRNAME_T	*LoginSourceMachine*	Server name from which the session or request is being made
ESS_APPNAME_T	*AppName*	Active application (if any) for the session or request
ESS_DBNAME_T	*DbName*	Active database (if any) for the session or request
ESS_USHORT_T	*DbRequestCode*	A positive integer representing an active session. Example: 774896669
ESS_DESC_T	*RequestString*	A string representing the type of request. For possible values, see "ESS_REQUESTINFOEX_T" on page 182.
ESS_TIME_T	*TimeStarted*	How long the session or request has been in progress (in seconds)
ESS_REQ_STATE_T	*State;*	State of the current session or request: whether it is processing, terminating, or terminated.

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
```

Data Type	Field	Description
ESS_USHORT_T	ESS_REQ_IN_PROGRESS (0)	The current session or request is processing.
ESS_USHORT_T	ESS_REQ_TERMINATING (1)	The current session or request is terminating.
ESS_USHORT_T	ESS_REQ_TERMINATED (2)	The current session or request is terminated.

ESS_SECURITY_MODE_T

Used by EssGetEssbaseSecurityMode. This data type returns information about the security mode of Essbase Server.

```
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.

```
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;
```

Data Type	Field	Description
ESS_ULONG_T	*seq_id_start*	Start of range
ESS_ULONG_T	*seq_id_upper_start*	Upper start of range
ESS_ULONG_T	*seq_id_end*	End of range
ESS_ULONG_T	*seq_id_upper_end*	Upper end of range

ESS_TIMERECORD_T

```
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_DBREQINFO_T" on page 126 structure. The times expressed in this structure are usually server times. The fields are:

Data Type	Field	Description
ESS_TIME_T	*TimeValue*	Time value in seconds after 1/1/70
ESS_USHORT_T	*Seconds*	Seconds after the minute. Values: 0-59.
ESS_USHORT_T	*Minutes*	Minutes after the hour. Values: 0-59.
ESS_USHORT_T	*Hours*	Hours since midnight. Values: 0-23.
ESS_USHORT_T	*Day*	Day of the month. Values: 1-31.
ESS_USHORT_T	*Month*	Months since January. Values: 0-11. January = 0.
ESS_USHORT_T	*Year*	Years since 1900.
ESS_USHORT_T	*Weekday*	Days since Sunday. Values: 0-6. Sunday = 0.

ESS_TRANSACTION_ENTRY_T

Contains

```
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
```

Data Type	Field	Description
ESS_ULONG_T	*seq_id*	Sequence ID
ESS_ULONG_T	*seq_id_upper*	Sequence ID upper for future
ESS_TIME_T	*time_start*	Operation start time
ESS_TIME_T	*time_end*	Operation end time
ESS_USERNAME_T	*username*	Executing user
ESS_UCHAR_T	*type*	Record type
ESS_UCHAR_T	*state*	Do not use this field and client
ESS_CHAR_T	*reserved1*	For future expansion

Data Type	Field	Description
ESS_TRANSACTION_REQSPECIFIC_T	*reqSpecDat*	Request specific data

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);
```

Data Type	Field	Description
ESS_UCHAR_T	*InpType*	is it time based or sequence id
ESS_UCHAR_T	*reserved1*	reserved
ESS_UCHAR_T	*reserved2*	reserved
ESS_UCHAR_T	*reserved3*	reserved
ESS_TIME32_T ESS_ULONG_T	*InpTime* *num_seq_id_range*	A union variable for the following values: ● Time to start replay ● Number of sequence ID-based input structures to follow

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;
```

Data Type	Field	Description
ESS_UCHAR_T	*ucReqType*	Request type
ESS_UCHAR_T	*reserved1*	Reserved
ESS_UCHAR_T	*reserved2*	Reserved
ESS_UCHAR_T	*reserved3*	Reserved
ESS_FILENAME_T ESS_LOG_DATALOAD_T ESS_LOG_DIMBLD_T ESS_FILENAME_T	*calcname* *dataload_info* *dimbld_info* *tmpotlfilename*	A union variable for the following values: ● Calc file name ● Data load details ● Build load details ● Temporary outline file name

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:

```
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;
```

Data Type	Field	Description
ESS_USERNAME_T	*UserName*	The user or group name
ESS_APPNAME_T	*AppName*	The application name
ESS_ACCESS_T	*Access*	The assigned access privilege to the application for the user or group. This field can take any combination of the following bit values: ● ESS_PRIV_NONE ● ESS_PRIV_APPLOAD ● ESS_PRIV_APPDESIGN
ESS_ACCESS_T	*MaxAccess*	The maximum access privilege to the application for the user or group from all sources

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,
  ESS_GROUPAPPEX_T, *ESS_PGROUPAPPEX_T, **ESS_PPGROUPAPPEX_T;
```

Data Type	Field	Description
ESS_USERNAME_T	*UserName*	The user or group 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	*Type*	Type of the structure. This field can contain the following values: ● ESS_TYPE_USER ● ESS_TYPE_GROUP
ESS_APPNAME_T	*AppName*	The application name
ESS_ACCESS_T	*Access*	The assigned access privilege to the application for the user or group. This field can take any combination of the following bit values: ● ESS_PRIV_NONE ● ESS_PRIV_APPLOAD ● ESS_PRIV_APPDESIGN
ESS_ACCESS_T	*MaxAccess*	The maximum access privilege to the application for the user or group from all sources

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;
```

Data Type	Field	Description
ESS_USERNAME_T	*UserName*	The user or group name
ESS_USERNAME_T	*ProviderName*	Name of the user directory. Example: `@Native Directory`

Data Type	Field	Description
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	*Type*	Type of the structure. This field can contain the following value: • ESS_TYPE_USER • ESS_TYPE_GROUP
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.
ESS_FTRNAME_T	*FilterName*	The name of the assigned database filter, if any. If none, the first byte is NULL.

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.

```
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_PUSERINFO_T, **ESS_PPUSERINFO_T,
  ESS_GROUPINFO_T, *ESS_PGROUPINFO_T, **ESS_PPGROUPINFO_T;
```

Data Type	Field	Description
ESS_USERNAME_T	*Name*	User or group name
ESS_APPNAME_T	*AppName*	Name of currently connected application (if applicable)
ESS_DBNAME_T	*DbName*	Name of the currently connected database(if applicable)
ESS_BOOL_T	*Login*	Flag to indicate login status (users only)
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;
```

Data Type	Field	Description
ESS_USERNAME_T	*Name*	User name
ESS_USERNAME_T	*ProviderName*	Name of the user directory. Example: `@Native Directory`
ESS_PASSWORD_T	*Password*	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.
ESS_APPNAME_T	*AppName*	Name of the currently connected application (if applicable)
ESS_DBNAME_T	*DbName*	Name of the currently connected database (if applicable)
ESS_BOOL_T	*Login*	Flag to indicate login status
ESS_USHORT_T	*Type*	Type of the structure. This field can contain the following value: ● ESS_TYPE_USER
ESS_ACCESS_T	*Access*	User assigned default access privileges. Values can be 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 */

Data Type	Field	Description
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_USHORT_T	*FailCount*	Count of the failed login attempts since the last successful login
ESS_LOGINID_T	*LoginId*	User login identification tag
ESS_DESC_T	*Description*	User description
ESS_EMAIL_T	*EMailID*	User email address
ESS_BOOL_T	*LockedOut*	Flag indicating that the user is locked out
ESS_BOOL_T	*PwdChgNow*	Flag indicating that the user must change the password
ESS_PROTOCOL_T	*protocol*	External authentication protocol.
ESS_CONNPARAM_T	*connparam*	Unique identity attribute identifying a user or group in a directory. Example: `native://nvid=f0ed2a6d7fb07688:5a342200:1265973105c:-7f46?USER`

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 standardESS_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;
```

```
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_USERINFOEX_T, *ESS_PUSERINFOEX_T, **ESS_PPUSERINFOEX_T;
```

Data Type	Field	Description
ESS_USERNAME_T	*Name*	Externally authenticed User name.
ESS_PASSWORD_T	*Password*	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.
ESS_APPNAME_T	*AppName*	Name of currently connected application (if applicable)
ESS_DBNAME_T	*DbName*	Name of the currently connected database(if applicable)
ESS_BOOL_T	*Login*	Flag to indicate login status (users only)
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 /* Administrator, 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_PROTOCOL_T	*protocol*	External authentication protocol.

Data Type	Field	Description
ESS_CONNPARAM_T	*connparam*	External authentication connection.

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. 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.

```
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;
```

Data Type	Field	Description
ESS_SVRNAME_T	*Server*	Name of server where variable is defined (optional)
ESS_APPNAME_T	*AppName*	Name of application to restrict variable to
ESS_DBNAME_T	*DbName*	Name of database to restrict variable to. If used, it requires that application be set.
ESS_MBRNAME_T	*VarName*	Name of substitution variable.
ESS_CHAR_T	*VarValue[256]*	Value of substitution variable.

8 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.

Function	Description
EssListAliases	Lists all the alias tables in the active database.
EssLoadAlias	Loads an alias table for the active database from a structured text file
EssGetAlias	Gets the active alias table name from the active database for a user.
EssSetAlias	Sets the active alias table in the active database for a user.
EssDisplayAlias	Dumps contents of alias table in active database.
EssRemoveAlias	Removes an alias table from the active database.
EssClearAliases	Clears all alias tables for the active database.

C Main API Application Functions

The application functions can create new applications, and modify, copy, get information about and otherwise manage existing applications.

Function	Description
EssGetActive	Gets the names of the caller's current active application and database.
EssSetActive	Sets the callers active application and database.
EssClearActive	Clears the user's current active application and database.
EssListApplications	Lists all applications which are accessible to the caller.
EssConvertApplicationtoUnicode	Converts a non Unicode mode application to a Unicode mode application.
EssCreateApplication	Creates a new application, either on the client or the server.
EssCreateApplicationEx	Creates a new application with the option of setting the application type: Unicode- or non-Unicode mode.
EssCreateStorageTypedApplicationEx	Creates a new application with options for setting the data storage mode (block or aggregate) and application type (Unicode- or non-Unicode mode).
EssDeleteApplication	Deletes an existing application, either on the client or the server.
EssRenameApplication	Renames an existing application, either on the client or the server.
EssCopyApplication	Copies an existing application, either on the client or the server, to a new application, including all associated databases and objects.
EssGetApplicationInfoEx	Gets information from one or more applications
EssGetApplicationState	Gets an application state structure, which contains user-configurable parameters for the application.
EssSetApplicationState	Sets user-configurable parameters for the application using the application's state structure.

Function	Description
EssGetApplicationInfo	Gets an application's information structure, which contains non user-configurable parameters for the application.
EssLoadApplication	Starts an application on the server.
EssUnloadApplication	Stops an application on the server.

C Main API Attributes Functions

These C Main functions are for attributes.

Function	Description
EssCheckAttributes	Returns the attribute type for given attribute dimensions, base dimensions, attribute members, and base members
EssFreeStructure	Frees memory dynamically allocated for string type attribute information
EssGetAssociatedAttributesInfo	Returns the attribute members associated with a given base member
EssGetAttributeInfo	Returns attribute information for a given attribute member or dimension
EssGetAttributeSpecifications	Retrieves attribute specifications for the outline

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.

Function	Description
EssBeginDataload	Starts sending an update specification to the active database.
EssBeginDataloadASO	Starts a data load on an aggregate storage database.
EssClearDatabase	Clears all loaded data in the active database.
EssCommitDatabase	Forces all data blocks in the active database to be written to disk.
EssCopyDatabase	Copies an existing database, either on the client or the server, to a new database, including all associated databases and objects.
EssCreateDatabase	Creates a new database within an application, on client or server.
EssDeleteDatabase	Deletes an existing database from an application, on client or server.
EssEndDataload	Marks the end of an update specification being sent to the active database.

Function	Description
EssGetCurrencyRateInfo	Gets a list of structures containing rate information for all members of the tagged currency partition dimension in the active database outline.
EssGetDatabaseInfo	Gets a database's information structure, which contains non user-configurable parameters for the database.
EssGetDatabaseInfoEx	Gets information for one or more databases.
EssGetDatabaseNote	Gets a database's note-of-the-day message.
EssGetDatabaseState	Gets a database's state structure, which contains user-configurable parameters for the database.
EssGetDatabaseStats	Gets the active database's Stats structure, which contains statistical information about the database.
EssListCurrencyDatabases	Lists all currency databases within a specific application that are accessible to the caller.
EssListDatabases	Lists all databases that are accessible to the caller, either within a specific application, or on an entire server.
EssListExistingLoadBuffers	Returns the list of structures that describe existing data load buffers for an aggregate storage database.
EssLoadBufferInit	Creates a temporary data load buffer.
EssLoadBufferTerm	Destroys the temporary data-load memory buffer(s) allocated by oadBufferInit.
EssLoadDatabase	Starts a database.
EssMergeDatabaseData	Merges two or more data slices into a single data slice.
EssRenameDatabase	Renames a database on client or server.
EssSetDatabaseNote	Sets a database's note-of-the-day message.
EssSetDatabaseState	Sets user-configurable parameters for the database using the database's state structure.
EssUnloadDatabase	Stops a database within an application on the server.
EssValidateDB	Checks the database for data integrity.

C Main API Database Member Functions

These functions obtain information about database members and build database dimensions.

Function	Description
EssQueryDatabaseMembers	Performs a report-style query to list a selection of database member information.
EssCheckMemberName	Checks if a string is a valid member name within the active database outline.
EssGetMemberInfo	Gets a structure containing information about a specific member in the active database outline.

Function	Description
EssGetMemberCalc	Gets the calc equation for a specific member in the active database outline.
EssGetDimensionInfo	Gets dimension information.
EssBuildDimension	Allows the creation of a dimension in the active database from a data file and rules file.
EssBuildDimFile	This function builds a data file to be used in the addition or removal of members from the outline in the active database.
EssBuildDimStart	This function starts the process of the adding or removing members from the outline in the active database.

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.

- EssDTInit
- EssDTOpen
- EssDTClose
- EssDTExit
- EssDTGetData
- EssDTGetHeader
- EssDTGetHeaderInfo
- EssDTListReports
- EssDTAPIClose
- EssDTAPIConnect
- EssDTAPIExecuteReport
- EssDTAPIExit
- EssDTAPIGetData
- EssDTAPIGetColumns
- EssDTAPIGetError
- EssDTAPIGetInfo
- EssDTAPIGetReports
- EssDTAPIInit
- EssDTAPISetConnection
- EssDTAPISetInfo

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.

Function	Description
EssArchiveBegin	Prepares database for archive by setting READ-ONLY status.
EssArchiveEnd	After archive, returns database status to READ-WRITE.
EssCalcFile	Executes a calc script against the active database from a file.
EssExport	Exports data from the current database to a text file.
EssImport	Imports data from text files and other sources to the current database.
EssImportASO	Imports data from different sources to an aggregate storage database.
EssListDbFiles	Retrieves information on specified index and data files

Function	Description
EssReportFile	Sends a report specification to the active database from a file.
EssSetDefaultCalcFile	Sets the default calc script for the active database from a calc script file.
EssUpdateFile	Sends an update specification to the active database from a file.
EssUpdateFileEx	Sends an update specification to the active database from a file, capturing any data load errors.
EssUpdateFileASO	Sends an update specification to the active aggregate storage database from a file.
EssUpdateFileASOEx	Sends an update specification to the active aggregate storage database from a file, capturing any data load errors.
EssUpdateFileUTF8ASO	Sends an update specification to the active aggregate storage database from a UTF-8-encoded file.
EssUpdateFileUTF8ASOEx	Sends an update specification to the active aggregate storage database from a UTF-8-encoded file, capturing any data load errors.
EssUpdateFileUtf8Ex	Sends an update specification to the active database from a UTF-8-encoded file, capturing any data load errors
EssDisplayTriggers	Returns a list of all triggers associated with a database.
EssMdxTrig	Manipulates triggers based on the operations contained in an MDX language file.
EssListSpoolFiles	Returns a list of all the spool files associated with a database.
EssGetSpoolFile	Returns a specific spool file associated with a database.
EssDeleteAllSplFiles	Deletes all the spool files associated with a database.
EssDeleteSplFile	Deletes a specific spool file.

C Main API Group Administration Functions

These functions create groups, set and modify group attributes, and obtain information about existing groups.

Function	Description
EssListGroups	Lists all groups who have access to a particular Essbase Server.
EssCreateGroup	Creates a new group.
EssDeleteGroup	Deletes an existing group.
EssRenameGroup	Renames an existing group.
EssGetGroup	Gets a group information structure, which contains security information for the group.
EssSetGroup	Sets a group information structure.

Function	Description
EssGetGroupList	Gets the list of users who are members of a group (or the list of groups to which a user belongs).
EssSetGroupList	Sets list of users who are members of group.
EssAddToGroup	Adds user to a list of group members.
EssDeleteFromGroup	Removes a user from a list of group members

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.

Function	Description
EssAutoLogin	Displays a dialog box which allows the user to log in to an Essbase Server, and optionally selects an active application and database.
EssCreateLocalContext	Creates a local API context for use in local API operations.
EssDeleteLocalContext	Releases a local context previously created by EssCreateLocalContext().
EssGetAPIVersion	Gets the full version number of the connected API client module.
EssGetVersion	Gets the full version number of the connected Essbase Server.
EssInit	Initializes the API and message database.
EssLogin	Logs a user in to the Essbase Server.
EssLoginAs	Logs in to the Essbase Server as another user.
EssLoginEx	Logs in to the Essbase Server using an authentication token.
EssLoginExAs	Logs in to the Essbase Server as another user, using an authentication token.
EssLoginSetPassword	Logs in a user, and changes the password.
EssLogout	Logs a user out from an Essbase Server.
EssLogoutUser	Allows a Supervisor or Application Designer to disconnect another user from an Essbase Server.
EssLogSize	Returns the size of the Essbase Server log file (`essbase.log`), or of the application log file (*`appname`*`.log`).
EssShutdownServer	Allows a Supervisor to remotely stop the Essbase Server.
EssTerm	Terminates the API and releases all system resources used by the API.
EssValidateHCtx	Validates a specific API context handle (`hCtx`).
EssWriteToLogFile	Writes a message to the Essbase Server log file (`essbase.log`), or to the application log file (*`appname`*`.log`).

C Main API LRO Functions

These functions create, retrieve and delete LROs and return information about them.

Function	Description
EssLROAddObject	Links a reporting object to a data cell in an Essbase database.
EssLRODeleteCellObjects	Deletes all objects linked to a given data cell in an Essbase database.
EssLRODeleteObject	Deletes a specific object linked to a data cell in an Essbase database.
EssLROGetCatalog	Retrieves a list of LRO catalog entries for a given data cell in an Essbase database.
EssLROGetCatalogBatch	Retrieves a list of LRO catalog entries for multiple data cells in an Essbase database.
EssLROGetObject	Retrieves an object linked to a data cell in an Essbase database.
EssLROListObjects	Retrieves a list of all objects linked to cells in the active database for a given user name and/or modification date.
EssLROPurgeObjects	Deletes all objects linked to cells in the active database for a given user name and/or modification date.
EssLROUpdateObject	Stores an updated version of an LRO on the server.

C Main API Location Aliases Functions

These functions create, delete and list location aliases.

Function	Description
EssCreateLocationAlias	Maps an alias name to the host name, application name, database name, user login name, and user password
EssDeleteLocationAlias	Deletes an existing location alias
EssGetLocationAliasList	Returns all location aliases and the names to which the location aliases are mapped

C Main API Memory Allocation Functions

These functions manage memory for an application by allocating, reallocating and freeing blocks of memory.

Function	Description
EssAlloc	Allocates a block of memory, using the defined memory allocation scheme.
EssRealloc	Reallocates a previously-allocated block of memory.
EssFree	Frees a previously allocated block of memory, using the defined memory allocation scheme.

C Main API Miscellaneous Functions

These functions manage asynchronous processes, obtain state information, handle log files, and retrieve messages.

Function	Description
EssGetProcessState	Gets the current state of an asynchronous process, such as a calculate or data import.
EssCancelProcess	Cancels an asynchronous process which has not yet completed.
EssGetLogFile	Copies all or part of an application log file from the server to the client.
EssDeleteLogFile	Deletes an application log file on the server.
EssGetGlobalState	Gets the server global state structure which contains parameters for system administration.
EssSetGlobalState	Sets the server global state structure which contains parameters for system administration.
EssSetPath	Sets the ESSBASEPATH environment variable for the current process.

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.

Function	Description
EssGetLocalPath	Gets the full local file for an object file on the client.
EssListObjects	Lists all objects of types specified.
EssGetObjectInfo	Gets information about a specified object.
EssGetObject	Copies an object from the server to a local file, and optionally locks it.
EssPutObject	Copies an object from a local file to the server, and optionally unlocks it.
EssLockObject	Locks an object on the server to prevent other users from updating it.
EssUnlockObject	Unlocks a locked object on the server.
EssCreateObject	Creates a new object.
EssDeleteObject	Deletes an existing object.
EssRenameObject	Renames an existing object.
EssCopyObject	Copies an object.

C Main API Partition Functions

These functions manage partition operations on a database.

Function	Description
EssPartitionApplyOtlChangeFile	Tells the server to apply metadata changes to files.
EssPartitionApplyOtlChangeFileEx	Tells the server to apply metadata changes to files.
EssPartitionApplyOtlChangeRecs	Tells the server to apply metadata changes to records.
EssPartitionCloseDefFile	Closes the shared partition definition file.
EssPartitionFreeDefCtx	Frees memory dynamically allocated under shared partition context structures.
EssPartitionFreeOtlChanges	Frees up memory allocated by the ReadMetaChange routine.
EssPartitionGetAreaCellCount	Returns the number of cells in the specified slice string.
EssPartitionGetList	Returns a list of the partition partition definitions in which the currently selected database participates.
EssPartitionGetOtlChanges	Pulls meta data changes from a given source.
EssPartitionGetReplCells	Replicates all data cells that are identified in the replication partition from the source database to the selected target database.
EssPartitionNewDefFile	Creates and opens a new shared partition, definition file based upon input parameters supplied.
EssPartitionOpenDefFile	Opens an existing shared partition definition file.
EssPartitionPurgeOtlChangeFile	Purges meta changes made previous to the time specified with the TimeStamp parameter.
EssPartitionPutReplCells	Replicates all data cells that are identified in the replication partition from the selected source database to the target database.
EssPartitionReadDefFile	Replicates all data cells that are identified in the replication partition from the selected source database to the target database.
EssPartitionReadOtlChangeFile	Reads meta changes from a file into memory.
EssPartitionReplaceDefFile	Tells the server that a new shared partition file has been sent, which replaces any existing file for this database.
EssPartitionResetOtlChangeTime	Takes two partitions, one source and one destination. It takes the "last meta change" time from the source partition and assigns it as the "last meta change" time of the destination partition.
EssPartitionValidateDefinition	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.
EssPartitionValidateLocal	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.
EssPartitionWriteDefFile	Writes the current memory version of the shared partition definition file to disk.

C Main API Performance Stats Functions

These functions provide I/O performance statistics on threads, databases and applications.

Function	Description
EssDumpPerfStats	Provides a pointer to the character array that contains performance statistics tables
EssGetStatBufSize	Provides a pointer to the size of the buffer needed for the performance statistics tables
EssResetPerfStats	Resets values in the performance statistics tables to zero

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.

Function	Description
EssReport	Sends a report specification to the active database as a single string.
EssBeginReport	Starts sending a report specification to the active database.
EssEndReport	Marks the end of a report specification being sent to the active database.
EssUpdate	Sends an update to the active database as a single string.
EssUpdateFileASO	Sends an update specification to the active aggregate storage database from a file.
EssUpdateFileUTF8ASO	Sends an update specification to the active aggregate storage database from a UTF-8-encoded file.
EssBeginUpdate	Starts sending an update specification to the active database.
EssEndUpdate	Marks the end of an update specification being sent to the active database.
EssGetString	Gets a string of data from the active database.
EssSendString	Sends a string of data to the active database.
EssCalc	Sends and optionally executes a calc script against the active database as a single string.
EssBeginCalc	Starts sending a calc script and optionally executes it against the active database.
EssEndCalc	Marks the end of a calc script being sent to the active database.
EssDefaultCalc	Executes the default calculation for the active database.
EssGetDefaultCalc	Gets the default calc script for the active database.
EssListCalcFunctions	Lists all available calculator functions.
EssSetDefaultCalc	Sets the default calc script for a database.

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.

Function	Description
EssListFilters	Lists all filters for a database.
EssGetFilter	Starts getting the contents of a filter.
EssGetFilterRow	Gets the next row of a filter.
EssCreateFilter	Creates a filter. Starts setting the contents of the filter.
EssSetFilter	Creates or replaces a filter. Starts setting the contents of the filter.
EssSetFilterRow	Gets the next row of a filter.
EssGetFilterList	Gets the list of users who are assigned a filter.
EssSetFilterList	Sets the list of users who are assigned a filter.
EssDeleteFilter	Deletes an existing filter.
EssRenameFilter	Renames an existing filter.
EssCopyFilter	Copies an existing filter.
EssVerifyFilter	Verifies the syntax of a series of filter row strings against a specified database.
EssVerifyFilterRow	Verifies the syntax of a single filter row string against a specified database.

C Main API Substitution Variables Functions

These functions create, retrieve and delete substitution variables and return information about them.

Function	Description
EssCreateVariable	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.
EssDeleteVariable	This function deletes a substitution variable.
EssGetVariable	This function retrieves the value of a substitution variable.
EssListVariables	This function lists all substitution variables that conform to the input criteria.

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.

Function	Description
EssListUsers	Lists all users who have access to a particular Essbase Server.
EssCreateUser	Creates a new user.
EssDeleteUser	Deletes an existing user.
EssRenameUser	Renames an existing user.
EssGetUser	Gets a user information structure, which contains security information for user.
EssSetUser	Sets a user information structure that contains security information for user.
EssResetUser	Resets the user's security structure to its initial state.
EssSetPassword	Sets a user's password, erasing the existing password.
EssGetApplicationAccess	Gets a list of user application access structures, which contain information about user access to applications.
EssSetApplicationAccess	Sets a list of user application access structures.
EssGetDatabaseAccess	Gets a list of user database access structures.
EssSetDatabaseAccess	Sets a list of user database access structures.
EssGetCalcList	Gets the list of calc scripts objects accessible to the user.
EssSetCalcList	Sets the list of calc script objects which are available to a user.
EssListConnections	Lists all users who are connected to the current application and database.
EssListLogins	Lists information about currently connected users.
EssListRequests	Lists information about current Essbase user sessions or requests.
EssKillRequest	Terminates all or specific Essbase user sessions or requests.
EssListLocks	Lists all users who are connected to a specific application and database.
EssRemoveLocks	Removes all data block locks held by a user on a database.

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.

Function	Description
EssAddToGroupEx	Adds a user to the specified group. Similar to EssAddToGroup, but can accept a user directory specification or unique identity attribute.
EssCreateExtGroup	Creates a group in the external user directory.
EssDeleteFromGroupEx	Removes a user from a group. Similar to EssDeleteFromGroup, but can accept a user directory specification or unique identity attribute.
EssDeleteGroupEx	Deletes an existing group. Similar to EssDeleteGroup, but can accept a user directory specification or unique identity attribute.
EssDeleteUserEx	Deletes a user. Similar to EssDeleteUser, but can accept a user directory specification or unique identity attribute.
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.
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.
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.
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.
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.
EssKillRequestEx	Terminates specific user sessions or requests. Similar to EssKillRequest, but the input structure can include user directories and unique identity attributes.
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.
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.
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.
EssListLoginsEx	Returns the list of log in instances in the current session. Similar to EssListLogins, but includes users hosted in a user directory.
EssListRequestsEx	Returns information about active sessions and requests. Similar to EssListRequests, but includes users hosted in a user directory.
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.

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.

Function	Description
EssSetSSSecurityMode	Migrates Essbase Server and any existing users and groups to Oracle Hyperion Enterprise Performance Management System security mode.
EssSetUserToSS	Migrates a user to EPM System security mode.
EssSetGroupToSS	Migrates a group to EPM System security mode.
EssSetUsersToSS	Migrates all users to EPM System security mode.
EssSetGroupToSS	Migrates all groups to EPM System security mode.
EssGetEssbaseSecurityMode	Displays the type of security in use.
EssListSSMigrFailedUsers	Displays users that did not successfully migrate to Shared Services.
EssListSSMigrFailedGroups	Displays groups that did not successfully migrate to Shared Services.
EssReRegisterApplication	Re-establishes one or all Essbase applications as Shared Services applications.
EssSetEasLocation	Set or change the Essbase Administration Server location that will be registered with Shared Services upon application creation or migration.

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.

Function	Description
EssSetServerMode	Sets the mode of Essbase Server to be Unicode or non-Unicode.
EssGetServerMode	Indicates whether the Essbase Server is in Unicode mode or non-Unicode mode.
EssCreateApplicationEx	Creates a Unicode mode application.
EssConvertApplicationtoUnicode	Converts a non Unicode mode application to a Unicode mode application.
EssGetApplicationInfo and EssGetApplicationInfoEx	Returns application information, including locale information.
EssUpdateFileUTF8ASO	Sends an update specification to the active aggregate storage database from a UTF-8-encoded file.
EssUpdateFileUTF8ASOEx	Sends an update specification to the active aggregate storage database from a UTF-8-encoded file, capturing any data load errors.
EssUpdateFileUtf8Ex	Sends an update specification to the active database from a UTF-8-encoded file, capturing any data load errors

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);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
GroupName	ESS_STR_T	Group name.

Parameter	Data Type	Description
UserName	ESS_STR_T	Name of user to add to group list.

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

```
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

```
ESS_FUNC_M EssAddToGroupEx (hCtx, GroupId, UserId, bUsingIdentity);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle (input).

Parameter	Data Type	Description
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 add to 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

- 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

```
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_BOOL_T bisIdentity;
   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 ("\tUser list is empty\n");
      }
   }

   return (sts);
}
```

See Also

- EssDeleteFromGroupEx
- EssGetGroupListEx
- EssListGroupsInfoEx

EssAlloc

Allocates a block of memory, using the defined memory allocation scheme.

Syntax

```
ESS_FUNC_M EssAlloc (hInstance, Size, ppBlock);
```

Parameter	Data Type	Description
hInstance	ESS_HINST_T	API instance handle.
Size	ESS_SIZE_T	Size of memory block to allocate.
ppBlock	ESS_PPVOID_T	Address of pointer to receive allocated memory block.

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

```
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
- EssArchiveEnd

EssArchiveBegin

Prepares the server for archiving by changing server mode to Read-Only.

Syntax

```
ESS_FUNC_M EssArchiveBegin (hCtx, AppName, DbName, FileName);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle
AppName	ESS_STR_T	Name of application to archive
DbName	ESS_STR_T	Name of database to archive
FileName	ESS_STR_T	Name of file to contain archive information

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

```
ESS_FUNC_M
ESS_ArchiveBegin(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);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	Login context
AppName	ESS_STR_T	Application Name **Note:** Works only at the database level. The AppName parameter specifies an Application in order to access the database residing within.
DbName	ESS_STR_T	Database Name
BackupFileName	ESS_STR_T	Full path to the backup file in which to archive data. Specify the full path, for example: `c:\hyperion\Test.arc`
OptionsFileName	ESS_FILENAME_T	Reserved for the future. **Note:** For this release, use an empty string.
bOverWrite	ESS_BOOL_T	Boolean: • ESS_TRUE—oveoorwrite existing back up file • ESS_FALSE—do not overwrite

Return Value

Returns:

- 0—If successful
- Error number—If unsuccessful

Access

The caller must have Essbase Administrator access to the database.

Example

```
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\r\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\r\n",sts);

    sts = EssRestoreDatabase (hCtx, AppName, DbName,
                                             BackupFileName, bForceDiffName,
                                             count, replaceVol);
    printf("EssRestoreDatabase sts: %ld\r\n",sts);

    if (replaceVol)
           EssFree(hInst, replaceVol);
}
```

See Also

- EssRestoreDatabase

EssArchiveEnd

Restores the server to "read-write" mode after archiving is complete.

Syntax

```
ESS_FUNC_M EssArchiveEnd (hCtx, AppName, DbName);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
AppName	ESS_STR_T	Name of archived application.
DbName	ESS_STR_T	Name of archived database.

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

```
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

```
ESS_FUNC_M EssAsyncBuildDim(hCtx, RulesObj, DataObj, MbrUser, bOverwrite, usBuildOption,
szTmpOtlFile)
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	Essbase API context handle.
RulesObj	ESS_POBJDEF_T	Pointer to rules file object definition structure.
DataObj	ESS_POBJDEF_T	Pointer to data file object definition structure.
MbrUser	ESS_PMBRUSER_T	SQL user structure (if data source is SQL database). A NULL SQL user structure indicates a non SQL data source.
bOverwrite	ESS_BOOL_T	Indicates overwrite or append error message to the error file.
usBuildOption	ESS_USHORT_T	Valid values: • `ESS_INCDIMBUILD_BUILD` Build members only. • `ESS_INCDIMBUILD_VERIFY` Build members and verify the outline. • `ESS_INCDIMBUILD_SAVEOTL` Build members and save the outline to a temp outline file. • `ESS_INCDIMBUILD_ALL` Build members, verify the outline, and restructure. • `ESS_INCDIMBUILD_ABORT` Abort the build process.
szTmpOtlFile	ESS_STR_T	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.

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

```
void ESS_AsyncBuildDim()
{
    ESS_STS_T sts = 0;
    ESS_OBJDEF_T Rules;
    ESS_OBJDEF_T Data;
    ESS_PMBRUSER_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
        {
            DisplyProcessStateInfo(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";
    sts = 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);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
pRules	ESS_POBJDEF_T	Pointer to the rules file object definition structure.
pData	ESS_POBJDEF_T	Pointer to the data file object definition structure.
pMbrUser	ESS_PMBRUSER_T	Pointer to the SQL user structure (if data source is a SQL database). A NULL SQL user structure indicates a non SQL data source.
abortOnError	ESS_USHORT_T	If TRUE, import stops on the first error. Otherwise, it continues.

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_BLDDL_STATE_DONE);

        if(!procState.ilProcessStatus)
        {
            sts = EssCloseAsyncProc(hCtx, &procState);
            printf("EssCloseAsyncProc sts: %ld\n",sts);
        }
    }
}
```

See Also

- EssAsyncBuildDim
- EssGetAsyncProcLog
- EssGetAsyncProcState
- EssCancelAsyncProc
- EssCloseAsyncProc

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);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
pRules	ESS_POBJDEF_T	Pointer to the rules file object definition structure.
pData	ESS_POBJDEF_T	Pointer to the data file object definition structure.
pUser	ESS_PMBRUSER_T	Pointer to the SQL user structure (if data source is a SQL database). A NULL SQL user structure indicates a non SQL data source.
usAbortOnError	ESS_USHORT_T	If TRUE, import stops on the first error. Otherwise, it continues.
ulBufferID	ESS_ULONG_T	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.

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

```
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
                {
                    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_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);
```

Parameter	Data Type	Description
hInstance	ESS_HINST_T	API instance handle

Server	ESS_SVRNAME_T	Network server name string
		The server name can be expressed 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,
		`https://myhost:13080/aps/Essbase?clustername=Essbase-Cluster1&SecureMODE=Yes`

UserName	ESS_USERNAME_T	User name string
Password	ESS_PASSWORD_T	Password string
AppName	ESS_ APPNAME _T	Application name
DbName	ESS_ DBNAME _T	Database name
Options	ESS_USHORT_T	Options flag. Values: ● AUTO_NODIALOG—Attempts to log the user in without displaying the dialog, using the default settings (from the above arguments). ● AUTO_NOSELECT—Allows the user to log in without selecting an application and database (lower part of the dialog is not displayed). You can use both AUTO_NODIALOG and AUTO_NOSELECT with an OR operator (|) to log in a user without a dialog box and not select an application and database. ● **AUTO_NODIALOG|AUTO_NOSELECT: AUTO_DEFAULT**—Enables the user to log in and select an application and database interactively in the dialog box.
pAccess	ESS_PACCESS_T	Address of variable to receive database access level.
phCtx	ESS_PHCTX_T	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.

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

```
ESS_FUNC_M EssBeginCalc (hCtx, Calculate);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
Calculate	ESS_BOOL_T	Controls calculation of the calc script. If TRUE, the calc script is executed.

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

```
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);
      while(!sts && (pState.State !=
             ESS_STATE_DONE))
         sts = EssGetProcessState (hCtx, &pState);
```

```
    }
    return(sts);
}
```

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

```
ESS_STS_T EssBeginDataload (hCtx, Store, Unlock, abortOnError, pRules);
```

Parameter	Data Type	Description
hCtx;	ESS_HCTX_T	API context handle.
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	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.
abortOnError;	ESS_BOOL_T	If TRUE, data load stops on the first error. Otherwise, data load continues.
pRules;	"ESS_OBJDEF_T" on page 151	Pointer to the rules file object definition structure.

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

```
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

```
ESS_FUNC_M EssBeginDataloadASO (hCtx, Store, Unlock, abortOnError, pRules, ulBufferId);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
Store	ESS_BOOL_T	Controls storage of data. If ESS_TRUE, data is stored in the server; if ESS_FALSE, no data is stored.
Unlock	ESS_BOOL_T	Not supported for aggregate storage databases. You must always pass ESS_FALSE for this parameter.
abortOnError	ESS_BOOL_T	ESS_TRUE indicates that the data load will be aborted in case of errors during the process.
pRules	"ESS_OBJDEF_T" on page 151	Pointer to the rules file object definition structure.
ulBufferId	ESS_ULONG_T	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.

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

```
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

- EssUpdateFileUTF8ASO
- EssListExistingLoadBuffers
- EssMergeDatabaseData

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

```
ESS_STS_T EssBeginDataloadEx (hCtx, Store, Unlock, abortOnError, pRules, fullMbrNames);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
Store	ESS_BOOL_T	Controls storage of data. • ESS_TRUE Data is stored in the server. • ESS_FALSE No data is stored.
Unlock	ESS_BOOL_T	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.
abortOnError;	ESS_BOOL_T	If TRUE, data load stops on the first error. Otherwise, data load continues.
pRules	"ESS_OBJDEF_T" on page 151	Pointer to the rules file object definition structure.
fullMbrNames	ESS_BOOL_T	If TRUE, the error log prints full member names for the entire record.

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

```
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

```
ESS_FUNC_M EssBeginIncrementalBuildDim(hCtx);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	Essbase API context handle.

Return Value

Returns zero if successful; error code if unsuccessful.

Example

```
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,&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
- 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);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
Output	ESS_BOOL_T	Controls output of data. If TRUE, data is output from the server, according to the specified report. If FALSE, no data is output.
Lock	ESS_BOOL_T	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 **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

```
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

- 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)
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
RulesObj	ESS_POBJDEF_T	Pointer to rules file object definition structure.
usBuildOption	ESS_USHORT_T	Valid values: • ESS_INCDIMBUILD_BUILD Build members only. • ESS_INCDIMBUILD_VERIFY Build members and verify the outline. • ESS_INCDIMBUILD_SAVEOTL Build members and save the outline to a temp outline file. • ESS_INCDIMBUILD_ALL Build members, verify the outline, and restructure.
szTmpOtlFilename	ESS_STR_T	The temp outline file name. Essbasecreates a temporary outline file with extension "otb" if the resulting outline in this round of dimension build has outline verification errors.

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

```
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,&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
- 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

```
ESS_FUNC_M EssBeginUpdate (hCtx, Store, Unlock);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
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	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.

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

```
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);
   /*****************************************************************/
   /*                                                               */
   /* When a SQL data source is defined in the rules file, define   */
   /* the variables in the ESS_OBJDEF_T DataObj structure as follows: */
   /*    DataObj.hCtx     = hCtx;                                   */
   /*    DataObj.AppName  = NULL;                                   */
   /*    DataObj.DbName   = NULL;                                   */
   /*    DataObj.ObjType  = ESS_OBJTYPE_NONE;                       */
   /*    DataObj.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 =  EssBuildDimension (hCtx, &RulesObj, &DataObj,       */
   /*           &User, ErrorName);                                  */
   /*                                                               */
   /*****************************************************************/
}
```

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

```
ESS_FUNC_M EssBuildDimFile (hCtx, RulesObj, DataObj, MbrUser,
ErrorName, fOverwriteErrorFile);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
RulesObj	"ESS_OBJDEF_T" on page 151	Pointer to rules file object definition structure.
DataObj	"ESS_OBJDEF_T" on page 151	Pointer to data file object definition structure.
MbrUser	"ESS_MBRUSER_T" on page 148	SQL user structure (if data source is SQL database). NULL structure indicates a non-SQL data source.
ErrorName	ESS_STR_T	Error name output on client.
fOverwriteErrorFile	ESS_BOOL_T	A Boolean value which detemines whether this function overwrites an existing file name *ErrorFile*.

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 repeatly 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 resturcturing.

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
hCtx	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

```
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

```
ESS_FUNC_M EssCalc (hCtx, Calculate, CalcScript);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
Calculate	ESS_BOOL_T	Controls calculation of the calc script. If TRUE, the calc script is executed and the call is asynchronous.
CalcScript	ESS_STR_T	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 **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

```
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

```
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

```
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

- EssAsyncImportASO
- EssGetAsyncProcLog
- EssGetAsyncProcState
- EssCloseAsyncProc

EssCancelProcess

Cancels an asynchronous process that has not yet completed

Syntax

```
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

```
ESS_VOID_T
ESS_CancelProcess(ESS_HCTX_T hCtx)
{
   ESS_FUNC_M            sts = ESS_STS_NOERR;
   ESS_STR_T         Script;
   ESS_PROCSTATE_T   pState;
   ESS_USHORT_T      Count;

   Script = "CALC ALL;";

   sts = EssBeginCalc (hCtx,ESS_TRUE);

   if (!sts)
      sts = EssSendString (hCtx, Script);
   if (!sts)
      sts = 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

Parameter	Data Type	Description
hCtx;	ESS_HCTX_T	Context handle
Count;	ESS_USHORT_T	Number of given dimensions and members
pMemberNameArray;	ESS_PMBRNAME_T	An array of names of given dimensions and members
ppAttributeTypeArray;	ESS_PPUSHORT_T	One of the following constant identifiers (see Table 6, "C API Attributes Terminology," on page 102) for the attribute type array: • ESS_ATTRIBUTE_DIMENSION • ESS_ATTRIBUTE_MEMBER • ESS_STANDARD_DIMENSION • ESS_STANDARD_MEMBER • ESS_BASE_DIMENSION • ESS_BASE_MEMBER • ESS_ATTRIBUTED_MEMBER • ESS_INVALID_MEMBER

Access

This function requires no special privileges.

Example

```
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");
                 }
                 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);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
MbrName	ESS_STR_T	Member name to be verified.
pValid	ESS_PBOOL_T	Address of variable to receive valid member flag. Set to TRUE if member is valid.

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

```
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

```
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

```
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

```
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

```
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

```
ESS_FUNC_M EssClearDatabase (hCtx);
```

Parameter	Data Type	Description
hCtx	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

```
ESS_FUNC_M
ESS_ClearDb (ESS_HCTX_T     hCtx)
{
   ESS_FUNC_M        sts = ESS_STS_NOERR;
   sts = EssClearDatabase(hCtx);
   return (sts);
}
```

See Also

- EssDeleteDatabase
- EssUnloadDatabase
- EssSetActive

EssCloseAsyncProc

Closes the connection for a finished or canceled asynchronous dimension build or data load, and returns the current state of the process.

Syntax

```
ESS_FUNC_M EssCloseAsyncProc (hCtx, ProcState);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
ProcState	ESS_PBLDDL_STATE_T	Address of pointer to receive allocated process state structure.

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

```
ESS_FUNC_M EssClrSpanRelationalSource (hCtx);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.

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

```
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);
}
```

See Also

- EssSetSpanRelationalPartition

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

```
#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 */
#ifdef AD_UTF8
                        , ESS_API_UTF8
#endif
        };

        /* get appname and dbname from the argument list */
        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);
}
```

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);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
AppName	ESS_STR_T	The name of the application to be migrated. The named application must exist and not be in Unicode mode.

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

```
ESS_FUNC_M EssCopyApplication (hCtx, hSrcCtx, SrcApp, DestApp);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
hSrcCtx	ESS_HCTX_T	Not used—should be same as hCtx.
SrcApp	ESS_STR_T	Name of existing application to copy.
DestApp	ESS_STR_T	Name of new application. See "Application Name Limits" on page 1727.

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

```
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

```
ESS_FUNC_M EssCopyDatabase (hCtx, hSrcCtx, SrcApp, DestApp, SrcDb, DestDb);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
hSrcCtx	ESS_HCTX_T	Not used - should be same as *hCtx*.
SrcApp	ESS_STR_T	Name of source application.
DestApp;	ESS_STR_T	Name of destination application.
SrcDb;	ESS_STR_T	Name of existing database to copy.
DestDb	ESS_STR_T	Name of new database. See "Database Name Limits" on page 1728.

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

```
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

ESS_FUNC_M **EssCopyFilter** (*hCtx, hSrcCtx, SrcApp, DestApp, SrcDb, DestDb, SrcName, DestName*);

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
hSrcCtx	ESS_HCTX_T	Not used—should be same as *hCtx*.
SrcApp	ESS_STR_T	Source application name.
DestApp	ESS_STR_T	Destination application name.
SrcDb	ESS_STR_T	Source databasename.

Parameter	Data Type	Description
DestDb	ESS_STR_T	Destination database name.
SrcName	ESS_STR_T	Source name of existing filter to be copied.
DestName	ESS_STR_T	Destination name of copied filter. See "Filter Name Limits" on page 1728.

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

```
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

- EssRenameFilter

EssCopyObject

Copies an object to the server or client object system.

Syntax

```
ESS_FUNC_M EssCopyObject (hSrcCtx, hDestCtx, ObjType,
SrcApp, DestApp, SrcDb, DestDb, SrcObj, DestObj);
```

Parameter	Data Type	Description
hSrcCtx	ESS_HCTX_T	API context handle for source object. Can be local context handle returned by EssCreateLocalContext().
hDestCtx	ESS_HCTX_T	API context handle for destination object. Can be local context handle returned by EssCreateLocalContext().
ObjType	ESS_OBJTYPE_T	Object type (must be single type). See Bitmask Data Types for possible values.
SrcApp	ESS_STR_T	Source application name.
DestApp	ESS_STR_T	Destination application name.
SrcDb	ESS_STR_T	Source database name. If NULL, uses the source application subdirectory.
DestDb	ESS_STR_T	Destination database name. If NULL, uses the destination application subdirectory.
SrcObj	ESS_STR_T	Name of source object to copy from.
DestObj	ESS_STR_T	Name of 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. 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

```
ESS_FUNC_M
ESS_CopyObject(ESS_HCTX_T hCtx)
{
```

```
   ESS_FUNC_M          sts = ESS_STS_NOERR;
   ESS_HCTX_T         hDestCtx;
   ESS_STR_T          SrcApp;
   ESS_STR_T          DestApp;
   ESS_STR_T          SrcDb;
   ESS_STR_T          DestDb;
   ESS_STR_T          SrcObj;
   ESS_STR_T          DestObj;
   ESS_OBJTYPE_T      ObjType;

   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.\r\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

```
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

```
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.

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 **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).

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).

See Also

- EssCreateStorageTypedApplicationEx
- EssCreateDatabaseEx
- EssCreateObject
- EssConvertApplicationtoUnicode

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.

Syntax

```
ESS_FUNC_M EssCreateDatabase (hCtx, AppName, DbName, DbType);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
AppName	ESS_STR_T	Name of application to contain database.
DbName	ESS_STR_T	Name of database to create. See "Database Name Limits" on page 1728.
DbType	ESS_USHORT_T	Type of database to create. Can be ESS_DBTYPE_NORMAL, or ESS_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 **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

```
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

ESS_FUNC_M **EssCreateDatabaseEx** (*hCtx, AppName, DbName, DbType, bNonUniqueName*);

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	Essbase API context handle.
AppName	ESS_STR_T	Name of application to contain database.
DbName	ESS_STR_T	Name of database to create. See "Database Name Limits" on page 1728.
DbType	ESS_USHORT_T	Type of database to create. Can be ESS_DBTYPE_NORMAL, or ESS_DBTYPE_CURRENCY
bNonUniqueName	ESS_BOOL_T	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().

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

```
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

```
ESS_FUNC_M EssCreateDrillThruURL (hCtx, pUrl);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle
pUrl	ESS_PDURLINFO_T	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 (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("\nValid case:\n");
   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
hCtx	ESS_HCTX_T	API context handle (input).

Parameter	Data Type	Description
GroupName	ESS_STR_T	Name of group to create (input). See "Group Name Limits" on page 1728.

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

```
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

```
ESS_FUNC_M EssCreateExtUser (hCtx, UserName, Password, SecurityProvider,
ProviderConnectionParameters);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
UserName	ESS_STR_T	Name of user to create. See "User Name Limits" on page 1728.
Password	ESS_STR_T	Security password for new user. See "Password Limits" on page 1728. **Note:** The *Password* parameter has been made redundant by changes for Shared Services. You can use an empty string for this parameter.
SecurityProvider	ESS_PROTOCOL_T	The name of the external authentication mechanism.
ProviderConnectionParameters	ESS_CONNPARAM_T	Parameters used by the external authentication mechanism, if any.

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

```
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	Data Type	Description
hCtx	ESS_HCTX_T	API context handle
AppName	ESS_STR_T	Application name
DbName	ESS_STR_T	Database name
FilterName	ESS_STR_T	Filter name. See "Filter Name Limits" on page 1728.
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 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

```
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] = "@IDESCENDANTS(Scenario)";
      RowString[1] = "@IDESCENDANTS(Product)";
      RowString[2] = "Qtr1, @IDESCENDANTS(\"Colas\")";

      AccessAry[0] = ESS_ACCESS_READ;
      AccessAry[1] = ESS_ACCESS_NONE;
      AccessAry[2] = ESS_ACCESS_WRITE;
      /***** 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

```
ESS_FUNC_M EssCreateGroup (hCtx, GroupName);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
GroupName	ESS_STR_T	Name of 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 **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

```
ESS_FUNC_M
ESS_CreateGroup (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

```
ESS_FUNC_M EssCreateLocalContext (hInstance, UserName, Password, phLocalCtx);
```

Parameter	Data Type	Description
hInstance;	ESS_HINST_T	API instance handle.

Parameter	Data Type	Description
UserName;	ESS_STR_T	Currently not used - should be NULL.
Password;	ESS_STR_T	Currently not used - should be NULL.
phLocalCtx;	ESS_PHCTX_T	Address of variable to receive Essbase local context handle.

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

```
ESS_FUNC_M EssCreateLocationAlias (hCtx, pAlias, pHost, pApp, pDb, pName, pPassword);
```

Parameter	Data Type	Description
hCtx;	ESS_HCTX_T	API context handle
pAlias;	ESS_STR_T	Location alias
pHost;	ESS_STR_T	Target host
pApp;	ESS_STR_T	Target application
pDb;	ESS_STR_T	Target database

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

```
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

```
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

```
ESS_FUNC_M EssCreateStorageTypedApplication (hCtx, AppName, StorageType);
```

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.
StorageType	ESS_DATA_STORAGE_T	The data storage type of the new application. 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

```
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

```
ESS_FUNC_M EssCreateStorageTypedApplicationEx (hCtx, AppName, storageType, 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.

Parameter	Data Type	Description
StorageType	ESS_DATA_STORAGE_T	The data storage type of the new application. 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
usAppType	ESS_USHORT_T	The application type (Unicode or non-Unicode) of the new application. The valid values for usAppType are: ● ESS_APP_UNICODE - 0x0003—Unicode application. The function fails if the server is not in Unicode mode. ● ESS_APP_NONUNICODE - 0x0002—Non-Unicode application.

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

```
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

```
ESS_FUNC_M EssCreateUser (hCtx, UserName, Password);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
UserName	ESS_STR_T	Name of user to create. See "User Name Limits" on page 1728.
Password	ESS_STR_T	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 **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

```
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

```
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

```
/*
** 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'");

   /* 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'");
   }
   /* 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");
   }
   /* 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");
   }
   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           sts = ESS_STS_NOERR;
   ESS_PROCSTATE_T   pState;

   sts = EssDefaultCalc(hCtx);
   if (!sts)
```

```
   {
      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

```
ESS_FUNC_M EssDeleteAllSplFiles (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.

Access

This function requires the caller to have database Design privilege (ESS_PRIV_DBDESIGN) for the specified database.

See Also

- EssDisplayTriggers
- EssListSpoolFiles
- EssGetSpoolFile
- EssDeleteSplFile
- 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

```
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

```
ESS_FUNC_M EssDeleteDrillThruURL (hCtx, URLName);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle
URLName	ESS_STR_T	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 (ESS_PRIV_DBDESIGN) for the specified database.
- Caller must have selected the specified database as the active database using EssSetActive().

Example

```
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

```
ESS_FUNC_M EssDeleteFilter (hCtx, AppName, DbName, FilterName);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle
AppName	ESS_STR_T	Application name
DbName	ESS_STR_T	Database name
FilterName	ESS_STR_T	Filter name

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_DeleteFilter (ESS_HCTX_T  hCtx)
{
```

```
   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

```
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

```
ESS_FUNC_M EssDeleteGroup (hCtx, GroupName);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
GroupName	ESS_STR_T	Name of group to delete.

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_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
- EssListGroups
- EssRenameGroup

EssDeleteGroupEx

Deletes an existing group. Similar to EssDeleteGroup, but can accept a user directory specification or unique identity attribute for *GroupID*.

Syntax

```
ESS_FUNC_M EssDeleteGroupEx (hCtx, GroupId, bisIdentity );
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle (input).

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

```
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

```
ESS_FUNC_M
ESS_DeleteLogFile (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

```
ESS_FUNC_M EssDeleteObject (hCtx, ObjType, AppName, DbName, ObjName);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
ObjType	ESS_OBJTYPE_T	Object type (must be single type).
AppName	ESS_STR_T	Application name.
DbName	ESS_STR_T	Database name. If NULL, uses the application subdirectory.
ObjName	ESS_STR_T	Object name to delete.

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

```
ESS_FUNC_M
ESS_DeleteObject (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

Deletes a specific trigger logfile for a database.

Syntax

```
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	databasename.
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
- EssDeleteAllSplFiles
- EssMdxTrig

EssDeleteUser

Deletes a user.

Syntax

```
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

```
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

- EssCreateExtUser
- EssCreateUser
- EssListUsers
- EssRenameUser

EssDeleteUserEx

Deletes a user. Similar to EssDeleteUser, but can accept a user directory specification or unique identity attribute.

Syntax

```
ESS_FUNC_M EssDeleteUserEx (hCtx, UserId, bIsIdentity);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	Context handle (input).
UserId	ESS_STR_T	Name of user to delete (input). Can be specified as `username@provider` or as a unique identity attribute.
bIsIdentity	ESS_BOOL_T	Input. Indicates if *UserId* is a name or an identity. If TRUE, *UserId* is an identity.

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

```
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

```
ESS_FUNC_M EssDeleteVariable (hCtx, pVariable);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	Context handle to the API.
pVariable	"ESS_VARIABLE_T" on page 197	The pointer to the structure containing the description of the substitution variable being deleted.

Return Value

If successful, returns zero.

Example

```
/*
** 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");
}

/*********************************************************/
/* 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 the variables - at the level of the App          */
/*********************************************************/

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);
     }
  }
}
/*********************************************************/
/* List variables at the server level                    */
```

```
   /*******************************************************/
   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("\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 EssDeleteVariable\n\n\n");
   else
      printf("\n --> Error in EssDeleteVariable number: %d\n\n\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);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
AliasName	ESS_STR_T	Name of alias table.
pCount	ESS_PUSHORT_T	Address of variable to receive count of aliases.
ppAliases	"ESS_MBRALT_T" on page 148	Address of pointer to receive member alias 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

```
ESS_FUNC_M
ESS_DisplayAlias (ESS_HCTX_T    hCtx)
{
   ESS_FUNC_M     sts = ESS_STS_NOERR;
   ESS_USHORT_T  Count;
   ESS_USHORT_T  ind;
   ESS_PMBRALT_T Altlist;
   ESS_STR_T     AltName;

   AltName = "TestAlias";

   sts = 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 (sts);
}
```

See Also

- EssListAliases

EssDisplayTriggers

Lists all triggers for a database.

Syntax

```
ESS_FUNC_M EssDisplayTriggers (hCtx, AppName, DbName, pszTrg, pCount, ppTriggerList);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
AppName	ESS_STR_T	Application name.
DbName	ESS_STR_T	database name
pszTrg	ESS_STR_T	The name of the specific trigger to return information for. If pszTrg is "" (empty string), then all triggers in the specified database will be returned.
pCount	ESS_PUSHORT_T	Address of the variable to receive the number of triggers for which information will be returned.
ppTriggerList	ESS_PPTRIGGERINFO_T	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.

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

```
ESS_FUNC_M EssDTAPIClose (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
- 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

```
ESS_FUNC_M EssDTAPIConnect (pDTAPIInst);
```

Parameter	Data Type	Description
pDTAPIInst;	ESS_DTAPIHINST_T	Initialized drill-through instance handle for the given range of data cells

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

- EssDTAPISetConnection
- EssDTAPISetInfo

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);
```

Parameter	Data Type	Description
pDTAPIInst;	ESS_DTAPIHINST_T	Initialized drill-through instance handle for the given range of data cells
ppCol;	"ESS_DTAPIHEADER_T" on page 135	An array of report header structures for the given columns
pulCount;	ESS_PULONG_T	Number of data blocks in the *ppCol* report header information array

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

```
ESS_FUNC_M EssDTAPIGetData (pDTAPIInst, ppData, pulRowCount, pulColCount);
```

Parameter	Data Type	Description
pDTAPIInst;	ESS_DTAPIHINST_T	Initialized drill-through instance handle for the given range of data cells
ppData;	"ESS_DTAPIDATA_T" on page 134	An array of report data structures for the given data cells
pulRowCount;	ESS_PULONG_T	Number of rows for the data blocks in the *ppData* report data array
pulColCount;	ESS_PULONG_T	Number of columns for the data blocks in the *ppData* report data array

See Also

- Chapter 6, "Using the C Main API"
- "C Main API Structures" on page 114
- EssDTAPIClose
- EssDTAPIConnect
- EssDTAPIExecuteReport
- EssDTAPIExit
- EssDTAPIGetColumns
- EssDTAPIGetError
- EssDTAPIGetInfo
- EssDTAPIGetReports
- EssDTAPIInit
- EssDTAPISetConnection
- EssDTAPISetInfo

EssDTAPIGetError

Retrieves the error status, and message.

Syntax

```
ESS_FUNC_M EssDTAPIGetError (pDTAPIInst, ppData, pMsg, ulCount);
```

Parameter	Data Type	Description
pDTAPIInst;	ESS_DTAPIHINST_T	Initialized drill-through instance handle for the given range of data cells
ppData;	ESS_STS_T	Error status
pMsg;	ESS_PSTR_T	Error message
ulCount;	ESS_ULONG_T	Size of the error message buffer

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

EssDTAPIGetInfo

Retrieves drill-through connection information for a given drill-through handle.

Syntax

```
ESS_FUNC_M EssDTAPIGetInfo (pDTAPIInst, pDTInfo);
```

Parameter	Data Type	Description
pDTAPIInst;	ESS_PDTAPIHINST_T	Initialized drill-through instance handle for the 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

- 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
- EssDTAPIGetError
- EssDTAPIGetReports
- EssDTAPIInit

- EssDTAPISetConnection
- EssDTAPISetInfo

EssDTAPIGetReports

Returns the list of reports for the given drill-through instance handle.

Syntax

```
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

```
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

Notes

- **EssDTAPIInit**() initializes *ppDTAPIHInst*.
- Currently, *pDTAPIInit* (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
- `EssDTAPIClose`
- `EssDTAPIConnect`
- `EssDTAPIExecuteReport`
- `EssDTAPIExit`
- `EssDTAPIGetColumns`
- `EssDTAPIGetData`
- `EssDTAPIGetError`
- `EssDTAPIGetInfo`
- `EssDTAPIGetReports`
- `EssDTAPISetConnection`
- `EssDTAPISetInfo`

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);
```

Parameter	Data Type	Description
pDTAPIInst;	ESS_PDTAPIHINST_T	Pointer to a drill-through initialization structure
pEMC;	ESS_PSTR_T	Extended Member Comment
ulCount;	ESS_ULONG_T	Number of Extended Member Comment blocks
pDTInfo;	ESS_PSTR_T	Connection information

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`

- EssDTAPIConnect
- EssDTAPIExecuteReport
- EssDTAPIExit
- EssDTAPIGetColumns
- EssDTAPIGetData
- EssDTAPIGetError
- EssDTAPIGetInfo
- EssDTAPIGetReports
- EssDTAPIInit
- EssDTAPISetInfo

EssDTAPISetInfo

Sts drill-through connection information for a given drill-through handle.

Syntax

```
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
- EssDTExit
- 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

- EssDTListReports
- EssDTOpen

EssDTGetData

Retrieves an array of report data for the given drill-through instance handle.

Syntax

```
ESS_FUNC_M EssDTGetData (pDTInst, pData, pulCount);
```

Parameter	Data Type	Description
pDTInst;	ESS_PDTHINST_T	Initialized drill-through instance handle
pData;	"ESS_DTDATA_T" on page 136	Array of report data structures for given data cells
pulCount;	ESS_PULONG_T	Number of data blocks in the *pData* header information array

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

Rerieves an array of report header structures for the given drill-through instance handle.

Syntax

```
ESS_FUNC_M EssDTGetHeader (pDTInst, pBuffer, pulCount);
```

Parameter	Data Type	Description
pDTInst;	ESS_PDTHINST_T	Initialized drill-through instance handle
pBuffer;	"ESS_DTBUFFER_T" on page 136	An array of report header structures for the given columns
pulCount;	ESS_PULONG_T	Number of data blocks in the *pBuffer* report header information array

See Also

- Chapter 6, "Using the C Main API"
- "C Main API Structures" on page 114
- EssDTClose
- EssDTExit
- EssDTGetData
- EssDTGetHeaderInfo
- EssDTInit
- EssDTListReports
- EssDTOpen

EssDTGetHeaderInfo

Rtrieves report data header information for the given drill-through instance handle.

Syntax

```
ESS_FUNC_M EssDTGetHeaderInfo (pDTInst, ppHeader, pulCount);
```

Parameter	Data Type	Description
pDTInst	ESS_PDTHINST_T	Initialized drill-through instance handle
ppHeader	"ESS_DTHEADER_T" on page 137	Array of header information structures for the given drill-through instance handle
pulCount	ESS_PULONG_T	Number of blocks in the *ppHeader* header information array

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

Sarts a drill-through session, and returns a drill-through instance handle.

Syntax

```
ESS_FUNC_M EssDTInit (pInit, pDTInst);
```

Parameter	Data Type	Description
pDTInit;	ESS_PDTINIT_T	(Currently not used, and set to NULL.)
ppDTInst;	ESS_PDTHINST_T	Pointer to a drill-through initialization structure

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

Rturns a list of report names for the given drill-through instance handle.

Syntax

```
ESS_FUNC_M EssDTListReports (pDTInst, pBuffer, pulCount);
```

Parameter	Data Type	Description
pDTInst	ESS_PDTHINST_T	Initialized drill-through instance handle.
pBuffer	ESS_PSTR_T	Array of report names for the given drill-through instance handle.
pulCount	ESS_PULONG_T	Count of blocks in the pBuffer header information array.

See Also

- Chapter 6, "Using the C Main API"
- "C Main API Structures" on page 114
- `EssDTClose`
- `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

```
ESS_FUNC_M EssDTOpen (pDTInst, pEMC, ulCount, pConnection);
```

Parameter	Data Type	Description
pDTInst;	ESS_PDTHINST_T	Pointer to a drill-through initialization structure
pEMC;	ESS_PSTR_T	Extended Member Comment
ulCount;	ESS_ULONG_T	Number of Extended Member Comment blocks
pConnection;	ESS_PSTR_T	Connection information

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

```
ESS_FUNC_M EssDumpPerfStats (hCtx, pStatBuf, [thdSN])
```

Parameter	Data Type	Description
hCtx;	ESS_HCTX_T	API context handle (input)
pStatBuf;	ESS_STR_T	Pointer to the address where performance statistics tables will be dumped (input)
thdSN;	ESS_INT_T	**Optional.** Thread serial number from which to dump statistics (input). Default is 0 (all threads are dumped).

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

```
/* 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");
```

```
  }
  return(ESS_STS_NOERR);
}
```

See Also

- EssGetStatBufSize
- EssResetPerfStats

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

```
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);
```

```
      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

```
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

- 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

```
ESS_FUNC_M EssEndIncrementalBuildDim (hCtx, restructOption, szTmpOtlFile, ErrorName,
bOverwrite)
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	Essbase API context handle.
restructOption	ESS_SHORT_T	Restructure option. Valid value: • ESS_DOR_ALLDATA Keep all data • ESS_DOR_NODATA Discard all data • ESS_DOR_LOWDATA Keep all level 0 data • ESS_DOR_INDATA Keep all input data
szTmpOtlFile	ESS_STR_T	The temp outline file name.
ErrorName	ESS_STR_T	Name of error output file on client.
bOverwrite	ESS_BOOL_T	Indicates overwrite or append error message to the error file.

Return Value

Returns zero if successful; error code if unsuccessful.

Example

```
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,&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
- EssEndStreamBuildDim

EssEndReport

Mtion 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

```
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)
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	Essbase API context handle.
ErrorFileName	ESS_STR_T	Name of error output file on client.
ErFileOverWrite	ESS_BOOL_T	Indicates overwrite or append error message to the error file.

Return Value

Returns zero if successful; error code if unsuccessful.

Example

```
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,&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
- EssEndStreamBuildDim

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

```
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

```
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

ESS_FUNC_M **EssFixIBH** (*hCtx*, *action*);

Parameter	Data Type	Description
hCtx;	ESS_HCTX_T	Context handle
action;	ESS_IBH_ACTION	An enumeration type. For this release the only valid value is REMOVE.

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
hInstance	ESS_HINST_T	API instance handle.
pBlock	ESS_PVOID_T	Pointer to allocated memory block.

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_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

- EssAlloc
- EssInit
- EssOtlGetMemberCommentEx
- EssOtlSetMemberCommentEx
- EssRealloc

EssFreeMbrErr

Frees the memory allocated for a linked list of ESS_MBRERR_T structures.

Syntax

```
ESS_FUNC_M EssFreeMbrErr (hCtx, ppMbrErr);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
ppMbrErr	ESS_PMBRERR_T	Pointer to linked list contained in ESS_MBRERR_T.

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.

See Also

- EssImport

EssFreeStructure

Frees memory dynamically allocated by **EssGetAttributeInfo**() and **EssGetMemberInfo**() for string type attribute information.

Syntax

Parameter	Data Type	Description
hInst;	ESS_HINST_T	The instance handle of the process that called **EssGetAttributeInfo**() or **EssGetMemberInfo**() to allocate the structure.
structId;	ESS_ULONG_T	One of the following constant identifiers for the structure: • ESS_DT_STRUCT_ATTRIBUTEINFO • ESS_DT_STRUCT_ATTRSPECS • ESS_DT_STRUCT_MEMBERINFO
count;	ESS_ULONG_T	Number of structures
structPtr;	ESS_PVOID_T	Pointer to memory

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

```
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_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

```
ESS_FUNC_M EssGetActive (hCtx, pAppName, pDbName, pAccess);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
pAppName	ESS_PSTR_T	Address of pointer to receive allocated application name string.
pDbName	ESS_PSTR_T	Address of pointer to receive allocated database name string.
pAccess;	ESS_PACCESS_T	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.

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

```
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

- EssClearActive
- EssSetActive

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

- EssSetAlias

EssGetAPIVersion

Returns the version of the Essbase API used to compile the current application.

Syntax

```
ESS_FUNC_M EssGetAPIVersion (Version);
```

Parameter	Data Type	Description
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

```
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

```
ESS_FUNC_M EssGetApplicationAccess (hCtx, UserName, AppName, pCount, ppUserApp);
```

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.
AppName	ESS_STR_T	Application name. If NULL, lists all applications for the specified user.
pCount	ESS_PUSHORT_T	Address of variable to receive count of user application structures.
ppUserApp	"ESS_USERAPP_T, ESS_GROUPAPP_T" on page 189	Address of pointer to receive an allocated array of user application structures.

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

```
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\r\n",
                  UserApp[ind].UserName,
                  UserApp[ind].AppName,
                  UserApp[ind].Access,
                  UserApp[ind].MaxAccess);
         }
         EssFree (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);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle (input).
UserId	ESS_STR_T	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 application.
bIsIdentity	ESS_BOOL_T	Input. Indicates if *UserID* is a name or an identity. If TRUE, *UserID* is an identity.
type	ESS_USHORT_T	Type of entity (input). Indicates if *UserID* is a group or a user. Can be one of the following: • ESS_TYPE_USER • ESS_TYPE_GROUP
AppName	ESS_STR_T	Application name (input). If NULL, lists all applications for the specified user.
pCount	ESS_PUSHORT_T	Address of variable to receive count of user application structures (output).

Parameter	Data Type	Description
ppUserApp	ESS_PPUSERAPPEX_T	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.

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

```
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);
      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_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;
         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_PUSERAPPEX_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);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle (logged in)
AppName	ESS_STR_T	Application name
ppAppInfo	"ESS_APPINFO_T" on page 115	Address of pointer to receive allocated application info structure

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 ("\r\n-------Application Info-------\r\n\r\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 ("\r\n--List of Databases--\r\n\r\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);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle (logged in).
AppName	ESS_STR_T	Name of application for which to return information. If NULL, returns information for all applications.
pusCount	ESS_PUSHORT_T	Number of information structures returned.
ppAppInfoEx	"ESS_APPINFOEX_T" on page 116	Address of pointer to an array of allocated application info structures.

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

```
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("\n-----Application Info Ex -----\n\n");
         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);
      }
   }
```

```
   return (sts);
}
```

See Also

- EssGetApplicationInfo
- EssGetApplicationState
- EssGetDatabaseInfo

EssGetApplicationState

Gets an application's state structure, which contains user-configurable parameters for the application.

Syntax

```
ESS_FUNC_M EssGetApplicationState (hCtx, AppName, ppAppState);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
AppName	ESS_STR_T	Application name.
ppAppState	"ESS_APPSTATE_T" on page 117	Address of pointer to receive allocated application state structure.

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

```
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

Parameter	Data Type	Description
hCtx;	ESS_HCTX_T	Context handle
BaseMbrName;	ESS_STR_T	Base member name
AttrDimName;	ESS_STR_T	(Optional) attribute dimension name
pCount;	ESS_PULONG_T	Number of attribute members returned
ppAttrInfo;	"ESS_ATTRIBUTEINFO_T" on page 118	Attribute information

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

```
//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 ("\nAssociated 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)
```

```
                {
                   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

```
ESS_FUNC_M EssGetAsyncProcLog (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

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

```
ESS_FUNC_M EssGetAsyncProcState (hCtx, pBldDlState);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
pBldDlState	ESS_PBLDDL_STATE_T	Address of pointer to receive allocated process state structure.

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

Parameter	Data Type	Description
hCtx;	ESS_HCTX_T	Context handle
szAttributeName;	ESS_STR_T	Name of the attribute member or dimension
pAttributeInfo;	"ESS_ATTRIBUTEINFO_T" on page 118	Attribute information

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

```
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);
```

```
   /* sts = EssGetAttributeInfo(hCtx, "intro date", &pAttributeInfo); */
   /* sts = EssGetAttributeInfo(hCtx, "can", &pAttributeInfo); */
   /* sts = EssGetAttributeInfo(hCtx, "pkg type", &pAttributeInfo); */

   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

Parameter	Data Type	Description
hCtx;	ESS_HCTX_T	Context handle
pAttrSpecs;	"ESS_ATTRSPECS_T" on page 119	Attribute specifications

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

```
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_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
- EssFreeStructure
- EssGetAssociatedAttributesInfo
- EssGetAttributeInfo
- EssOtlAssociateAttributeDimension
- EssOtlAssociateAttributeMember
- EssOtlDisassociateAttributeDimension

- EssOtlDisassociateAttributeMember
- EssOtlFindAttributeMembers
- EssOtlFreeStructure
- EssOtlGetAssociatedAttributes
- EssOtlGetAttributeInfo
- EssOtlGetAttributeSpecifications
- EssOtlQueryAttributes
- EssOtlSetAttributeSpecifications

EssGetCalcList

Gets the list of calc scripts objects which are accessible to a user.

Syntax

```
ESS_FUNC_M EssGetCalcList (hCtx, UserName, AppName, DbName, pAllCalcs, pCount,
ppCalcList);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle
UserName	ESS_STR_T	User name
AppName	ESS_STR_T	Application name
DbName	ESS_STR_T	Database name
pAllCalcs	ESS_PBOOL_T	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.
pCount	ESS_PUSHORT_T	Address of variable to receive a count of the number of accessible calc script objects
ppCalcList	ESS_PPOBJNAME_T	Address of a pointer to receive an allocated array of calc script object names

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

```
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

```
ESS_FUNC_M EssGetCookie (ESS_HCTX_T, ESS_PPVOID_T);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
ppCookie	ESS_PPVOID_T	Address of a pointer to receive cookie.

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);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle
noMbrs	ESS_USHORT_T	Number of members in the member list *pMbrs*
pMbrs	ESS_PSTR_T	Pointer to the list of member names (or Aliases); the array size is assumed to be the dimension count
nURLXML	ESS_PUSHORT_T	Number of URL XMLs returned
ppURLXMLLen	ESS_PPUSHORT_T	Returns length of URL XML generated
ppURLXML	ESS_PPVOID_T	Returns pointers to the URL XML byte stream

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";
pMbrs[3] = "New York";
pMbrs[4] = "Sales";
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);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
pCount	ESS_PLONG_T	Address of variable to receive the count of rate info structures.
ppRateInfo	"ESS_RATEINFO_T" on page 179	Address of pointer to receive allocated array of currency rate info structures.

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

```
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\r\n", pRateInfoList[i].MbrName, rateStr);
            }
            else
            {
               printf ("Partition [%s] : %s\r\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

```
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       sts = 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   = "";

   sts = EssGetDatabaseAccess(hCtx, UserName,
            AppName, DbName, &Count, &UserDb);
   if(!sts)
   {
      if(Count && UserDb)
      {
printf ("\r\n-------Database Access List-------
                  \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("====================================\r\n");
         }
         EssFree (hInst, UserDb);
     }
     else
printf ("\r\nDatabase list is empty\r\n\r\n");
  }
  return (sts);
}
```

See Also

- EssGetDatabaseAccessEx
- EssGetApplicationAccess
- EssGetUser
- EssListUsers

- EssSetDatabaseAccess

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

```
ESS_FUNC_M EssGetDatabaseAccessEx (hCtx, UserId, bIsIdentity, type, AppName, DbName,
pCount, ppUserDb);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle (input).
UserId	ESS_STR_T	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.
bIsIdentity	ESS_BOOL_T	Input. Indicates if *UserID* is a name or an identity. If TRUE, *UserID* is an identity.
type	ESS_USHORT_T	Type of entity (input). Indicates if *UserID* is a group or a user. Can be one of the following: • ESS_TYPE_USER • ESS_TYPE_GROUP
AppName	ESS_STR_T	Application name (input). If NULL, lists all applications and databases for the specified user.
DbName	ESS_STR_T	Database name (input). 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 (output).
ppUserDb	ESS_PPUSERDBEX_T	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.

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

```
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;
        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;
        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_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);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
AppName	ESS_STR_T	Application name.
DbName	ESS_STR_T	Database name.
ppDbInfo	"ESS_DBINFO_T" on page 124	Address of pointer to receive allocated database info structure.

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);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
AppName	ESS_STR_T	Name of application for which to return database information. If NULL, returns information for all applications and databases.
DbName	ESS_STR_T	Name of database for which to return database information. If NULL, returns information for all databases.

Parameter	Data Type	Description
pusCount	ESS_PUSHORT_T	Number of information structures to be returned
ppDbInfo	"ESS_DBINFO_T" on page 124	Pointer to array of information structures.

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

```
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\n");
      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);
printf("---------------------------------\r\n\r\n");
      }
      EssFree(hInst, DbInfo);
   }
      return (sts);
}
```

See Also

- EssGetApplicationInfo
- EssGetDatabaseInfo
- EssGetDatabaseState
- EssGetDatabaseStats

EssGetDatabaseNote

Gets a database's note-of-the-day message.

Syntax

```
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);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle
AppName	ESS_STR_T	Application name
DbName	ESS_STR_T	Database name
ppDbState	"ESS_DBSTATE_T" on page 127	Address of pointer to receive allocated database state structure

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

```
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\r\n", pDbState->CrTypeMember);
            if (pDbState->CrConvType ==
                ESS_CRCTYPE_DIV)
printf ("Currency Conversion Type:             %s\r\n", "ESS_CRCTYPE_DIV");
            else if (pDbState->CrConvType ==
                     ESS_CRCTYPE_MULT)
printf ("Currency Conversion Type:             %s\r\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
```

```
                )
{
   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

```
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

```
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

```
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.

- TheDimDataType 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

```
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 --------\r\n\r\n");
      for(ind = 0; ind < nDims; ind++)
      {
printf("Dimension Name: %s\r\n",
        DimInfo[ind].DimName);
printf("Dimension Number: %d\r\n",
        DimInfo[ind].DimNumber);

         switch (DimInfo[ind].DimType)
         {
            case ESS_DIMTYPE_DENSE:
printf("Dimension Type: %s\r\n","DENSE");
               break;
            default:
printf("Dimension Type: %s\r\n","SPARSE");
               break;
         }
printf("\r\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

```
ESS_FUNC_M EssGetDrillThruURL (hCtx, URLName, &pUrl);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle
URLName	ESS_STR_T	Drill-through URL name
pUrl	ESS_PDURLINFO_T	URL definition

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

```
static void DisplayUrlDefn (ESS_PDURLINFO_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_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

```
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 recieve type of security in use.

Return Value

Returns 0 if successful; otherwise, returns an error.

Example

```
/*
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

```
ESS_FUNC_M EssGetExtUser (hCtx, UserName, ppExtUserInfo);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
UserName	ESS_STR_T	Name of user.
ppExtUserInfo	"ESS_USERINFOEX_T" on page 195	Address of pointer to receive allocated user info structure.

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

```
ESS_FUNC_M EssGetFilter (hCtx, AppName, DbName, FilterName, pActive, pAccess);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
AppName	ESS_STR_T	Application name.
DbName	ESS_STR_T	Database name.
FilterName	ESS_STR_T	Filter name.
pActive	ESS_PBOOL_T	Address of variable to receive filter active flag. If TRUE, the filter is currently in effect for the specified database.
pAccess	ESS_PACCESS_T	Address of variable to receive the default filter access level. For possible values, see "Bitmask Data Types (C)" on page 96.

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

```
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\r\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);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
AppName	ESS_STR_T	Application name.
DbName	ESS_STR_T	Database name.
FilterName	ESS_STR_T	Filter name.
pCount	ESS_PUSHORT_T	Address of variable to receive count of users assigned this filter.
ppUserList	ESS_PPUSERNAME_T	Address of pointer to receive allocated array of user names.

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

```
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\r\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

```
ESS_FUNC_M EssGetFilterRow (hCtx, pRowString, pAccess);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
pRowString	ESS_PSTR_T	Address of pointer to receive the next row of the filter.
pAccess	ESS_PACCESS_T	Address of variable to receive the access level for the filter row. Possible values are listed in "Bitmask Data Types (C)" on page 96.

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

```
ESS_FUNC_M EssGetGlobalState (hCtx, ppGlobal);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
ppGlobal	"ESS_GLOBAL_T" on page 140	Address of pointer to receive allocated global state structure.

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

```
ESS_FUNC_M
ESS_GetGlobalState (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");
    printf("Security->%d Logins->%d\r\n",
            pGlobal->Security, pGlobal->Logins);
    printf("Access->%ld 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

```
ESS_FUNC_M EssGetGroup (hCtx, GroupName, ppGroupInfo);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
GroupName	ESS_STR_T	Group name.
ppGroupInfo	"ESS_USERINFO_T, ESS_GROUPINFO_T" on page 192	Address of pointer to receive allocated group info structure (output).

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

```
ESS_FUNC_M EssGetGroupInfoEx (hCtx, GroupId, bisIdentity, ppGroupInfo);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle (input).
GroupId	ESS_STR_T	Group name (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.
ppGroupInfo	ESS_PGROUPINFOID_T	Address of pointer to receive allocated group info structure (output). The group list structure can include user directories and unique identity attributes.

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

```
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_DBALL\n", groupInfo.Access);
      break;
   case ESS_ACCESS_APPMANAGE:
      printf("\tAccess: %d - ESS_ACCESS_APPMANAGE\n", groupInfo.Access);
      break;
   case ESS_ACCESS_DBCREATE:
      printf("\tAccess: %d - ESS_ACCESS_DBCREATE\n", groupInfo.Access);
      break;
   case ESS_ACCESS_DBMANAGE:
      printf("\tAccess: %d - ESS_ACCESS_DBMANAGE\n", groupInfo.Access);
      break;
   case ESS_ACCESS_CALC:
      printf("\tAccess: %d - ESS_ACCESS_CALC\n", groupInfo.Access);
      break;
   case ESS_ACCESS_WRITE:
      printf("\tAccess: %d - ESS_ACCESS_WRITE\n", groupInfo.Access);
      break;
   case ESS_ACCESS_READ:
      printf("\tAccess: %d - ESS_ACCESS_READ\n", groupInfo.Access);
      break;
   case ESS_PRIV_USERCREATE:
      printf("\tAccess: %d - ESS_PRIV_USERCREATE\n", groupInfo.Access);
      break;
   case ESS_PRIV_APPCREATE:
      printf("\tAccess: %d - ESS_PRIV_APPCREATE\n", groupInfo.Access);
```

```
        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_DBALL\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);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
GroupName	ESS_USERNAME_T	User name or group name.
pCount	ESS_PUSHORT_T	Address of variable to receive count of user names.
ppUserList	ESS_PPUSERNAME_T	Address of pointer to receive allocated array of user name strings.

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 ("\r\n-------%s User List-------\r\n\r\n", GroupName);
         for (ind = 0; ind < Items; ind++)
         {
             if (UserList [ind])
                printf ("%s\r\n", UserList [ind]);
         }
         EssFree (hInst, UserList);
      }
      else
         printf ("\r\nUsers list is empty\r\n\r\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);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle (input).

Parameter	Data Type	Description
GroupName	ESS_STR_T	Group name or identity (input). Can be specified as `groupname@provider` or as a unique identity attribute.
bIsIdentity	ESS_BOOL_T	Input. Indicates if *GroupName* is a name or an identity. If TRUE, *GroupName* is an identity.
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 a 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

```
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

```
ESS_FUNC_M EssGetIBH (hCtx, destFileName);
```

Parameter	Data Type	Description
hCtx;	ESS_HCTX_T	Context handle
destFileName;	ESS_STR_T	Name of the file in which the IBH information is to be stored at client side.

See Also

- EssLocateIBH
- EssFixIBH

EssGetLocalPath

Gets the full local file path for a specific object file on the client.

Syntax

```
ESS_FUNC_M EssGetLocalPath (hCtx, ObjType, AppName, DbName, ObjName, Create, pPath);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle returned by **EssCreateLocalContext**()
ObjType	ESS_OBJTYPE_T	Object type (must be single type). See "Bitmask Data Types (C)" on page 96 for a list of object types.
AppName	ESS_STR_T	Application name or NULL (ESS_NULL). If NULL, command assumes a file name, and **EssGetLocalPath**() returns the *ObjName* in *pPath* as is.
DbName	ESS_STR_T	Database name. If NULL, uses the application subdirectory.
ObjName	ESS_STR_T	Object name or file name if *AppName* is NULL *ObjName* is not parsed for correctness; no suffix is appended to the path.
Create	ESS_BOOL_T	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.
pPath	ESS_PSTR_T	Address of pointer to receive allocated local path name string

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

```
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

```
ESS_FUNC_M EssGetLocationAliasList (hCtx, pusListCnt, ppAliasNames, ppHostNames,
ppAppNames, ppDbNames, ppUserNames);
```

Parameter	Data Type	Description
hCtx;	ESS_HCTX_T	API context handle
pusListCnt;	ESS_PUSHORT_T	Number of location aliases returned
ppAliasNames;	ESS_PSTR_T *	Location alias name buffer
ppHostNames;	ESS_PSTR_T *	Host name buffer
ppAppNames;	ESS_PSTR_T *	Application name buffer
ppDbNames;	ESS_PSTR_T *	Database name buffer
ppUserNames;	ESS_PSTR_T *	User login name buffer

Notes

- *hCtx* is the only input parameter.

- *pusListCnt*, *ppAliasNames*, *ppHostNames*, *ppAppNames*, *ppDbNames* and *ppUserNames* 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

```
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 *TimeStamp* 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

```
ESS_FUNC_M
ESS_GetLogFile (ESS_HCTX_T hCtx)
{
```

```
    ESS_FUNC_M     sts = ESS_STS_NOERR;
    ESS_STR_T     AppName;
    ESS_TIME_T    TimeStamp;
    ESS_STR_T     LocalName;

    AppName   = "Sample";
    LocalName = "C:\\Hyperion\\products\\Essbase\\EssbaseServer\\test.log";

    /* Get entire log file */
    TimeStamp = 0;

    sts = EssGetLogFile(hCtx, AppName, TimeStamp,
          LocalName);
    return (sts);
}
```

See Also

- EssDeleteLogFile
- EssLogSize
- EssWriteToLogFile

EssGetMemberCalc

Gets the calc equation for a specific member in the active database outline.

Syntax

```
ESS_FUNC_M EssGetMemberCalc (hCtx, MbrName, pCalcStr, pLastCalcStr);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
MbrName	ESS_STR_T	Member name.
pCalcStr	ESS_PSTR_T	Address of pointer to receive allocated member calc string.
pLastCalcStr	ESS_PSTR_T	Address of pointer to receive allocated 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 *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

```
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

```
ESS_FUNC_M EssGetMemberInfo (hCtx, MbrName, ppMbrInfo);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
MbrName	ESS_STR_T	Member name.
ppMbrInfo	"ESS_MEMBERINFO_T" on page 149	Address of pointer to receive allocated member information structure.

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 theStatus 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

```
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

```
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.

Parameter	Data Type	Description
LocalName	ESS_STR_T	Full path name of local destination file on client.
Lock	ESS_BOOL_T	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 Designer privilege (ESS_PRIV_APPDESIGN or ESS_PRIV_DBDESIGN) for the specified application or database containing the object.

Example

```
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_OBJTYPE_T     ObjType;
   ESS_STR_T         LocalName;
   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

```
ESS_FUNC_M EssGetObjectInfo (hCtx, ObjType, AppName, DbName, ObjName, ppObject);
```

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	Object name.
ppObject	ESS_PPOBJINFO_T	Address of pointer to receive allocated object info structure.

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

```
ESS_FUNC_M EssGetProcessState (hCtx, pProcState);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
pProcState	"ESS_PROCSTATE_T" on page 178	Pointer to process state structure

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

```
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))
```

```
        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

```
ESS_FUNC_M EssGetServerLocaleString (hCtx, localeString);
```

Parameter	Data Type	Description
hCtx;	ESS_HCTX_T	Context handle
localeString;	ESS_PSTR_T	Address of pointer to receive allocated string of server locale description.

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

```
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

```
ESS_FUNC_M EssGetServerMode(hCtx, *bUnicode);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle (logged in)
bUnicode	ESS_BOOL_T	The returned value, *bUnicode*, where *bUnicode* can be: • 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. • 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.

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

```
ESS_FUNC_M EssGetSpoolFile (hCtx, AppName, DbName, SplName, LocalName);
```

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 a specific spool file to return.
LocalName	ESS_STR_T	The new name of the spool file on the server.

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
- EssDeleteAllSplFiles
- 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);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
AppName	ESS_STR_T	Application name.
DbName	ESS_STR_T	Database name.
pSvrOutlineInfo	"ESS_SVROTLINFO_T" on page 710	Pointer to structure containing outline information stored on the Essbase Server.

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

Parameter	Data Type	Description
hCtx;	ESS_HCTX_T	API context handle
pBufSize;	ESS_PULONG_T	Pointer to the size of the buffer needed for the character array that will hold the performance statistics tables

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

```
ESS_FUNC_M EssGetString (hCtx, pString);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
pString	ESS_PSTR_T	Address of pointer to receive allocated returned data string.

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

```
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

```
ESS_FUNC_M EssGetUser (hCtx, UserName, ppUserInfo);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
UserName	ESS_STR_T	User name.
ppUserInfo	"ESS_USERINFO_T, ESS_GROUPINFO_T" on page 192	Address of pointer to receive allocated user info structure.

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

```
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

EssGetUserEx

Gets a user information structure, which contains security information for the user.

Syntax

```
ESS_FUNC_M EssGetUserEx (hCtx, UserName, ppUserInfo);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
UserName	ESS_STR_T	User name.
ppUserInfoEx	"ESS_USERINFOEX_T" on page 195	Address of pointer to receive info structure of externally authenticated user.

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

```
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);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle (input).
UserName	ESS_STR_T	User name (input). Can be specified as `username@provider` or as a unique identity attribute.
bIsIdentity	ESS_BOOL_T	Input. Indicates if *UserID* is a name or an identity. If TRUE, *UserID* is an identity.
ppUserInfo	ESS_PUSERINFOID_T	Address of pointer to receive allocated user info structure (output). The user list structure can include user directories and unique identity attributes.

Notes

The memory allocated for *ppUserpInfo* 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

```
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_DBALL\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_DBALL\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);
   //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);
   }
```

```
   return (sts);
}
```

See Also

- EssListUsersInfoEx

EssGetUserType

Enables you to find the application access type for a user.

Syntax

```
ESS_FUNC_M EssGetUserType (hCtx, UserName, UserType)
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	Essbase API context handle.
UserName	ESS_STR_T	Name of user.
UserType	ESS_PUSER_TYPE_T	Application access type defined for the UserName specified.

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);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	Context handle to the API.
pVariable	"ESS_VARIABLE_T" on page 197	The pointer to the structure containing the description of the specified substitution variable.

Return Value

If successful, **EssGetVariable**() returns the value of the substitution variable in the *VarValue* field of structure ESS_VARIABLE_T.

Example

```
/*
** ESS_GetVariable() 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\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");
    else
      printf("\n --> Error in EssGetVariable number: %d\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
```

```
Application name :
Database name    :
Variable value   : Market
 --> No Errors in EssGetVariable
```

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

```
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

```
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,
```

```
        &Revision);
   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);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
pRules	"ESS_OBJDEF_T" on page 151	Pointer to the rules file object definition structure.
pData	"ESS_OBJDEF_T" on page 151	Pointer to the data file object definition structure.
ppMbrErr	"ESS_MBRERR_T" on page 148	Pointer to linked list of errors contained in ESS_MBRERR_T. Possible errors are: • ESS_MBRERR_BADDIM • ESS_MBRERR_BADGEN • ESS_MBRERR_UNKNOWN • ESS_MBRERR_BADACCESS • ESS_MBRERR_BADSYNTAX
pMbrUser	"ESS_MBRUSER_T" on page 148	Pointer to the SQL user structure (if data source is a SQL database). A NULL SQL user structure indicates a non SQL data source.
abortOnError	ESS_USHORT_T	If TRUE, import stops on the first error. Otherwise, it continues.

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

```
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_PMBRERR_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_M EssImportASO (hCtx, pRules, pData, ppMbrErr, pUser, usabortOnError,
ulBufferId);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
pRules	"ESS_OBJDEF_T" on page 151	Pointer to the rules file object definition structure.
pData	"ESS_OBJDEF_T" on page 151	Pointer to the data file object definition structure.
ppMbrErr	"ESS_MBRERR_T" on page 148	Pointer to linked list of errors contained in ESS_MBRERR_T. Possible errors are: • ESS_MBRERR_BADDIM • ESS_MBRERR_BADGEN • ESS_MBRERR_UNKNOWN • ESS_MBRERR_BADACCESS • ESS_MBRERR_BADSYNTAX
pUser	"ESS_MBRUSER_T" on page 148	Pointer to the SQL user structure (if data source is a SQL database). A NULL SQL user structure indicates a non SQL data source.
usabortOnError	ESS_USHORT_T	If TRUE, import stops on the first error. Oherwise, it continues.
ulBufferId	ESS_ULONG_T	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.

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

```
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_DUPLICATES_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)
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	Essbase API context handle.
RulesObj	ESS_POBJDEF_T	Pointer to rules file object definition structure.
DataObj	ESS_POBJDEF_T	Pointer to data file object definition structure.
MbrUser	ESS_PMBRUSER_T	SQL user structure (if data source is SQL database). A NULL SQL user structure indicates a non SQL data source.

Parameter	Data Type	Description
ErrorName	ESS_STR_T	Name of error output file on client.
bOverwrite	ESS_BOOL_T	Indicates overwrite or append error message to the error file.
usBuildOption	ESS_USHORT_T	Valid values: • `ESS_INCDIMBUILD_BUILD` Build members only. • `ESS_INCDIMBUILD_VERIFY` Build members and verify the outline. • `ESS_INCDIMBUILD_SAVEOTL` Build members and save the outline to a temp outline file. • `ESS_INCDIMBUILD_ALL` Build members, verify the outline, and restructure.
szTmpOtlFile	ESS_STR_T	The temp outline file name. Essbasecreates a temporary outline file with extension "otb" if the resulting outline in this round of dimension build has outline verification errors.

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

```
ESS_FUNC_M EssInit (pInitStruct, phInstance);
```

Parameter	Data Type	Description
pInitStruct	"ESS_INIT_T" on page 141	Pointer to API initialization structure
phInstance	ESS_PHINST_T	Pointer to Essbase API instance handle

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 */
  {
    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 */
    NULL,      /* Reserved for internal use. */
               /* Set to NULL */
  };
/* Initialize the API  */
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

```
ESS_FUNC_M EssKillRequest (hCtx, pRequestInfoStruct);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
pRequestInfoStruct	ESS_PREQUESTINFO_T	Pointer to the Request Information structure.

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

```
#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 */
```

```
        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 = 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

```
ESS_FUNC_M EssKillRequestEx (hCtx, pRequestInfoStruct);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle (input).
pRequestInfoStruct	ESS_PREQUESTINFOEX_T	Pointer to the Request Information structure (input).

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

```
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] );

            index++;
        }

        EssFree ( hInst, requestInfo );
    }
}
```

See Also

- EssListRequestsEx

EssListAliases

Lists all the alias tables in the active database.

Syntax

```
ESS_FUNC_M EssListAliases (hCtx, pCount ppAliasList);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
pCount	ESS_PUSHORT_T	Address of variable to receive count of alias tables.
ppAliasList	ESS_PPALIASNAME_T	Address of pointer to receive an allocated array of alias table names.

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

```
ESS_FUNC_M
ESS_ListAliases (ESS_HCTX_T  hCtx,
                 ESS_HINST_T hInst
                )
{
   ESS_FUNC_M           sts = ESS_STS_NOERR;
   ESS_USHORT_T      Count;
   ESS_USHORT_T      ind;
   ESS_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

ESS_FUNC_M **EssListApplications** (*hCtx, pCount, ppAppList*);

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle
pCount	ESS_PUSHORT_T	Address of variable to receive count of returned applications
ppAppList	ESS_PPAPPNAME_T	Address of pointer to receive allocated array of application name strings

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

```
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 ("\r\nApplication 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

```
ESS_FUNC_M EssListCalcFunctions (hCtx, pCalcFunc);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
pCalcFunc	ESS_PSTR_T	Pointer to the string containing the available calculator functions. The string is in the form of XML.

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>
</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();
/* end forward declarations */

ESS_FUNC_M      MessageHandler (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 */)
{
   printf( "%d: %s\n", MessageNumber, MessageString );
   return 0;
}

void apiInit()
{
   ESS_STS_T    sts;
   ESS_INIT_T InitStruct = {
      ESS_API_VERSION,
      NULL,
      0L,
      255,
      NULL,
      NULL,
      NULL,
      NULL,
      NULL,
      (ESS_PFUNC_T)MessageHandler,
      NULL,
      0L
   };

   printf( "ESSBASE API v.%x\n", ESS_API_VERSION );

   if ((sts = EssInit(&InitStruct, &hInst)) != ESS_STS_NOERR)
   {
      printf( "API init failure: %d\n", sts);
      exit(1);
   }
```

```
}

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);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle
pCount	ESS_PUSHORT_T	Variable to receive count of users
ppUserList	"ESS_USERINFO_T, ESS_GROUPINFO_T" on page 192	Pointer to an array of user information structures. This array is allocated by the API

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

```
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_DBALL\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_DBALL\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);
   //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_PUSERINFOID_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

```
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("\r\n---- Currency Databases ----\r\n\r\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");
               }
            }
         }
         EssFree(hInst, pDbsList);
      }
      else
         printf("\r\nCurrency Database List is Empty\r\n\r\n");
   }
   return (sts);
}
```

See Also

- EssGetDatabaseInfo
- EssGetDatabaseState
- EssListApplications
- EssListDatabases
- EssListObjects

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);
```

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 applications and databases
ppDbList	"ESS_APPDB_T" on page 114	Address of pointer to receive allocated array of application/databasename structures

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\r\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);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle
AppName	ESS_STR_T	Use NULL. Function always applies to the currently selected database.
DbName	ESS_STR_T	Use NULL. Function always applies to the currently selected database.
pCount	ESS_PULONG_T	Address of variable to receive count of load buffers

Parameter	Data Type	Description
paLoadBuffers	"ESS_LOAD_BUFFER_T" on page 145**	Pointer to load buffer information structure

Return Value

Returns zero if successful; otherwise, returns an error code.

Example

```
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

Parameter	Data Type	Description
hCtx;	ESS_HCTX_T	Context handle
szAppName;	ESS_STR_T	Application name
szDbName;	ESS_STR_T	Database name
usFileType;	ESS_USHORT_T	One of the following file types to be returned: • ESS_FILETYPE_INDEX • ESS_FILETYPE_DATA • ESS_FILETYPE_INDEX | ESS_FILETYPE_DATA
pNmbrOfFiles;	ESS_PUSHORT_T	Pointer to the number of index and data files returned
ppDbInfoArray;	"ESS_DBFILEINFO_T" on page 123	Pointer to an array of database file information structures returned

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

```
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++ )
   {
      /*************************************************
       * Format and display the information in the current  *
       * persistent database file information array element *
       *************************************************/
      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

```
static void DisplayUrlDefn (ESS_PDURLINFO_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);
```

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 application.
Protocol	ESS_STR_T	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.
Count	ESS_PUSHORT_T	Address of variable to receive count of users.
ppUserList	"ESS_USERINFOEX_T" on page 195	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_USsERINFO_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.

See Also

- EssSetExtUser
- EssCreateExtUser
- EssGetExtUser

EssListFilters

Lists all filters for a database.

Syntax

```
ESS_FUNC_M EssListFilters (hCtx, AppName, DbName, Count, ppFilterList);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
AppName	ESS_STR_T	Application name.
DbName	ESS_STR_T	databasename.
pCount	ESS_PUSHORT_T	Address of variable to receive count of filter names.
ppFilterList	ESS_PPFTRNAME_T	Address of pointer to receive an allocated array of filter name strings.

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

```
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-------\r\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\r\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 (see "ESS_USERINFO_T, ESS_GROUPINFO_T" on page 192)	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

```
ESS_FUNC_M EssListGroups (hCtx, AppName, DbName, pCount, ppGroupList);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
AppName	ESS_STR_T	Application name. If NULL, lists all groups.
DbName	ESS_STR_T	databasename. If NULL, lists groups for all databases within application.
pCount	ESS_PUSHORT_T	Address of variable to receive count of groups.
ppGroupList	"ESS_USERINFO_T, ESS_GROUPINFO_T" on page 192	Address of pointer to receive an allocated array of group info structures.

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

```
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\r\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\r\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);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle (input).
AppName	ESS_STR_T	Application name (input). If NULL, lists all groups.
DbName	ESS_STR_T	Database name (input). If NULL, lists groups for all databases within the application.
pCount	ESS_PUSHORT_T	Address of variable to receive count of groups (output).
ppGroupList	ESS_PPGROUPINFOID_T	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.

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

```
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_DBALL\n", groupInfo.Access);
         break;
      case ESS_ACCESS_APPMANAGE:
         printf("\tAccess: %d - ESS_ACCESS_APPMANAGE\n", groupInfo.Access);
```

```
      break;
   case ESS_ACCESS_DBCREATE:
      printf("\tAccess: %d - ESS_ACCESS_DBCREATE\n", groupInfo.Access);
      break;
   case ESS_ACCESS_DBMANAGE:
      printf("\tAccess: %d - ESS_ACCESS_DBMANAGE\n", groupInfo.Access);
      break;
   case ESS_ACCESS_CALC:
      printf("\tAccess: %d - ESS_ACCESS_CALC\n", groupInfo.Access);
      break;
   case ESS_ACCESS_WRITE:
      printf("\tAccess: %d - ESS_ACCESS_WRITE\n", groupInfo.Access);
      break;
   case ESS_ACCESS_READ:
      printf("\tAccess: %d - ESS_ACCESS_READ\n", groupInfo.Access);
      break;
   case ESS_PRIV_USERCREATE:
      printf("\tAccess: %d - ESS_PRIV_USERCREATE\n", groupInfo.Access);
      break;
   case ESS_PRIV_APPCREATE:
      printf("\tAccess: %d - ESS_PRIV_APPCREATE\n", groupInfo.Access);
      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_DBALL\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_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

```
ESS_FUNC_M EssListLocks (hCtx, AppName, DbName, pCount, ppLockList);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.

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

```
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 ("\r\nExclusive Lock List on %s:%s is empty\r\n\r\n", AppName, DbName);
   }
   return (sts);
}
```

See Also

- EssListLocksEx
- EssListConnections

- EssListUsers
- EssRemoveLocks

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

```
ESS_FUNC_M EssListLocksEx (hCtx, AppName, DbName, pCount, ppLockList);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle (input).
AppName	ESS_STR_T	Application name (input).
DbName	ESS_STR_T	Database name (input).
pCount	ESS_PUSHORT_T	Address of variable to receive the user count (output).
ppLockList	ESS_PPLOCKINFOEX_T	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.

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

```
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);
```

```
   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("\nNumber 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

```
ESS_FUNC_M EssListLogins (hCtx, count, logins);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
count	ESS_PUSHORT_T	Pointer to the number of logins returned from the server.
logins	ESS_PPCONNECTINFO_T	Pointer to an array of a ESS_CONNECTINFO_T structure containing connection information.

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 log in instances in the current session. Similar to EssListLogins, but includes users hosted in a user directory.

Syntax

```
ESS_FUNC_M EssListLoginsEx (hCtx, count, logins);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle (input).
count	ESS_PUSHORT_T	Pointer to the number of log ins returned from the server (output).
logins	ESS_PPCONNECTINFOEX_T	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.

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

```
void DisplayLoginInfo(ESS_CONNECTINFOEX_T login)
{
```

```
   ESS_STS_T sts = ESS_STS_NOERR;

   printf("\tName: %s\n", login.Name);
   printf("\tApp Name: %s\n", login.AppName);
   printf("\tDb Name: %s\n", login.DbName);
   printf("\tLogin MachineName: %s\n", 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_PAPPDB_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);
```

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 (may be multiple types joined by bitwise OR (|)). 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, lists objects in the application subdirectory.
pCount	ESS_PUSHORT_T	Address of variable to receive the count of objects of the appropriate type(s).
ppObjList	"ESS_OBJINFO_T" on page 152	Address of pointer to receive allocated array of object info structures.

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

```
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);
```

```
   if (!sts)
   {
      if (objCnt && pObject)
      {
         pNextObject = pObject;
         for (objInd = 0; objInd < objCnt; objInd++)
         {
             if (pNextObject)
             {
printf ("Name: %s \r\nUser: %s\r\nTime Stamp: %ld\r\n",
                          pNextObject->Name,
                          pNextObject->User,
                          pNextObject->TimeStamp);

                pNextObject = pNextObject + 1;
             }
         }
         EssFree (hInst, pObject);
      }
      else
printf ("\r\nObject List is Empty\r\n\r\n");
   }
   return(sts);
}
```

See Also

- EssGetObject
- EssGetObjectInfo

EssListRequests

Returns information about active sessions and requests.

Syntax

ESS_FUNC_M **EssListRequests** (*hCtx, UserName, AppName, DbName, RequestCount, pRequestInfo*);

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
UserName	ESS_STR_T	User name
AppName	ESS_STR_T	Application name
DbName	ESS_STR_T	Database name
RequestCount	ESS_PUSHORT_T	Number of requests (output)
ppRequestInfoStruct	ESS_PPREQUESTINFO_T	Request type (output)

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

```
#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_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 */
        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 );
            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

- 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);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle (input).
UserId	ESS_STR_T	User name or identity (input). If an identity, includes a unique identity string identifying the user in a user directory.
bIsIdentity	ESS_BOOL_T	Indicates if a user identity or name is used (input). If TRUE, indicates that *UserId* is a unique identity attribute. If FALSE, *UserId* is a user name.
AppName	ESS_STR_T	Application name (input).
DbName	ESS_STR_T	Database name (input).
RequestCount	ESS_PUSHORT_T	Number of requests (output).
pRequestInfo	ESS_PREQUESTINFOEX_T	Request type (output).

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

```
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] );
```

```
            index++;
        }

        EssFree ( hInst, requestInfo );
    }
}
```

See Also

- EssKillRequestEx

EssListSpoolFiles

Lists all trigger log files for a database.

Syntax

```
ESS_FUNC_M EssListSpoolFiles (hCtx, AppName, DbName, pCount, ppFileList);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
AppName	ESS_STR_T	Application name.
DbName	ESS_STR_T	Database name.
pCount	ESS_PUSHORT_T	Address of variable to receive count of spool file names.
ppFileList	ESS_PPOBJINFO_T	Address of pointer to receive an allocated array of spool file name objects.

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

```
ESS_FUNC_M EssListSSMigrFailedGroups (hCtx,  count, pNativeUserList);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
count	ESS_PUSHORT_T	Address of variable to receive count of migration failed groups.
pNativeUserList	ESS_PPUSERNAME_T	Address of pointer to receive an allocated array of migration failed groups.

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

```
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");
   }
   else
      printf("Failed to get Shared Services migration failed Groups list.\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);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
count	ESS_PUSHORT_T	Address of variable to receive count of migration failed users.
pNativeUserList	ESS_PPUSERNAME_T	Address of pointer to receive an allocated array of migration failed users.

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          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\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");
```

```
   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

```
ESS_FUNC_M EssListTransactions(hCtx, TimeSrc, InpTime,
            ListOption, FileName, pCount, ppResults);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	Login context must be set active before calling this API.
TimeSrc	ESS_USHORT_T	The option that specifies where to get the start time for display transactions.
InpTime	ESS_TIME32_T	Input the time if TimeSrc is ESS_TRLOG_TIMESPECIFIED.The time is a ULONG representing the number of seconds since January 1, 1970.
ListOption	ESS_USHORT_T	The option that specifies the destination of the output. See "List Option Constants (C)" on page 105
FileName	ESS_STR_T	If ListOption is either of the LIST_TRANSACTIONS_* options, then the content is written to this file on the server machine: • You can enter a full path. • Default: `$ARBORPATH/app`
pCount	ESS_PULONG_T	Number of entries returned
ppResults	ESS_PPTRANSACTION_ENTRY_T	The entries returned if ListOption is ESS_LIST_TRANSACTIONS_TOCLIENT

Return Value

- 0—If successful
  - `pCount` contains the number of returned entries
  - `ppResults` contains the returned entries if ListOptions 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

```
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_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);

        /* 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;
```

```
    sts = EssReplayTransactions (hCtx, AppName, DbName,
                                                    ReplayInp, pSeqIds);
    printf("EssReplayTransactions sts: %ld\r\n",sts);
    printf("\n\n");

    /* 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,
                                          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_LASTREPLAYTIME;
    sts = EssReplayTransactions (hCtx, AppName,
                        DbName, ReplayInp, pSeqIds);
    printf("EssReplayTransactions sts: %ld\r\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 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        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-------\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);
          }
          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);
```

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 application.
SecurityProvider	ESS_STR_T	Name of the external authentication mechanism.
pCount	ESS_PUSHORT_T	Address of variable to receive count of users.
ppUserList	"ESS_USERINFOEX_T" on page 195	Address of pointer to receive an allocated array of user info structures. The *AppName* and *DbName* fields of the returned user info structures contain 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        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\n-------User List-------\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);
         }
         EssFree (hInst, Users);
      }
      else
printf ("\r\nUsers list is empty\r\n\r\n");
   }
   return (sts);
}
```

See Also

- EssGetUser
- EssListConnections
- EssListGroups
- 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);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle (input).
AppName	ESS_STR_T	Application name (input). If NULL, lists all users.
DbName	ESS_STR_T	Database name (input). If NULL, lists users for all databases within the application.
pCount	ESS_PUSHORT_T	Address of variable to receive count of users (output).
ppUserList	ESS_PPUSERINFOID_T	Address of pointer to receive an allocated array of user info structures (output). The user list structure can include user directories and unique identity attributes.

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

```
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_DBALL\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_DBALL\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);
   //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_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

```
ESS_FUNC_M EssListVariables (hCtx, pCriteria, pNumVars, ppVarList);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	Context handle to the API.
pCriteria	"ESS_VARIABLE_T" on page 197	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."
pNumVars	ESS_PULONG_T	The pointer to an unsigned long value indicating the number of variables being returned in the *ppVarList* parameter.
ppVarList	"ESS_VARIABLE_T" on page 197	The pointer to an array of substitution variable structures. It is the responsibility of the caller to free this array by calling EssFree.

Return Value

If successful, returns zero.

Example

```
/*
** 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_PVARIABLE_T   pVariables;
   ESS_ULONG_T       ulCount, i;
```

```
ESS_VARIABLE_T    Variable;
printf("\n ****************************************");
printf("\n **** An example of using EssListVariables");
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\n\n\n");
   else
      printf("\n --> Error in EssListVariables number: %d\n\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 : 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 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

```
ESS_FUNC_M EssLoadAlias (hCtx, AliasName, FileName);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
AliasName	ESS_STR_T	Name of alias table to load.
FileName	ESS_STR_T	Full path name of structured alias names file on the server.

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

```
ESS_FUNC_M EssLoadBufferInit (hCtx, AppName, DbName, ulBufferId,
ulDuplicateAggregationMethod,
ulOptionFlags, ulSize);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
AppName	ESS_STR_T	Name of the application for which to create the load buffer.
DbName	ESS_STR_T	Name of the database for which to create the load buffer.
ulBufferId	ESS_ULONG_T	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.

Parameter	Data Type	Description

ulDuplicateAggregationMethod	ESS_ULONG_T	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.

Parameter	Data Type	Description

ulOptionFlags	ESS_ULONG_T	One or more of the following load buffer options:
		• ESS_ASO_DATA_LOAD_BUFFER_IGNORE_MISSING_VALUES: Ignores #MISSING values in the incoming data stream.
		`#define ESS_ASO_DATA_LOAD_BUFFER_IGNORE_MISSING_VALUES 0x00000001`
		• ESS_ASO_DATA_LOAD_BUFFER_IGNORE_ZERO_VALUES: Ignores zeros in the incoming data stream.
		`#define ESS_ASO_DATA_LOAD_BUFFER_IGNORE_ZERO_VALUES 0x00000002`
		• ESS_ASO_DATA_LOAD_BUFFER_WAIT_FOR_RESOURCES: 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.
		`#define ESS_ASO_DATA_LOAD_BUFFER_WAIT_FOR_RESOURCES 0x00000004`
		Use bitwise OR (|) to specify multiple `ulOptions`; for example:
		`ESS_ASO_DATA_LOAD_BUFFER_IGNORE_MISSING_VALUES | ESS_ASO_DATA_LOAD_BUFFER_IGNORE_ZERO_VALUES`

ulSize	ESS_ULONG_T	Percentage of total load buffer resources this load buffer may use. Possible values: 0 to 100. For a value of 0, Essbase uses a self-determined, default load buffer size. If the total size of all load buffers exceeds 100, the operation fails.

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

```
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

- EssBeginDataloadASO
- EssSendString
- EssEndDataload
- EssLoadBufferTerm
- EssImportASO
- EssUpdateFileASO
- EssUpdateFileUTF8ASO
- EssListExistingLoadBuffers

- EssMergeDatabaseData

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

```
ESS_FUNC_M EssLoadBufferTerm (hCtx, AppName, DbName, ulBufferCnt, *ulBufferIdAry,
ulCommitType,
ulActionType, ulOptions);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
AppName	ESS_STR_T	Name of the application.
DbName	ESS_STR_T	Name of the database.
ulBufferCnt	ESS_ULONG_T	Number of buffers in the list.
*ulBufferIdAry	ESS_ULONG_T	Array of buffer IDs that will be affected by this operation.

Parameter	Data Type	Description

ulCommitType	ESS_ULONG_T	One of the following constants for combining the values stored in the buffer with the values already stored in the database:
		• 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`
		When using the abort option, the ulCommitType and ulOptions settings are ignored

Parameter	Data Type	Description

ulOptions	ESS_ULONG_T	One of the following constants:
		• ESS_ASO_DATA_LOAD_INCR_TO_MAIN_SLICE: Write the data currently stored in the buffer to the main slice in the database.
		`#define ESS_ASO_DATA_LOAD_INCR_TO_MAIN_SLICE 0`
		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.
		• 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.
		`#define ESS_ASO_DATA_LOAD_INCR_TO_NEW_SLICE 1`
		• 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).
		`#define ESS_ASO_DATA_LOAD_INCR_TO_NEW_SLICE_LIGHTWEIGHT 2`
		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

```
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
- EssBeginDataloadASO
- EssSendString
- EssEndDataload
- EssImportASO
- EssUpdateFileASO
- EssUpdateFileUTF8ASO
- EssListExistingLoadBuffers
- EssMergeDatabaseData

EssLoadDatabase

Starts a database within an Application on the server.

Syntax

```
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

```
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

```
ESS_FUNC_M EssLocateIBH (hCtx, dbName);
```

Parameter	Data Type	Description
hCtx;	ESS_HCTX_T	API context handle.
dbName;	ESS_STR_T	Name of the database.

See Also

- EssFixIBH
- EssGetIBH

EssLockObject

Locks an object on the server or the client object system to prevent other users from updating it.

Syntax

```
ESS_FUNC_M EssLockObject (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 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. 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

```
ESS_FUNC_M
ESS_LockObject (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\r\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);
```

Parameter	Data Type	Description
hInstance	ESS_HINST_T	API instance handle.

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,
		`https://myhost:13080/aps/Essbase?clustername=Essbase-Cluster1&SecureMODE=Yes`

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

```
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\r\n", pAppsDbs[ind].AppName);
printf ("%s\r\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

```
ESS_FUNC_M EssLoginAs (hInstance, Server, UserName, Password, UserNameAs, pDbCount,
ppDbList, phCtx);
```

Parameter	Data Type	Description
hInstance	ESS_HINST_T	API instance handle.

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,
		`https://myhost:13080/aps/Essbase?clustername=Essbase-Cluster1&SecureMODE=Yes`

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`

- EssLogout
- EssSetActive

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);
```

Parameter	Data Type	Description
hInstance	ESS_HINST_T	API instance handle.

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,
		`https://myhost:13080/aps/Essbase?clustername=Essbase-Cluster1&SecureMODE=Yes`

Token	ESS_STR_T	The token representing the username and password of an authenticated user.
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 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

```
ESS_FUNC_M EssLoginExAs (hInstance, Server, Token, UserNameAs, pDbCount, ppDbList,
phCtx);
```

Parameter	Data Type	Description
hInstance	ESS_HINST_T	API instance handle.

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,
		`https://myhost:13080/aps/Essbase?clustername=Essbase-Cluster1&SecureMODE=Yes`

Token	ESS_STR_T	The token representing the user name and password of an authenticated user.
UserNameAs	ESS_STR_T	User name string for the user you want to impersonate.

Parameter	Data Type	Description
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 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

Parameter	Data Type	Description
HInstance	ESS_HINST_T	API instance handle.

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,
		`https://myhost:13080/aps/Essbase?clustername=Essbase-Cluster1&SecureMODE=Yes`

UserName	ESS_STR_T	User name.
Password	ESS_STR_T	Old password.
NewPassword	ESS_STR_T	New password.
pDbCount	ESS_PUSHORT_T	Number of accessible databases.
ppDbList	ESS_PPAPPDB_T	Address of the pointer to an array of accessible application-database structures.
phCtx	ESS_PHCTX_T	Pointer to the context handle.

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

```
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
hCtx	ESS_HCTX_T	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 **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

```
ESS_FUNC_M
ESS_Logout (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

```
ESS_FUNC_M EssLogoutUser (hCtx, LoginId);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle of user forcing the log out.
LoginId	ESS_LOGINID_T	Login ID of user to be logged out.

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

```
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_ULONG_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
- EssWriteToLogFile

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

```
ESS_STS_T  ESS_LROAddObject (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;
```

```
        ESS_ULONG_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

```
ESS_FUNC_M EssLRODeleteCellObjects (hCtx, memCount, pMemComb, pulLROCount, pLRODescList);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.

Parameter	Data Type	Description
memCount	ESS_ULONG_T	Number of members specified in *pMemComb*.
pMemComb	ESS_PVOID_T	Array of member names.
pulLROCount	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

```
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);
```

```
        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_PLRODESC_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
- EssLRODeleteObject

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)*

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
memCount	ESS_ULONG_T *	Array of 'Number of members' specified in *pMemComb,*, one for each cell.
pMemComb	ESS_PMBRNAMECOMB_T *	Array of 'member name' combination. Each element of array itself is an array of member names, one for each cell.
cellCount	ESS_ULONG_T	Count of LRO cells.
pulLROCount	ESS_ULONG_T *	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.
pLRODescList	"ESS_LRODESC_API_T" on page 108	Address of pointer to the list of LRO catalog entries returned to caller.

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             mbrsCount[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, mbrsCount[0] * memberLength, (ESS_PPVOID_T)&(ppMemComb[0]));
   if (status)
      goto exit;
   pMemComb = ppMemComb[0];
   memset(pMemComb, 0, mbrsCount[0]* memberLength);
   strcpy((pMemComb)[0], "Jan");
   strcpy((pMemComb)[1], "Sales");
   strcpy((pMemComb)[2], "100-10");
   strcpy((pMemComb)[3], "New York");
   strcpy((pMemComb)[4], "Actual");
```

```
   /* 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 inormation 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(pulLroCount, 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<pulLroCount[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\n",(pLroDesc+i)->lro.note);
            }
            printf("User name - %s\n", (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 (pulLroCount)
         EssFree(hInst, pulLroCount);

      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

```
ESS_FUNC_M EssLROGetObject (hCtx, plinkId, targetFile, usOption,
pRetLRODesc);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
plinkId ;	"ESS_LROHANDLE_API_T" on page 109	Pointer to object identification structure.
targetFile	ESS_STR_T	The name of the target file into which the object is retrieved.
usOption	ESS_USHORT_T	Option specifying whether to retrieve the object, its catalog entry, or both. Use one of the following: • ESS_LRO_OBJ_API retrieves only the object. • ESS_LRO_CATALOG_API retrieves only the catalog entry. • ESS_LRO_BOTH_API retrieves object and catalog entry.
pRetLRODesc	"ESS_LRODESC_API_T" on page 108	Pointer to object's description structure..

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

```
ESS_FUNC_M ESS_LRO GetObject (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

```
ESS_FUNC_M EssLROListObjects (hCtx, userName, listDate, pulLROCount, pLRODescList));
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
userName	ESS_CHAR_T	A user name. If specified, returns a list of all objects last modified by the given user.
listDate	ESS_TIME_T	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.
pulLROCount	ESS_ULONG_T *	Number of LRO catalog entries returned.
pLRODescList;	"ESS_LRODESC_API_T" on page 108	Address of pointer to the list of LRO catalog entries returned.

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

```
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
- EssLROPurgeObjects

EssLROPurgeObjects

Deletes all objects linked to cells in the active database for a given user name and/or modification date.

Syntax

```
ESS_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

```
ESS_FUNC_M ESS_LRO PurgeObjects (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              purgeDate[ESS_DATESIZE];
        ESS_CHAR_T              buf[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);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
plinkId	"ESS_LROHANDLE_API_T" on page 109	Pointer to object identification structure.
usOption	ESS_USHORT_T	Option specifying whether to store the object, its catalog entry, or both. Use one of the following: • ESS_LRO_OBJ_API stores only the object. • ESS_LRO_CATALOG_API stores only the catalog entry. • ESS_LRO_BOTH_API stores the object and the catalog entry.
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 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

```
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;

        /* Update both object and catalog */
        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

- EssLROGetObject
- EssLROAddObject
- EssLRODeleteObject

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

```
ESS_FUNC_M EssMdxTrig (hCtx, AppName, DbName, mdxStatement);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
AppName	ESS_STR_T	Application name.
DbName	ESS_STR_T	Database name.
mdxStatement	ESS_STR_T	An MDX statement that specifies whether to create, replace, delete, enable, or disable a specific trigger.

Access

This function requires the caller to have database Design privilege (ESS_PRIV_DBDESIGN) for the specified database.

See Also

- EssDisplayTriggers
- EssListSpoolFiles
- EssGetSpoolFile
- EssDeleteSplFile
- EssDeleteAllSplFiles

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

```
ESS_FUNC_M EssMergeDatabaseData (hCtx, AppName, DbName, ulOptions);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle
AppName	ESS_STR_T	Use NULL. Function always applies to the currently selected database.

Parameter	Data Type	Description
DbName	ESS_STR_T	Use NULL. Function always applies to the currently selected database.
ulOptions	ESS_ULONG_T	One of the following constants: • `#define ESS_MERGE_DATABASE_DATA_ALL 1`: Merges all data slices into one. • `#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.

Return Value

Returns zero if successful; otherwise, returns an error code.

Example

```
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("\nCreate the buffer:\n");
        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_slice1a";
```

```
        isAbortOnError = ESS_TRUE;

        printf("\nLoad 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("\nMerge 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 compatability. For complete information, see EssPartitionApplyOtlChangeFileEx.

Syntax

```
ESS_FUNC_M EssPartitionApplyOtlChangeFile (hCtx, usFileName, ppszFileName);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle
usFileName	ESS_USHORT_T	Number of outline change files
ppszFileName	ESS_PSTR_T	Array of file names; array size is defined by usFileName

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

```
ESS_FUNC_M EssPartitionApplyOtlChangeFileEx (hCtx, usFileName, ppszFileName,
usDataDirectionType);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	Handle to Essbase API context
usFileName	ESS_USHORT_T	Number of outline change files
ppszFileName	ESS_PSTR_T	Array of file names; array size is defined by usFileName

usDataDirectionType	ESS_USHORT_T	One of the following Direction Type constants:
		`#define ESS_PARTITION_DATA_SOURCE 0x0001`
		`#define ESS_PARTITION_DATA_TARGET 0x0002`

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

```
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 = "app1";
   dbname = "src1";
   usType = ESS_PARTITION_OP_LINKED;
   dataFlowDir = ESS_PARTITION_DATA_SOURCE;
   uldimfilter     = ESS_DIMCHG_ALL;
   ulmbrfilter     = ESS_PARTITION_OTLMBR_ALL;
   ulmbrattrfilter = ESS_PARTITION_OTLPARTITION_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);

      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

- EssPartitionFreeOtlChanges
- EssPartitionGetAreaCellCount
- EssPartitionGetList
- EssPartitionGetOtlChanges
- EssPartitionGetReplCells
- EssPartitionNewDefFile
- EssPartitionOpenDefFile
- EssPartitionPurgeOtlChangeFile
- EssPartitionPutReplCells
- EssPartitionReadDefFile
- EssPartitionReadOtlChangeFile
- EssPartitionReplaceDefFile
- EssPartitionResetOtlChangeTime
- EssPartitionValidateDefinition
- EssPartitionWriteDefFile

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);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	Handle to API context.
pApplyRecords	"ESS_PARTOTL_SELECT_APPLY_T" on page 171	Records to apply.

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

```
ESS_FUNC_M  Ess_PartitionApplyOtlChangeRecs (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_OTLPARTITION_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("\tEssPartitionReadOtlChangeFile  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

- EssPartitionGetReplCells
- EssPartitionNewDefFile
- EssPartitionOpenDefFile
- EssPartitionPurgeOtlChangeFile
- EssPartitionPutReplCells
- EssPartitionReadDefFile
- EssPartitionReadOtlChangeFile
- EssPartitionReplaceDefFile
- EssPartitionResetOtlChangeTime
- EssPartitionValidateDefinition
- EssPartitionWriteDefFile

EssPartitionCloseDefFile

Closes the shared partition definition file.

Syntax

```
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

- EssPartitionGetOtlChanges
- EssPartitionGetReplCells
- EssPartitionNewDefFile
- EssPartitionOpenDefFile
- EssPartitionPurgeOtlChangeFile
- EssPartitionPutReplCells
- EssPartitionReadDefFile
- EssPartitionReadOtlChangeFile
- EssPartitionReplaceDefFile
- EssPartitionResetOtlChangeTime
- EssPartitionValidateDefinition
- EssPartitionValidateLocal
- EssPartitionWriteDefFile

EssPartitionFreeDefCtx

Frees memory dynamically allocated under shared-partition context structures.

Syntax

```
ESS_FUNC_M EssPartitionFreeDefCtx (hCtx, pDdbCtx);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	Api context.
pDdbCtx	"ESS_PART_T" on page 152	Pointer to shared-partition context.

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
- EssPartitionGetOtlChanges
- EssPartitionGetReplCells
- EssPartitionNewDefFile
- EssPartitionOpenDefFile
- EssPartitionPurgeOtlChangeFile
- EssPartitionPutReplCells
- EssPartitionReadDefFile
- EssPartitionReadOtlChangeFile
- EssPartitionReplaceDefFile
- EssPartitionResetOtlChangeTime
- EssPartitionValidateDefinition
- EssPartitionValidateLocal
- EssPartitionWriteDefFile

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
- 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

- EssPartitionGetList
- EssPartitionGetOtlChanges
- EssPartitionGetReplCells
- EssPartitionNewDefFile
- EssPartitionOpenDefFile
- EssPartitionPurgeOtlChangeFile
- EssPartitionPutReplCells
- EssPartitionReadDefFile
- EssPartitionReadOtlChangeFile
- EssPartitionReplaceDefFile
- EssPartitionResetOtlChangeTime
- EssPartitionValidateDefinition
- EssPartitionWriteDefFile

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);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
pSelectPartition	"ESS_PARTSLCT_T" on page 171	Criteria to select partitions.
pusCount	ESS_PUSHORT_T	Count of partitions returned.
ppPartition	"ESS_PART_INFO_T" on page 154	Pointer to allocated array of partition information structures.

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;
   memset(&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\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",
              info->MetaDirection==ESS_PARTITION_OTL_SOURCE ? "(Outline Change Source)":
              info->MetaDirection==ESS_PARTITION_OTL_TARGET ? "(Outline Change
Target)" :"(Unknown Outline Change Type)",
              info->SvrName, info->AppName, info->DbName);
      printf("    Outline last changed: %s\n",
                info->LastMetaUpdateTime==0 ? "Never\n"
                                            : 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\n"
                                         : 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\n",
                info->LastUpdateTime==0 ? "Never\n"
                                        : ctime(&info->LastUpdateTime),
                info->IncrRefreshAllowed ? "Incrementally replicatable" :
```

```
                                                "Not incrementally replicatable");
      }
   }#/* end for */
   }#/* end if */
   if (partitionp) EssFree(hInst, partitionp);
   return(sts);
}
```

See Also

- "Constant and Structure Definitions for Partitions (C)" on page 109
- EssPartitionApplyOtlChangeFile
- EssPartitionApplyOtlChangeRecs
- EssPartitionCloseDefFile
- EssPartitionFreeDefCtx
- EssPartitionFreeOtlChanges
- EssPartitionGetAreaCellCount
- EssPartitionGetOtlChanges
- EssPartitionGetReplCells
- EssPartitionNewDefFile
- EssPartitionOpenDefFile
- EssPartitionPurgeOtlChangeFile
- EssPartitionPutReplCells
- EssPartitionReadDefFile
- EssPartitionReadOtlChangeFile
- EssPartitionReplaceDefFile
- EssPartitionResetOtlChangeTime
- EssPartitionValidateDefinition
- EssPartitionWriteDefFile

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

```
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=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 = "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

- EssPartitionFreeOtlChanges
- EssPartitionGetAreaCellCount
- EssPartitionGetList
- EssPartitionGetReplCells
- EssPartitionNewDefFile
- EssPartitionOpenDefFile
- EssPartitionPurgeOtlChangeFile
- EssPartitionPutReplCells
- EssPartitionReadDefFile
- EssPartitionReadOtlChangeFile
- EssPartitionReplaceDefFile
- EssPartitionResetOtlChangeTime
- EssPartitionValidateDefinition
- EssPartitionWriteDefFile

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
- EssPartitionFreeOtlChanges
- EssPartitionGetAreaCellCount
- EssPartitionGetList
- EssPartitionGetOtlChanges
- EssPartitionNewDefFile
- 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

ESS_FUNC_M **EssPartitionNewDefFile** (*hCtx, pszFileName, pHostDatabase, piFileHandle, ppDdbCtx*);

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API network context.
pszFileName	ESS_STR_T	Name of file to be created (full path).
pHostDatabase	"ESS_PART_CONNECT_INFO_T" on page 153	Identifies the host database.
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_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 = EssPartitionNewDefFile(hCtx,pszFileName,&HostDatabase,&iFileHandle,&pDdbCtx);
   printf("EssPartitionNewDefFile  sts: %ld\n",sts);

   if (!sts)
   {

       /* ...
          ...  process definition file information
          ...
       */
           sts = EssPartitionWriteDefFile(hCtx,iFileHandle,pDdbCtx);

       printf("\tEssPartitionWriteDefFile   sts: %ld\n",sts);

           sts = EssPartitionCloseDefFile(hCtx,iFileHandle);

       printf("\tEssPartitionCloseDefFile   sts: %ld\n",sts);

           sts = EssPartitionFreeDefCtx(hCtx,pDdbCtx);

       printf("\tEssPartitionFreeDefCtx   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
- EssPartitionOpenDefFile
- EssPartitionPurgeOtlChangeFile
- EssPartitionPutReplCells
- EssPartitionReadDefFile
- EssPartitionReadOtlChangeFile
- EssPartitionReplaceDefFile
- EssPartitionResetOtlChangeTime
- EssPartitionValidateDefinition
- EssPartitionValidateLocal
- EssPartitionWriteDefFile

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\\app1\\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
```

```
            ...
        */
            sts = EssPartitionCloseDefFile(hCtx,iFileHandle);

        printf("\tEssPartitionCloseDefFile   sts: %ld\n",sts);

            sts = EssPartitionFreeDefCtx(hCtx,pDdbCtx);

        printf("\tEssPartitionFreeDefCtx   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
- EssPartitionPurgeOtlChangeFile
- EssPartitionPutReplCells
- EssPartitionReadDefFile
- EssPartitionReadOtlChangeFile
- EssPartitionReplaceDefFile
- EssPartitionResetOtlChangeTime
- EssPartitionValidateDefinition
- EssPartitionValidateLocal
- EssPartitionWriteDefFile

EssPartitionPurgeOtlChangeFile

Purges changes made previous to the time specified with the *TimeStamp* parameter.

Syntax

```
ESS_FUNC_M EssPartitionPurgeOtlChangeFile (hCtx, pPartition, TimeStamp);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
pPartition	"ESS_PART_DEFINED_T" on page 153	Partition specification
TimeStamp	ESS_TIME_T	Purge all change records before this time.

Return Value

Returns zero if successful, error code if unsuccessful.

Example

```
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

- EssPartitionGetAreaCellCount
- EssPartitionGetList
- EssPartitionGetOtlChanges
- EssPartitionGetReplCells
- EssPartitionNewDefFile
- EssPartitionOpenDefFile
- EssPartitionPurgeOtlChangeFile
- EssPartitionReadDefFile
- EssPartitionReadOtlChangeFile
- EssPartitionReplaceDefFile
- EssPartitionResetOtlChangeTime
- EssPartitionValidateDefinition
- EssPartitionWriteDefFile

EssPartitionReadDefFile

Reads a partition definition file into memory.

Syntax

```
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

- EssPartitionCloseDefFile
- EssPartitionFreeDefCtx
- EssPartitionFreeOtlChanges
- EssPartitionGetAreaCellCount
- EssPartitionGetList
- EssPartitionGetOtlChanges
- EssPartitionGetReplCells
- EssPartitionNewDefFile
- EssPartitionOpenDefFile
- EssPartitionPurgeOtlChangeFile
- EssPartitionPutReplCells
- EssPartitionReadOtlChangeFile
- EssPartitionReplaceDefFile
- EssPartitionResetOtlChangeTime
- EssPartitionValidateDefinition
- EssPartitionValidateLocal
- EssPartitionWriteDefFile

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);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
pSelectMetaRecords	"ESS_PARTOTL_SELECT_CHG_T" on page 171	Criteria to select records to read.
pMetaChangeRead	ESS_PREAD_T	Pointer to meta change records read from the file.

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

```
ESS_FUNC_M ESS_PartitionReadOtlChangeFile(ESS_HCTX_T hCtx, ESS_HINST_T hInst)
{
   ESS_FUNC_M          sts;
   ESS_STR_T          chgfilename;
   ESS_TIME_T         time;
   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(&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;
   sts = EssPartitionReadOtlChangeFile (hCtx, &SelectMetaRecords, &MetaChangeRead);
   printf("EssPartitionReadOtlChangeFile   sts:  %ld\n",sts);
   sts = EssPartitionFreeOtlChanges(hCtx);
   printf("\tEssPartitionFreeOtlChanges   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

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

- EssPartitionReadOtlChangeFile
- EssPartitionResetOtlChangeTime
- EssPartitionValidateDefinition
- EssPartitionValidateLocal
- EssPartitionWriteDefFile

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);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
pSourcePartition	"ESS_PART_DEFINED_T" on page 153	Partition for the new time.
pDestinationPartition	"ESS_PART_DEFINED_T" on page 153	Partition where the time is reset.

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);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
pSelectVerify	"ESS_PARTSLCT_VALIDATE_T" on page 172	Description of the partition to verify.

Parameter	Data Type	Description
pulInvalidComponent	ESS_PULONG_T	Number of errors and warnings resulting from validation.
ppInvalidComponent	"ESS_PARTDEF_INVALID_T" on page 155	List of errors and warnings resulting from validation.
pRemoteDDBFileName	ESS_STR_T	Remote server partition definition file name.

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

```
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);
```

```
  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

```
ESS_FUNC_M EssPartitionValidateLocal (hCtx, pusValidateResult);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
pusValidateResult	ESS_PUSHORT_T	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

```
ESS_FUNC_M ESS_PartitionValidateLocal(ESS_HCTX_T hCtx)
{
   ESS_FUNC_M    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

- EssPartitionGetReplCells
- EssPartitionNewDefFile
- EssPartitionOpenDefFile
- EssPartitionPurgeOtlChangeFile
- EssPartitionPutReplCells
- EssPartitionReadDefFile
- EssPartitionReadOtlChangeFile
- EssPartitionReplaceDefFile
- EssPartitionResetOtlChangeTime
- EssPartitionValidateDefinition
- EssPartitionWriteDefFile

EssPartitionWriteDefFile

Writes the current memory version of the shared-partition definition file to disk.

Syntax

```
ESS_FUNC_M EssPartitionWriteDefFile (hCtx, iFileHandle, TpDdbCtx);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	Essbase API context handle.
iFileHandle	ESS_INT_T	Handle to shared partition definition file.
pDdbCtx	"ESS_PART_T" on page 152	Values to be written out.

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
- EssPartitionCloseDefFile
- EssPartitionFreeDefCtx

- EssPartitionFreeOtlChanges
- EssPartitionGetAreaCellCount
- EssPartitionGetList
- EssPartitionGetOtlChanges
- EssPartitionGetReplCells
- EssPartitionNewDefFile
- EssPartitionOpenDefFile
- EssPartitionPurgeOtlChangeFile
- EssPartitionPutReplCells
- EssPartitionReadDefFile
- EssPartitionReadOtlChangeFile
- EssPartitionReplaceDefFile
- EssPartitionResetOtlChangeTime
- EssPartitionValidateDefinition
- EssPartitionValidateLocal

EssPerformAllocationASO

Performs or verifies an allocation on an aggregate storage database.

Syntax

```
ESS_FUNC_M EssPerformAllocationASO (hCtx, verifyOnly, errorList, allocStruct);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
verifyOnly	ESS_BOOL_T	Flag to indicate validation of allocation parameters without performing allocation. If it is set to ESS_TRUE, the allocation parameters are validated only. If it is ESS_FALSE, the allocation is verified and executed.
errorList	"ESS_PERF_ALLOC_ERROR_T" on page 173**	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.
allocStruct	"ESS_PERF_ALLOC_T" on page 174*	Structure specifying the allocation parameters.

Return Value

Returns 0 if successful; otherwise, returns an error.

Example

```
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_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_DIVIDEAMT;
   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);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
verifyOnly	ESS_BOOL_T	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.
errorList	"ESS_PERF_ALLOC_ERROR_T" on page 173**	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.
calcStruct	"ESS_PERF_CUSTCALC_T" on page 177*	Pointer to a client-allocated custom calculation structure and parameters.

Return Value

Returns 0 if successful; otherwise, returns an error.

Example

```
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], [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

```
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

```
ESS_FUNC_M
ESS_PutObject (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

- DESCENDANTSOF
- PARENTOF
- ANCESTORSOF
- ALLSIBILINGSOF

- 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 |
                    <NEWLINESEPARATED |
                    <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:

```
<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:

```
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

```
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);
```

Parameter	Data Type	Description
hInstance	ESS_HINST_T	API instance handle.
Size	ESS_SIZE_T	New size of memory block to reallocate.
ppBlock	ESS_PPVOID_T	Address of pointer to previously allocated memory block, to be updated to point to reallocated memory block.

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

```
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

```
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

```
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

Removes all data block locks on a database which are currently held by a user.

Syntax

```
ESS_FUNC_M EssRemoveLocks (hCtx, AppName, DbName, LoginId);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
AppName	ESS_STR_T	Application name.
DbName	ESS_STR_T	Database name.
LoginId	ESS_LOGINID_T	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 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

```
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_T        AppName;
   ESS_STR_T        DbName;
   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

```
ESS_FUNC_M EssReplayTransactions(hCtx, AppName,
                                 DbName, ReplayDat, pSeqIds);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	Login context
AppName	ESS_STR_T	Application Name
DbName	ESS_STR_T	Database Name
ReplayDat	ESS_TRANSACTION_REPLAY_INP_T	Replay input parameters
pSeqIds	ESS_PSEQID_T	Array of sequence ID ranges if input type is sequence ID

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);
        /* 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_GIVENTIME;
    ReplayDat.value.InpTime = timestamp;
    sts = EssReplayTransactions (hCtx, AppName, DbName,
                                                        ReplayDat, pSeqIds);
    printf("EssReplayTransactions sts: %ld\r\n",sts);
    printf("\n\n");

    /* 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,
                                          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
```

```
seqid_array[0].seq_id_end_upper = 0
seqid_array[1].seq_id_start = 8
seqid_array[1].seq_id_end = 10
seqid_array[1].seq_id_start_upper = 0
seqid_array[1].seq_id_end_upper = 0
seqid_array[2].seq_id_start = 12
seqid_array[2].seq_id_end = 16
seqid_array[2].seq_id_start_upper = 0
seqid_array[2].seq_id_end_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

```
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

```
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

```
ESS_FUNC_M  EssRenameDatabase (hCtx, AppName, OldName, NewName);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
AppName	ESS_STR_T	Application name.
OldName	ESS_STR_T	Name of existing database to rename.
NewName	ESS_STR_T	New name of 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 (ESS_PRIV_DBCREATE).

Example

```
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);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle
AppName	ESS_STR_T	Application name
DbName	ESS_STR_T	Database name
OldName	ESS_STR_T	Old name of existing filter to be renamed
NewName	ESS_STR_T	New name of 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 (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

```
ESS_FUNC_M
ESS_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

```
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

```
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 = EssRenameObject(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

```
ESS_FUNC_M  EssReport (hCtx, Output, Lock, RptSpec);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
Output	ESS_BOOL_T	Controls output of data. If TRUE, data is output from the server, according to the specified report. If FALSE, no data is output.
Lock	ESS_BOOL_T	Controls block locking. If TRUE, all blocks which are accessed by the report specification are locked for update. If FALSE, no blocks are locked.
RptSpec	ESS_STR_T	The report specification, as a single string (must be less than 64 KB)..

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

```
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);
```

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 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 EssCreateLocalContext.
AppName	ESS_STR_T	Application name for report file location.
DbName	ESS_STR_T	Database name for report file location.
FileName	ESS_STR_T	Name of report specification file. It is not necessary to specify the file extension; the extension is understood to be .rep.
Output	ESS_BOOL_T	Controls output of data. If TRUE, data is output from the server, according to the specified report. If FALSE, no data is output.
Lock	ESS_BOOL_T	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 **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 re-registration will be skipped with a warning.

Example

```
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

```
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

Parameter	Data Type	Description
hCtx;	ESS_HCTX_T	API context handle
persistence;	ESS_ULONG_T	One of the following values indicating the persistence of the set of tables to be reset: • 0: Reset short term tables only • 1: Reset short and medium term tables • 2: Reset short, medium and long term tables • 3: Disable performance statistics gathering • 4: Enable performance statistics gathering
scope;	ESS_ULONG_T	One of the following values indicating the scope of the set of tables to be reset: • 1: Reset thread-based tables only • 2: Reset database-based tables only • 4: Reset server-based tables only • 7: Reset all tables

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;
}
```

See Also

- EssDumpPerfStats
- EssGetStatBufSize

EssResetUser

Resets the user's security structure to its initial state.

Syntax

```
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

```
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

- EssInit

- EssLogin
- EssLogout

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);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	Login context
AppName	ESS_STR_T	Application Name **Note:** Works only at the database level. The AppName parameter specifies an Application in order to access the database residing within.
DbName	ESS_STR_T	Database Name
BackupFileName	ESS_STR_T	Full path to the backup file from which to read archive data. Specify the full path, for example: `c:\hyperion\Test.arc`

Parameter	Data Type	Description
bForceDiffName	ESS_BOOL_T	Use a different application and database names for the restore. ● ESS_TRUE—force a different application and/or database name for the restore. If you use ESS_TRUE and the application and database name is same as the backup, the result is the same as ESS_FALSE ● 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.
Count	ESS_USHORT_T	Optional Number of disk volume replacement structures that are being restored.
ReplaceVol	ESS_PDISKVOLUME_REPLACE_T	Optional Disk volume replacement input structures.

Return Value

Returns:

- 0—If successful
- Error number—If unsuccessful

Access

The caller must have Essbase Administrator access to the database.

Example

```
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\r\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\r\n",sts);

    sts = EssRestoreDatabase (hCtx, AppName, DbName,
                                                BackupFileName, bForceDiffName,
                                                count, replaceVol);

    printf("EssRestoreDatabase sts: %ld\r\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

```
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

- EssBeginUpdate
- EssGetString

EssSetActive

Sets the caller's active application and database.

Syntax

```
ESS_FUNC_M  EssSetActive (hCtx, AppName, DbName, pAccess);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
AppName	ESS_STR_T	Application name.
DbName	ESS_STR_T	Database name.
pAccess	ESS_PACCESS_T	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.

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

```
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

```
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

```
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

```
ESS_FUNC_M  EssSetApplicationAccess (hCtx, Count, pUserApp);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
Count	ESS_USHORT_T	Count of user application structures.
pUserApp	"ESS_USERAPP_T, ESS_GROUPAPP_T" on page 189	Pointer to an array of user application structures.

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

```
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

```
ESS_FUNC_M EssSetApplicationAccessEx (hCtx, Count, pUserApp);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle (input).
Count	ESS_USHORT_T	Count of user application structures (input).
pUserApp	ESS_PUSERAPPEX_T	Pointer to an array of user application structures (input).

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

```
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);
      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_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;
         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_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);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
AppName	ESS_STR_T	Application name.
pAppState	"ESS_APPSTATE_T" on page 117	Pointer to application state structure.

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 *lroSizeLimit*.

Return Value

None.

Access

This function requires the caller to have application designer privilege (ESS_PRIV_APPDESIGN) for the specified application.

Example

```
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

```
ESS_FUNC_M  EssSetCalcList (hCtx, UserName, AppName, DbName, AllCalcs, Count,
pCalcList);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle
UserName	ESS_STR_T	User name
AppName	ESS_STR_T	Application name
DbName	ESS_STR_T	Database name. If NULL, uses Application subdirectory

Parameter	Data Type	Description
AllCalcs	ESS_BOOL_T	Allow all calcs flag. If TRUE, the user can access all calc scripts, otherwise, they can only access those specified in the CalcList argument.
Count	ESS_USHORT_T	Count of the number of accessible calc script objects
pCalcList	ESS_POBJNAME_T	Pointer to an array of calc script object names

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

```
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_OBJNAME_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);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle (input).
UserID	ESS_STR_T	Input. User or group for which to set the calculation list. Can be specified as `name@provider` or as a unique identity attribute.
bIsIdentity	ESS_BOOL_T	Input. Indicates whether *UserID* is a name or an identity. If TRUE, *UserID* is an identity.
entityType	ESS_USHORT_T	Input. Type of entity contained in *UserID*. Can be one of the following: • ESS_TYPE_USER • ESS_TYPE_GROUP
AppName	ESS_STR_T	Application name (input).
DbName	ESS_STR_T	Database name (input).
AllCalc	ESS_BOOL_T	Allow all calcs flag (input). If TRUE, the user or group can access all calculation scripts; otherwise, only those specified in *CalcList* are accessible.
Count	ESS_USHORT_T	Count of the number of accessible calculation script objects (input).
pCalcList	ESS_POBJNAME_T	Pointer to an array of accessible calculation script object names (input).

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

```
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("\tThis user has access to all script on %s %s\n", AppName, DbName);
   else
   {
      if(!sts && pCalcList)
      {
         printf("-------- Get Calc List -----------\r\n");
            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_PUSERINFO_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);
}
```

See Also

- EssGetCalcList
- EssListObjects
- EssListUsersInfoEx

EssSetDatabaseAccess

Sets a list of user database access structures, which contain information about user access to databases.

Syntax

```
ESS_FUNC_M EssSetDatabaseAccess (hCtx, Count, pUserDb);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
Count	ESS_USHORT_T	Count of user database structures.
pUserDb	"ESS_USERDB_T, ESS_GROUPDB_T" on page 190	Pointer to an array of user database structures.

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

```
ESS_FUNC_M
ESS_SetDatabaseAccess (ESS_HCTX_T  hCtx)
{
   ESS_FUNC_M      sts = 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

```
ESS_FUNC_M EssSetDatabaseAccessEx (hCtx, Count, pUserDb);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle (input).
Count	ESS_USHORT_T	Count of user database structures (input).
pUserDb	ESS_PUSERDBEX_T	Pointer to an array of user database structures (input).

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

```
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;
   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;
         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", 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

```
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

```
ESS_FUNC_M
ESS_SetDatabaseNote (ESS_HCTX_T hCtx)
{
   ESS_FUNC_M      sts = ESS_STS_NOERR;
   ESS_STR_T      AppName;
   ESS_STR_T      DbName;
   ESS_STR_T      DbNote;

   AppName = "Sample";
   DbName  = "Basic";
   DbNote  = "This is a test";

   sts = EssSetDatabaseNote(hCtx, AppName, DbName,
         DbNote);

   return (sts);
}
```

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

```
ESS_FUNC_M  EssSetDefaultCalcFile (hDestCtx, hSrcCtx, AppName, DbName, FileName);
```

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 default calc script file

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

```
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

- EssGetDefaultCalc
- EssSetDefaultCalc

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");

   sts = EssSetEasLocation(hCtx, easLoc);

   if (sts)
      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

```
ESS_FUNC_M EssSetExtUser (hCtx, type, UserName, Password, Protocol, ConnParam);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
type	ESS_USHORT_T	Type of user (external).
UserName	ESS_STR_T	Name of user.
Password	ESS_STR_T	User's password.
Protocol	ESS_STR_T	External authentication protocol: CSS, for Shared Services mode.
ConnParam	ESS_STR_T	External authentication connection parameters. Null, for CSS protocol.

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

```
ESS_FUNC_M  EssSetFilter (hCtx, AppName, DbName, FilterName, Active, Access);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle
AppName	ESS_STR_T	Application name
DbName	ESS_STR_T	Database name
FilterName	ESS_STR_T	Filter name. See "Filter Name Limits" on page 1728.

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

```
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;
      AccessAry[2] = ESS_ACCESS_WRITE;
      /***** 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);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle
AppName	ESS_STR_T	Application name
DbName	ESS_STR_T	Database name
FilterName	ESS_STR_T	Filter name
Count	ESS_USHORT_T	Count of groups or users assigned this filter
pUserList	ESS_PUSERNAME_T	Pointer to array of user names

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_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

```
ESS_FUNC_M EssSetFilterListEx (hCtx, AppName, DbName, FilterName, bIsIdentity,
entityType, Count, UserList);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle (input).
AppName	ESS_STR_T	Application name (input).
DbName	ESS_STR_T	Database name (input).
FilterName	ESS_STR_T	Filter name (input).

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

```
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);
      }
      else
         printf("none.");
      printf("\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);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
pGlobal	"ESS_GLOBAL_T" on page 140	Pointer to global state structure.

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

```
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

```
ESS_FUNC_M EssSetGroup (hCtx, pGroupInfo);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
pGroupInfo	"ESS_USERINFO_T, ESS_GROUPINFO_T" on page 192	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 *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

```
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);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
GroupName	ESS_STR_T	Group name or user name.
Count	ESS_USHORT_T	Count of user names.
pUserList	ESS_PUSERNAME_T	Pointer to an array of user name strings.

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
ESS_SetGroupList (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);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle (input).
EntityId	ESS_STR_T	User or group name (input). Can be specified as `name@provider` or as a unique identity attribute.
bIsIdentity	ESS_BOOL_T	Input. Indicates if *EntityId* is a name or an identity. If TRUE, *EntityId* is an identity.
entityType	ESS_USHORT_T	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.
Count	ESS_USHORT_T	Count of identities (input).
pIdList	ESS_PSTR_T	Pointer to an array of identities (input).
bUsingIdentity	ESS_BOOL_T	Input. Indicates if *EntityId* is a name or an identity. If TRUE, *EntityId* is an identity.

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

```
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

```
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);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
UserName	ESS_STR_T	User name.
Password	ESS_STR_T	New password for user.

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

```
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

```
ESS_FUNC_M EssSetPath (pszPath);
```

Parameter	Data Type	Description
pszPath;	ESS_STR_T	Pointer to the string describing the ESSBASEPATH environment variable

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

```
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

```
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

```
ESS_FUNC_M EssSetSpanRelationalPartition (hCtx);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.

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

```
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 = EssSetSpanRelationalPartition (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

- 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);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
option	ESS_USHORT_T	Integer representing the desired password creation method for migrated users. • 0—Use an administrator-specified password. • 1—Create passwords that are the same as user names for users being migrated to Shared Services. **Note:** The passwords are created as lower-case, even if there are upper-case letters in the user name. • 2—Automatically generate new passwords for the users being migrated to Shared Services.
NewPassword	ESS_STR_T	Password string (if *option* 2 is used).
FileName	ESS_STR_T	Name of file to contain saved passwords. If not provided, the default file is `$ARBORPATH/bin/MigratedUsersPassword.txt`.
flag	ESS_USHORT_T	Whether the passwords file, if already existing, should be overwritten. • ESS_FILE_OVERWRITE—Overwrite • ESS_FILE_NOOVERWRITE—Do not overwrite; return an error.

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_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

```
ESS_FUNC_M EssSetUserToSS (hCtx, UserName, option, NewPassword, FileName, flag);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
UserName	ESS_STR_T	Name of the user to convert to Shared Services (input).

Parameter	Data Type	Description
option	ESS_USHORT_T	Integer representing the desired password creation method for migrated users. ● 0—Use an administrator-specified password. ● 1—Create passwords that are the same as user names for users being migrated to Shared Services. **Note:** The passwords are created as lower-case, even if there are upper-case letters in the user name. ● 2—Automatically generate new passwords for the users being migrated to Shared Services.
NewPassword	ESS_STR_T	Password string (if *option* 2 is used).
FileName	ESS_STR_T	Name of file to contain saved passwords. If not provided, the default file is `$ARBORPATH/bin/MigratedUsersPassword.txt`.
flag	ESS_USHORT_T	Whether the passwords file, if already existing, should be overwritten. ● ESS_FILE_OVERWRITE—Overwrite ● ESS_FILE_NOOVERWRITE—Do not overwrite; return an error.

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_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

```
ESS_FUNC_M EssSetUser (hCtx, pUserInfo);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
pUserInfo	"ESS_USERINFO_T, ESS_GROUPINFO_T" on page 192	Pointer to user info structure.

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 = 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

ESS_FUNC_M **EssSetUser** (*hCtx, pUserInfo*);

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
pUserInfoEx	"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 theESS_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);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.

Parameter	Data Type	Description
option	ESS_USHORT_T	Integer representing the desired password creation method for migrated users. ● 0—Use an administrator-specified password. ● 1—Create passwords that are the same as user names for users being migrated to Shared Services. **Note:** The passwords are created as lower-case, even if there are upper-case letters in the user name. ● 2—Automatically generate new passwords for the users being migrated to Oracle's Hyperion® Shared Services.
NewPassword	ESS_STR_T	Password string (if *option* 2 is used).
FileName	ESS_STR_T	Name of file to contain saved passwords. If null, the default file is `$ARBORPATH/bin/MigratedUsersPassword.txt`.
flag	ESS_USHORT_T	Whether the passwords file, if already existing, should be overwritten. ● ESS_FILE_OVERWRITE—Overwrite ● ESS_FILE_NOOVERWRITE—Do not overwrite; return an error.

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_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 inOracle 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)
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	Essbase API context handle.
UserName	ESS_STR_T	Name of user.
UserType	ESS_USER_TYPE_T	ESS_USER_ESSBASE is the application access type, if no application access type is specified. This user will be enabled with all functionality.
Cmd	ESS_USERTYPE_CMD_T	Whether to add/remove/replace the type specified. Only the Essbase type, ESS_USER_ESSBASE, can be added or removed. • 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)
{
```

```
    ESS_FUNC_M                  sts = ESS_STS_NOERR;
    ESS_STR_T                   UserName="jsmith";
    ESS_USER_TYPE_T             UserType = ESS_USER_ESSBASE;
    ESS_USERTYPE_CMD_T          cmd = ESS_USERTYPE_CMD_ADD;
    sts = EssSetUserType        (hCtx, UserName, UserType, Cmd);
    return (sts);
}
```

See Also

- EssGetUserType

EssShutdownServer

Stops the 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

```
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

```
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

```
/* 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

```
ESS_FUNC_M  EssUnloadApplication (hCtx, AppName);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle
AppName	ESS_STR_T	Name of application to load

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

```
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

```
ESS_FUNC_M  EssUnloadDatabase (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 unload.

Return Value

None.

Access

This function requires the caller to have database load/unload privilege (ESS_PRIV_APPLOAD).

Example

```
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

- EssLoadDatabase
- EssUnloadDatabase

EssUnlockObject

Unlocks a locked object on the server or client object system.

Syntax

```
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

```
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

- EssListObjects
- EssLockObject
- EssPutObject

EssUpdate

Sends an update specification to the active database as a single string.

Syntax

```
ESS_FUNC_M  EssUpdate (hCtx, Store, Unlock, UpdtSpec);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
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	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.
UpdtSpec	ESS_STR_T	The update specification, as a single string (must be less than 32 KB).

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

```
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

```
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

```
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 bMerge = 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

```
ESS_FUNC_M  EssUpdateEx (hCtx, Store, Unlock, UpdtSpec, ppMbrError);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
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	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.
UpdtSpec	ESS_STR_T	The update specification, as a single string (must be less than 32 KB).
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.
- 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

```
void TestUpdateEx()
{
    ESS_STS_T sts = ESS_STS_NOERR;
    ESS_PMBRERR_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

```
ESS_FUNC_M  EssUpdateFile (hDestCtx, hSrcCtx, AppName, DbName, FileName, Store, Unlock);
```

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 report file location. The report 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	databasename 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	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.

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

```
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);
}
```

See Also

- EssBeginUpdate
- EssReportFile
- EssUpdate

EssUpdateFileASO

Sends an update specification to the active aggregate storage database from a file.

Syntax

```
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

```
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

```
ESS_FUNC_M EssUpdateFileASOEx (hDestCtx, hSrcCtx, AppName, DbName, FileName, Store,
Unlock, ulBufferId, ppMbrError);
```

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.
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

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

```
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("\tError 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);
```

```
    if(!sts)
    {
        VerifyDataload("'Mar' 'Sale' 'Curr Year' 'Original Price' '017589' '13668'
'Cash' 'No Promotion' '1 to 13 Years' 'Under 20,000' 'Digital Cameras' 10\n 'Camcorders'
20\n '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

```
ESS_FUNC_M EssUpdateFileEx (hDestCtx, hSrcCtx, AppName, DbName, FileName, Store, Unlock,
ppMbrError);
```

Parameter	Data Type	Description
hDestCtx	ESS_HCTX_T	API context handle of the target databae 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	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.

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

```
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

```
ESS_FUNC_M EssUpdateFileUTF8ASO (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 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
- 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

```
ESS_FUNC_M EssUpdateFileUTF8ASOEx (hDestCtx, hSrcCtx, AppName, DbName,
FileName, Store, Unlock, ulBufferId, ppMbrError);
```

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.

Parameter	Data Type	Description
ulBufferId	ESS_ULONG_T	ID number for the data load buffer.
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

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

```
ESS_FUNC_M EssUpdateUtf8Ex (hDestCtx, hSrcCtx, AppName, DbName, FileName, Store, Unlock,
ppMbrError);
```

Parameter	Data Type	Description
hDestCtx	ESS_HCTX_T	API context handle of the target databae 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	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.
ppMbrError	ESS_PPMBRERR_T	Pointer to linked list of errors contained in ESS_MBRERR_T. Possible errors are: • 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_ERRORLOAD—Unable to do dataload at Item/Record [*number*]. • ESS_MBRERR_UNKNOWN—

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`

- EssUpdateFileUTF8ASO
- EssUpdateUtf8Ex

EssUpdateUtf8Ex

Sends an update specification to the active database as a single UTF-8-encoded string.

Syntax

```
ESS_FUNC_M  EssUpdateUtf8Ex (hCtx, Store, Unlock, UpdtSpec, ppMbrError);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
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	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.
UpdtSpec	ESS_STR_T	The update specification, as a single string (must be less than 32 KB).
ppMbrError	ESS_PPMBRERR_T	Pointer to linked list of errors contained in ESS_MBRERR_T. Possible errors are: • 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_ERRORLOAD—Unable to do dataload at Item/Record [*number*]. • ESS_MBRERR_UNKNOWN—

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

```
ESS_FUNC_M EssValidateDB (hCtx, DbName, FileName);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
DbName	ESS_STR_T	Database name. Required, cannot be NULL.
FileName	ESS_STR_T	Error log file name, to be placed in the app\db directory on the server. Required.

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

```
ESS_VOID_T
ESS_ValidateDB (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

```
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

```
#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);
   }
   fprintf(stdout, "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, 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();
```

```
            ESS_Term();
}
```

See Also

- EssLogin
- EssAutoLogin
- EssTerm

EssVerifyFilter

Verifies the syntax of a series of filter row strings against a specified database.

Syntax

```
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

```
ESS_VOID_T
ESS_VerifyFilter (ESS_HCTX_T hCtx)
{
   ESS_FUNC_M        sts = 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);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
FormulaName	ESS_STR_T	The name of the formula to verify.

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);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
ruleFileName	ESS_STR_T	The name of the rules file to verify.
pNmColumns	ESS_PULONG_T	Pointer to the number of columns in the rules file.
ppColumnErrors	ESS_ULONG_T	Pointer to the array of errors found.

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.

Error code	Meaning
DAT_VERIFY_INVALIDMBR	There is an unknown member (or no member) in the field name.
DAT_VERIFY_INVALIDHDR	There is an unknown member in the header. (This error code applies to the header, and will
DAT_VERIFY_SAMENAME	This field has the same field name as another field.
DAT_VERIFY_DIMUSED	The dimension name is used in another field name or in the header.
DAT_VERIFY_MBRUSED	A member name used as part of a combination in this field is used as a single member name in
DAT_VERIFY_DIMINCROSSDIM	A dimension name is used in a cross-dimensional reference in the field name.
DAT_VERIFY_DATAFIELD	Only one field can have the Data Field attribute.
DAT_VERIFY_SIGNFLIPDIM	The dimension used for Sign Flip checking is not in the associated outline.
DAT_VERIFY_DUPINHEADER	This field name is also defined in the header definition.
DAT_VERIFY_DATEANDDATA	A field may be designated a Data Field or Date Field, but not both.
DAT_VERIFY_DATEFIELDNAME	The field name of a date field must be the name of a date dimension.
DAT_VERIFY_DATEFORMAT	There is an unrecognized date format for this date column.

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

```
{
   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
```

```
            {
               if( pColumnErrors[i] & DAT_VERIFY_INVALIDMBR )
                  printf("     There is an unknown member (or no member) in the field
name.\n");
               if( pColumnErrors[i] & DAT_VERIFY_INVALIDHDR )
                  printf("     There is an unknown member in the header.\n");
               if( pColumnErrors[i] & DAT_VERIFY_SAMENAME )
                  printf("     This field has the same field name as another field.\n");
               if( pColumnErrors[i] & DAT_VERIFY_DIMUSED )
                  printf("     The dimension name is used in another field name or in
the header.\n");
               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.\n");
               if( pColumnErrors[i] & DAT_VERIFY_DIMINCROSSDIM )
                 printf("     A dimension name is used in a cross-dimensional reference
in the field name.\n");
               if( pColumnErrors[i] & DAT_VERIFY_DATAFIELD )
                  printf("     Only one field can have the Data Field attribute.\n");
               if( pColumnErrors[i] & DAT_VERIFY_SIGNFLIPDIM )
                  printf("     The dimension used for Sign Flip checking is not in the
associated outline.\n");
               if( pColumnErrors[i] & DAT_VERIFY_DUPINHEADER )
                  printf("     This field name is also defined in the header definition.
\n");
               if( pColumnErrors[i] & DAT_VERIFY_DATEANDDATA )
                  printf("     A field may be designated a Data Field or a Date Field,
but not both.\n");
               if( pColumnErrors[i] & DAT_VERIFY_DATEFIELDNAME )
                 printf("     The field name of a date field must be the name of a date
dimension.\n");
               if( pColumnErrors[i] & DAT_VERIFY_DATEFORMAT )
                 printf("     There is an unrecognized date format for this date column.
\n");
            }
         }
         EssFree(hInst, pColumnErrors);
      }
   }
}
```

See Also

- EssVerifyFormula
- EssOtlVerifyFormula
- EssOtlVerifyOutlineEx

EssWriteToLogFile

Writes a message to the Essbase Server log file (`essbase.log`), or to the application log file (*appname*`.log`).

Syntax

```
ESS_FUNC_M  EssWriteToLogFile (hCtx, AgentLog, Message);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context handle.
AgentLog	ESS_BOOL_T	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`.
Message	ESS_STR_T	Message to be logged to the Essbase Server log file (`essbase.log`), or to the application log file (*`appname`*`.log`).

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

```
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

Part III

C Outline API

In C Outline API:

- Using the C Outline API
- C Outline API Declarations
- C Outline API Functions
- C Outline API Examples

9 Using the C Outline API

In This Chapter

C Outline API Overview 679
C Outline API Error Handling 679
C Outline API Server Outline Queries 680
C Outline API Outline Verification 680
C Outline API Memory Allocation 681
C Outline API Security Requirements 681
C Outline API Function Call Sequence 682
Typical C Outline API Task Sequence 682

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 relevent 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 **EssOtlFreeMembers**() 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**().

10 C Outline API Declarations

In This Chapter

C Outline API Error Return Values 685
C Outline API DTS Member Structures 691
C Outline API Symbolic Constant Definitions 692
ESS_ATTRIBUTEQUERY_T 697
ESS_GENLEVELNAME_T 698
ESS_GENLEVELNAMEEX_T 699
ESS_MBRCOUNTS_T 699
ESS_MBRINFO_T 699
ESS_OTLQUERYERRORLIST_T 703
ESS_OUTERROR_T 704
ESS_OUTLINEINFO_T 706
ESS_OUTLINEINFOEX_T 707
ESS_PERSPECTIVE_T 708
ESS_PREDICATE_T 709
ESS_SVROTLINFO_T 710
ESS_VALIDITYSET_T 710

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

Value	Description
OTLAPI_BAD_ALIASTABLE	Illegal alias table
OTLAPI_BAD_CONSOL	Invalid consolidation type (+,-,etc.)
OTLAPI_BAD_GENLEVELNAME	Invalid generation or level name
OTLAPI_BAD_HOUTLINE	Invalid outline handle passed to EssOtl... function
OTLAPI_BAD_MBRNAME	Invalid member name

Value	Description
OTLAPI_BAD_MEMBER	Invalid member handle
OTLAPI_BAD_MOVE	Illegal move of member. Can't move member to its descendant.
OTLAPI_BAD_OBJTYPE	Illegal object type
OTLAPI_BAD_OUTLINETYPE	Invalid outline type
OTLAPI_BAD_PERSPECTIVE2	Invalid perspective
OTLAPI_BAD_RENAMESHARE	A shared member cannot be renamed
OTLAPI_BAD_RESTRUCTTYPE	Invalid restructure type
OTLAPI_BAD_SCA_VALIDITYSET_TYPE	Perspectives/validity sets do not support this validity set type
OTLAPI_BAD_SMARTLISTNAME	Invalid text list name
OTLAPI_BAD_SORT COMPAREFUNC	Invalid sorting compare function
OTLAPI_BAD_SORTTYPE	Invalid sort type
OTLAPI_BAD_TRANSTYPE	Unknown transaction type when creating a transaction (internal error)
OTLAPI_BAD_USERATTR	Invalid user attribute
OTLAPI_CUR_NOACCOUNTS	There is no Accounts dimension. You need an Accounts dimension to create a currency database.
OTLAPI_CUR_NOCOUNTRY	There is no Country dimension. You need a Country dimension to create a currency database.
OTLAPI_CUR_NOTIME	There is no Time dimension. You need a Time dimension to create a currency database.
OTLAPI_ERR_ADDDELETEDIMDYNAMICCALC	Member in which to store data is type Dynamic Calc
OTLAPI_ERR_ADDNAMEUSED	Member name already used (add operation)
OTLAPI_ERR_ALIASTABLEEXISTS	Alias table already exists
OTLAPI_ERR_ALIASLANGUAGE_UNAVAILABLE	Alias table languages are unavailable for outline versions before 11.1.2.0.00.
OTLAPI_ERR_ALIASTABLENAME	Illegal alias table name
OTLAPI_ERR_ALREADY CURRENCY	The outline is a currency outline. You are trying to create a currency outline, and the initial outline is already a currency outline.
OUTAPI_ERR_ASO_COMPRESSIONMUSTBEDYNAMIC	Aggregate storage outlines require compression dimension to be a single dynamic hierarchy
OUTAPI_ERR_ASO_DIFFERENTNUMBEROFSHARES	This prototype should have same number of shared members as its previous sibling

Value	Description
OUTAPI_ERR_ASO_SHAREDMEMBERSNOTINSAMEORDER	This prototype must have each of its shared members as next sibling to its previous sibling's shared members
OTLAPI_ERR_ATTR_ATTACHED_WRONGLEVEL	This attribute is attached at wrong level at least once
OTLAPI_ERR_ATTRMBR_ALREADYASSOCIATED	Base member already associated with an attribute member from the same dimension
OTLAPI_ERR_BADDIM	Invalid dimension argument
OTLAPI_ERR_BADHIER	Invalid hierarchy type
OTLAPI_ERR_BADHIER_TOP	Invalid hierarchy member designation - Member must be at generation 1 or 2
OTLAPI_ERR_BADSHARE	Illegal share value
OTLAPI_ERR_BADSKIP	Illegal time balance skip value
OTLAPI_ERR_BADSTORAGE	Illegal dimension storage value
OTLAPI_ERR_BADSTORAGECATEGORY	Illegal storage category
OTLAPI_ERR_BADTIMEBAL	Illegal time balance value
OTLAPI_ERR_BSO_SOLVEORDER	Block storage outlines that have not been enabled for member types cannot have a solve order
OTLAPI_ERR_CANTIDENTIFYMBR_DUPLICATEDNAME	Cannot uniquely identify a member because the name is duplicated
OTLAPI_ERR_CNTS_INDEP_LAST	Independent dimension list must be ordered with continuous independent dimensions last
OTLAPI_ERR_CONFIGTOOMANYDIMS	Too many dimensions to configure automatically
OTLAPI_ERR_COPYALIASTABLE	Source and destination tables are the same
OTLAPI_ERR_CREATETEMP	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.
OTLAPI_ERR_CURTOOMANYDIMS	Too many dimensions in a currency outline. A currency outline is limited to four dimensions.
OTLAPI_ERR_DELETEDEFALIAS	Cannot delete the default alias table
OTLAPI_ERR_DISCRETE_DIFFERENT	An independent range must have the same discrete start and end members
OTLAPI_ERR_DISCRETE_OR_CNTS	Independent dimension types must be either discrete or continuous
OTLAPI_ERR_DUP_LANGCODE	The language code is assigned to another alias table within the same database
OTLAPI_ERR_DUPLICATEALIAS	Duplicate alias
OTLAPI_ERR_DUPLICATENAME	Duplicate member name

Value	Description
OTLAPI_ERR_DUPGENLEVNAME	Cannot add, rename, or set a member name or alias that duplicates a generation or level name
OTLAPI_ERR_EXPORT_INCORRECT_FLAGS	There are invalid export flags. Export cannot be enabled to limit extraction to the tree and alias table
OTLAPI_ERR_EXPORT_INVALID_ALIASTABLE	An invalid alias table is specified in the export options
OTLAPI_ERR_EXPORT_INVALID_DIMLIST	The number of dimensions or the dimension list specified in the export options is invalid
OTLAPI_ERR_EXPORT_INVALIDDIM_DIMLIST	The dimension name specified in the export options dimension list is invalid
OTLAPI_ERR_EXPORT_INVALID_VERSION	This export version is invalid. Enter a valid export version
OTLAPI_ERR_EXPORT_UNABLE_FILE	Cannot open the file to export the outline
OTLAPI_ERR_EXPORT_UNABLE_PROCESS	Cannot process the outline because of unsupported outline type
OTLAPI_ERR_FAILED_GET_ALIASNAMES	Failed to get all alias names due to failed alias identifier lookup
OTLAPI_ERR_FEATURE_UNAVAILABLE	The feature is unavailable in this outline version; please migrate outline first
OTLAPI_ERR_FILEIO	Could not read from or write to file
OTLAPI_ERR_FILEOPEN	Could not open file
OTLAPI_ERR_FORMATSTRING_MISMATCH	Implied share or label-only member has a different format string than the original member; original member's format string will be applied
OTLAPI_ERR_FORMATSTRING_NOT_MEMBTYPE_ENABLED	The use of format strings require the outline to be member-type enabled. This outline is not member type enabled
OTLAPI_ERR_FORMATSTRINGTOOLONG	Format String too long for single locale configuration
OTLAPI_ERR_FUNCTION_OBSOLETE	Function is obsolete
OTLAPI_ERR_GENLEVELEXISTS	Generation or level already has a name
OTLAPI_ERR_GENLEVELNAMEEXISTS	Generation or level name already exists
OTLAPI_ERR_GENLEVELVALUE	Illegal generation or level value
OTLAPI_ERR_GENLEVNAMEMBR	Cannot add a generation or level name that duplicates a member name or alias
OTLAPI_ERR_ILLEGALALIAS STRING	Illegal member combinational for alias
OTLAPI_ERR_ILLEGALCOMBOALIAS	Illegal combinational alias name
OTLAPI_ERR_ILLEGALCURRENCY	Illegal currency member
OTLAPI_ERR_ILLEGALDEFALIAS	Illegal default alias name
OTLAPI_ERR_ILLEGALNAME	Illegal member name

Value	Description
OTLAPI_ERR_ILLEGALTAG	Illegal dimension tag (category)
OTLAPI_ERR_ILLEGALOPTION	Occurs when the user passes in an invalid option to **EssOtlGetGenNames()** or **EssOtlGetLevelNames()**
OTLAPI_ERR_IMPLIED_SHARE_OLD_VERSION	The outline version is too old to set Implied Share
OTLAPI_ERR_INCORRECT_MEMBERTYPE	Member type can be only set to numeric or date types
OTLAPI_ERR_INVALID_SMARTLIST_HANDLE	Invalid text list handle
OTLAPI_ERR_INVALID_SMARTLIST_IMPORTFILE	Input file for importing text lists is invalid
OTLAPI_ERR_INVALIDID_SMARTLIST_IMPORTFILE	Invalid or duplicate ID in text list import file
OTLAPI_ERR_LANGCODE_TOOLONG	Alias table language code exceeds the maximum length
OTLAPI_ERR_LEAFLABEL	Leaf member defined as a label member
OTLAPI_ERR_MAXALIASTABLES	Maximum number of alias tables has been reached
OTLAPI_ERR_MEMBERCALC	Illegal member formula
OTLAPI_ERR_MEMBERTYPE_OFF	Cannot turn off the member type enabled setting of an outline
OTLAPI_ERR_MBRCOMMENTEXLEN	Extended member comment is too long
OTLAPI_ERR_MISSINGTEXT_SMARTLIST_IMPORTFILE	Missing text for ID in text list import file
OTLAPI_ERR_MULT_DATE_DIMS	An outline can have at most one dimension with date types on static members
OTLAPI_ERR_MULTIHIER_NOT_ENABLED	Cannot set hierarchy type; multiple hierarchies not enabled for dimension
OTLAPI_ERR_MULT_SMARTLIST_DIMS	An outline can have at most one dimension with smartlists on static members
OTLAPI_ERR_MUSTSAVE_BEFORE_EDIT	The outline must be saved and re-opened before it can be edited
OTLAPI_ERR_NOALIAS	No alias for this member
OTLAPI_ERR_NOALIASCODE	Get/Set alias table language code is not yet implemented
OTLAPI_ERR_NOALIASCOMBO	No alias combination
OTLAPI_ERR_NOATTRONCOMPRESSEDDIM	Attributes are not allowed on the compressed dimension
OTLAPI_ERR_NOFORMULA	No formula for this member
OUTAPI_ERR_NOMEMBTYPE	This outline version does not support typed members
OTLAPI_ERR_NOSHAREPROTO	Shared member with no actual member
OUTAPI_ERR_NOSMARTLISTS	This outline version does not support text lists
OTLAPI_ERR_NOTADIM	Dimension name expected
OTLAPI_ERR_NOT_A_TIME_MBR	Invalid argument passed. Not a date-time dimension member

Value	Description
OTLAPI_ERR_NOT_LINKEDATTRIBUTEDIM	Not a linked attribute dimension handle
OTLAPI_ERR_NOT_MEMBTYPE_ENABLED	This outline is not member type enabled
OTLAPI_ERR_NOTIMEDIM	No time dimension defined (can't do time balance operations without a time dimension)
OTLAPI_ERR_NOTVERIFIED	Outline has errors (when saving to the server)
OTLAPI_ERR_OBJ_NOTFOUND	Object not found
OTLAPI_ERR_OBJTYPE_NOTSUPPORTED	Function not supported in server side edit mode
OTLAPI_ERR_OPENMODE	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.
OTLAPI_ERR_OTLDATEFORMAT	Invalid outline property: date format
OTLAPI_ERR_OTLSHARED_FORMAT	Outline member's format string cannot be set for shared members
OTLAPI_ERR_OTLSHARED_TYPE	Outline member's type cannot be set for shared members
OTLAPI_ERR_QUERYHINT_INVALIDARRAYSIZE	Invalid query hint array size
OTLAPI_ERR_RENAMEDEFALIAS	Cannot rename the default alias table
OTLAPI_ERR_RENAMENAMEUSED	Member name already used (rename operation)
OTLAPI_ERR_SCA_NOT_ENABLED	This outline is not enabled for varying attributes
OTLAPI_ERR_SCA_UNAVAILABLE	Varying attributes feature is unavailable in this version
OTLAPI_ERR_SHAREDMEMBERFORMULA	Shared member cannot have a formula
OTLAPI_ERR_SHARENOTLEVEL0	Shared member not at level 0 (a shared member cannot be a parent of another member)
OTLAPI_ERR_SHAREUDA	Cannot set a user attribute for a shared member
OTLAPI_ERR_SMARTLISTNAMEUSED	Cannot add text list; text list name already used
OTLAPI_ERR_SMARTLIST_MAPMAXREACHED	Cannot add more than *n* text list texts to id mappings
OTLAPI_ERR_SMARTLISTMAXREACHED	Cannot add text list; *n* maximum text lists supported
OTLAPI_ERR_SMARTLIST_MISSING	Missing text list association for text-typed member
OTLAPI_ERR_TBTAGS_WITH_DYN_HIERARCHY	This member has a TB-Tag. That requires TIME dimension to only have STORED hierarchies
OTLAPI_ERR_TIMESPARSE	Accounts dimension is dense and time dimension sparse-is not used
OTLAPI_ERR_TYPED_ATTR_LEVEL0	Attribute members and non level-0 aggregate storage members cannot be set to Date or Text type

Value	Description
OTLAPI_ERR_TYPED_DIMS	Text typed members, date typed members, and stored members with format strings should be specified along the same dimension
OTLAPI_ERR_UNKNOWNDTSMBR	Unknown DTS member
OTLAPI_ERR_VALIDITYSET_MATCH	The validity set must match an existing set in the outline
OTLAPI_ERR_VIRTLEVONOFORMULA	Dynamic Calc members must have formulas or children, or else they cannot be calculated
OTLAPI_ERR_VIRTBADPARENT	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
OTLAPI_ERR_VIRTTOOMANYCHILDREN	Dynamic Calc member has more than 100 children
OTLAPI_FAILED_ASSIGN_DEFAULTGENNAMES	Failed to set time related generation names for the date-time dimension created
OTLAPI_ILLEGAL_SCA_TYPE_2	Varying attribute outlines do not allow duplicate names, and cannot be a currency outline
OTLAPI_INVALID_ARG	Invalid argument passed to ESSOTL function
OTLAPI_INVALID_QUERYID	Invalid query id argument passed
OTLAPI_INVALID_QUERY_OPTIONS	Invalid query options passed. Will be ignored
OTLAPI_NO_GENLEVELNAME	Cannot find generation or level name
OTLAPI_NO_USERATTR	Cannot find user attribute
OTLAPI_NULL_ARG	NULL argument passed to EssOtl... function
OTLAPI_OUTLINE_TOO_NEW	Outline is of a newer version than this program can understand
OTLAPI_SORT_TOOMANY	Too many members to sort (64K / 4 members is the maximum sorting capacity)
OTLAPI_SMARTLIST_ASSOC_EXISTS	Cannot delete a text list with existing associations
OTLAPI_SMARTLIST_DUP_IDORNAME	Duplicate text list element ID or name
OTLAPI_SMARTLIST_INVALID_TEXT	Invalid text list text
OTLAPI_WRONG_INDEPDIM_NM	The number of independent dimensions given in perspective does not match the outline

C Outline API DTS Member Structures

These structures contain information about Dynamic Time Series (DTS) members.

```
/*
  ESS_DTSMBRNAME_T, ESS_PDTSMBRNAME_T
  DTS member name structure
```

```
 */
ESS_TSA_ARRAY_API_typedef(char,                 ESS_DTSMBRNAME_T, ESS_MBRNAMELEN);
ESS_TSA_API_typedef       (ESS_DTSMBRNAME_T *,  ESS_PDTSMBRNAME_T);
ESS_TSA_API_typedef       (ESS_PDTSMBRNAME_T *, ESS_PPDTSMBRNAME_T);
```

Data Type	Field	Description
ESS_DTSMBRNAME_T	*szDTSMember*	The name of the DTS member.
ESS_MBRNAMELEN	*szName*	The length of the DTS member name.

```
/*
  ESS_DTSMBRINFO_T, ESS_PDTSMBRINFO_T
  DTS member info structure
 */
ESS_TSA_API_typedef_struct(ess_dtsmbrinfo_t)
{
   ESS_TSA_ELEMENT(ESS_DTSMBRNAME_T, szDTSMember);
   ESS_TSA_ELEMENT(ESS_USHORT_T,     usGen);
} ESS_TSA_END(ESS_DTSMBRINFO_T);

ESS_TSA_API_typedef(ESS_DTSMBRINFO_T *,  ESS_PDTSMBRINFO_T);
ESS_TSA_API_typedef(ESS_DTSMBRINFO_T **, ESS_PPDTSMBRINFO_T);
```

Data Type	Field	Description
ESS_DTSMBRNAME_T	*szDTSMember*	The name of the DTS member.
ESS_USHORT_T	*usGen*	The generation number of the DTS member.

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

Value	Description
ESS_CONV_NONE	Default conversion category. Member inherits category from parent.
ESS_CONV_CATEGORY	Define a Currency Conversion category for this member
ESS_CONV_NOCONV	No conversion for this member

Account Member Time Balance Skip Values

Only valid if time balance is not ESS_TIMEBAL_NONE.

Value	Description
ESS_SKIP_NONE	Don't skip anything
ESS_SKIP_MISSING	Skip the value if the data is #missing
ESS_SKIP_ZEROS	Skip the value if the data is 0
ESS_SKIP_BOTH	Skip the value if the data is #missing or 0

Account Member Time Balance Values

Value	Description
ESS_TIMEBAL_NONE	No time balance
ESS_TIMEBAL_FIRST	First time balance member
ESS_TIMEBAL_LAST	Last time balance member
ESS_TIMEBAL_AVG	Average time balance member

Dimension Categories

Used for optimizing storage when using storage auto-configure

Value	Description
ESS_STORECAT_ACCOUNTS	Accounts storage category
ESS_STORECAT_ATTRCALC	Attribute calculation (aggregation) storage category
ESS_STORECAT_ATTRIBUTE	Attribute storage category
ESS_STORECAT_BUSUNIT	Business Unit storage category

Value	Description
ESS_STORECAT_CUSTOMER	Customer storage category
ESS_STORECAT_DIST	Distribution Channel storage category
ESS_STORECAT_GEOG	Geographical Location storage category
ESS_STORECAT_MARKET	Market storage category
ESS_STORECAT_ORGAN	Organization storage category
ESS_STORECAT_OTHER	None or don't know storage category
ESS_STORECAT_PRODUCT	Product storage category
ESS_STORECAT_SCENARIO	Scenario storage category
ESS_STORECAT_TIME	Time storage category
ESS_STORECAT_UNITS	Units storage category

Dimension Categories (Tags)

Value	Description
ESS_CAT_ACCOUNTS	Accounts dimension
ESS_CAT_ATTRCALC	Attribute calculation dimension or member. Used internally for aggregation.
ESS_CAT_ATTRIBUTE	Attribute dimension or member
ESS_CAT_COUNTRY	Country dimension
ESS_CAT_CURPARTITION	Currency partition dimension. Valid only in non-currency databases.
ESS_CAT_NONE	No category
ESS_CAT_TIME	Time dimension
ESS_CAT_TYPE	Type dimension. Valid only in currency databases.

Member Types

Value	Description
ESS_MEMBERTYPE_NONE	No type
ESS_MEMBERTYPE_NUMERIC	Numeric type
ESS_MEMBERTYPE_SMARTLIST	Text List (SmartList) type

Value	Description
ESS_MEMBERTYPE_DATE	Date type

Generation and Level Options

You can use with **EssOtlGetGenNames**() and **EssOtlGetLevelNames**()

Value	Description
ESS_GENLEV_ALL	Returns default and user-defined names
ESS_GENLEV_ACTUAL	Returns only names that are user-defined
ESS_GENLEV_DEFAULT	Returns all default names, including the default names for generations and levels that also have user-defined names
ESS_GENLEV_NOACTUAL	Returns all default names, excluding the default names for generations and levels that also have user-defined names

Query Options

You can specify for certain query types in "ESS_PREDICATE_T" on page 709

Value	Description
ESS_MEMBERSONLY	Valid for ESS_SEARCH, ESS_WILDSEARCH
ESS_ALIASESONLY	Valid for ESS_SEARCH, ESS_WILDSEARCH
ESS_MEMBERSANDALIASES	Valid for ESS_SEARCH, ESS_WILDSEARCH
ESS_COUNTONLY	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.
ESS_INCLUDEHYBRIDANALYSIS	Includes relational sources if present.
ESS_EXCLUDEHYBRIDANALYSIS	Excludes relational sources if present.

Query Types

Used for defining the operation to perform in "ESS_PREDICATE_T" on page 709:

- 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
- ESS_DIMUSERATTRIBUTES
- ESS_INDEPDIMS
- ESS_SIBLINGS65
- ESS_INDEPDIMS_DISCRETE
- ESS_INDEPDIMS_CONTINUOUS

Restructure Values

Value	Description
ESS_DOR_ALLDATA	Keep all data
ESS_DOR_NODATA	Discard all data
ESS_DOR_LOWDATA	Keep only level 0 data
ESS_DOR_INDATA	Keep only input data
ESS_DOR_FORCE_ALLDATA	Reload all data

Share Constants

Value	Description
ESS_SHARE_DYNCALCNOSTORE	Shared member. A member tagged as no Dynamic Calc and Store.
ESS_SHARE_DYNCALCSTORE	Shared member. A member tagged as Dynamic Calc and Store.
ESS_SHARE_DATA	Normal member (default value)
ESS_SHARE_LABEL	Label member. Do not store data for this member.

Value	Description
ESS_SHARE_NEVER	Never share this member, even if it would normally be an implicit share.
ESS_SHARE_SHARE	Shared member. This member cannot have children and must have the actual member with the same name in the same dimension.

Sorting Options

Value	Description
ESS_SORT_ASCENDING	Sort in ascending order
ESS_SORT_DESCENDING	Sort in descending order
ESS_SORT_USERDEFINED	User supplies a custom sorting routine

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;
```

Data Type	Field	Description
ESS_BOOL_T	*bInputMemberIsHandle*	Boolean value: ● TRUE: attribute query by member handle ● FALSE: attribute query by member name string
ESS_HMEMBER_T ESS_STR_T	*uInputMember* *uInputMember.hMember* *uInputMember.szMember*	A union variable for the following member reference values: ● Member handle ● Member name string

Data Type	Field	Description
ESS_USHORT_T	*usInputMemberType*	A constant identifier indicating the data type of the member queried: ● ESS_ATTRIBUTE_DIMENSION ● ESS_ATTRIBUTE_MEMBER ● ESS_STANDARD_DIMENSION ● ESS_STANDARD_MEMBER ● ESS_BASE_DIMENSION ● ESS_BASE_MEMBER ● ESS_ATTRIBUTED_MEMBER See Table 6, "C API Attributes Terminology," on page 102.
ESS_USHORT_T	*usOutputMemberType*	A constant identifier indicating the data type of the member returned: ● ESS_ATTRIBUTE_DIMENSION ● ESS_ATTRIBUTE_MEMBER ● ESS_STANDARD_DIMENSION ● ESS_STANDARD_MEMBER ● ESS_BASE_DIMENSION ● ESS_BASE_MEMBER ● ESS_ATTRIBUTED_MEMBER ● ESS_INVALID_MEMBER
"ESS_ATTRIBUTEVALUE_T" on page 118	*Attribute*	A structure defining the attribute value for query input
ESS_USHORT_T	*usOperation*	A constant identifier indicating the type of query operation: ● ESS_EQ: equal to ● ESS_NEQ: not equal to ● ESS_GT: greater than ● ESS_LT: less than ● ESS_GTE: greater than or equal to ● ESS_LTE: less than or equal to ● ESS_TYPEOF ● ESS_ALL

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;
```

Data Type	Field	Description
ESS_USHORT_T	*usNumber*	Generation or level number.
ESS_MBRNAME_T	*szName*	Generation or level name.

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;
```

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_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_PPMBRCOUNTS_T;
```

Data Type	Field	Description
ESS_ULONG_T	*ulStart*	Starting member for retrieval of information.
ESS_ULONG_T	*ulMaxCount*	Maximum number of members to retrieve.
ESS_ULONG_T	*ulTotalCount*	Return of the total count of members that exist in the results of the query. This could be more than ulMaxCount.
ESS_ULONG_T	*ulReturnCount*	Return count of returned member handles. This should never be more than ulMaxCount.

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;
```

Data Type	Field	Description
ESS_MBRNAME_T	*szMember*	Member name. This field can be set only by the caller when creating the member.
ESS_USHORT_T	*usLevel*	Level of the member in the outline. This field cannot be modified.
ESS_USHORT_T	*usGen*	Generation of the member in the outline. This field cannot be modified.
ESS_USHORT_T	*usConsolidation*	Unary consolidation type. It can be one of the following: ● ESS_UCALC_ADD ● ESS_UCALC_SUB ● ESS_UCALC_MULT ● ESS_UCALC_DIV ● ESS_UCALC_PERCENT ● ESS_UCALC_NOOP
ESS_BOOL_T	*fTwoPass*	ESS_TRUE if two-pass calculation member.

Data Type	Field	Description
ESS_BOOL_T	*fExpense*	ESS_TRUE if expense member.
ESS_USHORT_T	*usConversion*	Currency Conversion type. This is valid only for members of the Accounts dimension. It can be one of the following: ● ESS_CONV_NONE ● ESS_CONV_CATEGORY ● ESS_CONV_NOCONV
ESS_MBRNAME_T	*szCurMember*	If member is of the Accounts dimension and *usConversion* is ESS_CONV_CATEGORY. This field defines the currency category. If member is of the Country dimension. This field defines the currency name. This field is undefined in all other situations.
ESS_USHORT_T	*usTimeBalance*	Time balance option. Valid field only for members of the Accounts dimension. It can be one of the following: ● ESS_TIMEBAL_NONE ● ESS_TIMEBAL_FIRST ● ESS_TIMEBAL_LAST ● ESS_TIMEBAL_AVG
ESS_USHORT_T	*usSkip*	Time balance skip option. Valid field only for members of the Accounts dimension if *usTimeBalance* is not equal to ESS_TIMEBAL_NONE. It can be one of the following: ● ESS_SKIP_NONE ● ESS_SKIP_MISSING ● ESS_SKIP_ZEROS ● ESS_SKIP_BOTH
ESS_USHORT_T	*usShare*	Share option. It can be one of the following: ● ESS_SHARE_DATA (default value) ● ESS_SHARE_DYNCALCSTORE ● ESS_SHARE_DYNCALCNOSTORE ● ESS_SHARE_LABEL ● ESS_SHARE_NEVER ● ESS_SHARE_SHARE (Valid for level 0 members only)
ESS_USHORT_T	*usStorage*	Dimension storage type. This field is valid only for dimension members and can be one of the following: ● ESS_DIMTYPE_DENSE ● ESS_DIMTYPE_SPARSE

Data Type	Field	Description
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	*fHasHAEnabled*	The dimension has relational partitions enabled. Valid only for Dimension members.

Data Type	Field	Description
RSS_PVOID_T	*pLastSibling*	Last sibling pointer
ESS_ULONG_T	*uSiblingCount*	Sibling count
ESS_BOOL_T	*fFormula*	Indicates whether has a formula
ESS_BOOL_T	*fUda*	Indicates whether has UDA
ESS_BOOL_T	*fAlias*	Indicates whether has alias
ESS_BOOL_T	*fIndependentDim*	For dimensions on varying attribute outlines; indicates if an independent dimension
ESS_UCHAR_T	*ucHierarchyType*	Defines the type of hierarchy based on the generation. If the member is generation 1, then: ● ESS_STORED_HIERARCHY indicates a single stored hierarchy ● ESS_DYNAMIC_HIERARCHY indicates a single dynamic hierarchy ● ESS_MULTIPLE_HIERARCHY_IS_ENABLED indicates multiple hierarchies If the member is generation 2, then: ● ESS_STORED_HIERARCHY indicates a sub hierarchy ● ESS_DYNAMIC_HIERARCHY indicates a dynamic sub hierarchy
ESS_UCHAR_T	*udDimSolveOrder*	Defines the solve order for the dimension.
ESS_UCHAR_T	*udSolveOrder*	Indicates the solve order value. The solve order can be 0-127.
ESS_BOOL_T	*fNonUniqueName*	Indicates whether the member name is unique
ESS_BOOL_T	*fFlow*	Indicates that member is type Flow

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_ULONG_T          ulCount;
   ESS_OTLQUERYERROR_T* ErrorArray;

} ESS_OTLQUERYERRORLIST_T, *ESS_POTLQUERYERRORLIST_T, **ESS_PPOTLQUERYERRORLIST_T;
```

Data Type	Field	Description
ESS_ULONG_T	*ulCount*	Number of errors returned during a query
ESS_OTLQUERYERROR_T*	*ErrorArray*	Pointer to an array of errors returned during a query

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.

```
typedef struct ESS_OUTERROR_T
{
   ESS_HMEMBER_T  hMember;
   ESS_ULONG_T    ulErrors;
} ESS_OUTERROR_T, *ESS_POUTERROR_T, **ESS_PPOUTERROR_T;
```

Data Type	Field	Description
ESS_HMEMBER_T	*hMember*	Handle to member with errors.
ESS_ULONG_T	*ulErrors*	Bitmask of errors for the member. See Values for *ulErrors*.

Values for *ulErrors*

The following are possible values for *ulErrors*:

- ESS_OUTERROR_ALIASSHARED
- ESS_OUTERROR3_ASO_BAD_AGGREGATION_OPERATOR
- ESS_OUTERROR3_ASO_BAD_NONLEAFMBR
- ESS_OUTERROR3_ASO_DYNASSOCD
- ESS_OUTERROR3_ASO_EITHERLABELORFORMULA
- ESS_OUTERROR3_ASO_INVALID_AGGLEVELUSAGE
- ESS_OUTERROR3_ASO_INVALIDATTRCALC
- ESS_OUTERROR3_ASO_ISDUPLICATESHAREINHIERARCHY
- ESS_OUTERROR3_ASO_LABEL_SPAN
- ESS_OUTERROR3_ASO_LEVELPRODUCT_TOO_LARGE
- ESS_OUTERROR3_ASO_NOATTRIBUTE_ON_ACCOUNTS
- ESS_OUTERROR3_ASO_NOFORMULA
- ESS_OUTERROR4_ASO_PROTOLEVELZERO
- ESS_OUTERROR3_ASO_SHAREDMBR
- ESS_OUTERROR3_ASO_TWOCHILDRENFORTHISOPER
- ESS_OUTERROR3_ASO_WHOLEACCOUNTSDIMVIRTUAL
- ESS_OUTERROR2_ATTRCALCABSENT
- ESS_OUTERROR2_ATTRDIMNOTASSOCIATED
- ESS_OUTERROR_BADATTRIBUTECODE
- ESS_OUTERROR_BADCATEGORY
- ESS_OUTERROR_BADSHARE

- ESS_OUTERROR_BADSKIP
- ESS_OUTERROR_BADSTORAGE
- ESS_OUTERROR_BADSTORAGECATEGORY
- ESS_OUTERROR_BADTIMEBAL
- ESS_OUTERROR2_BOOLEANNAMESETTING
- ESS_OUTERROR2_CHILDCOUNT
- ESS_OUTERROR_CURTOOMANYDIMS
- ESS_OUTERROR2_DATATYPEMISMATCH
- 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_ILLEGALATTRIBUTEPARENT
- 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_ILLEGALUDA
- 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.

```
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;
```

Data Type	Field	Description
ESS_BOOL_T	*fCaseSensitive*	Case-sensitive member names flag.
ESS_USHORT_T	*usOutlineType*	Type of the outline. It can be one of the following: ● ESS_DBTYPE_NORMAL Normal database ● ESS_DBTYPE_CURRENCY Currency database ● ESS_DBTYPE_NORMALMDX Database with MDX type formula ● ESS_DBTYPE_ASO Aggregate storage database ● ESS_DBTYPE_ROLAP ROLAP database ● ESS_DBTYPE_ASO71 Aggregate storage database with version 7.1 otl file
ESS_BOOL_T	*fAutoConfigure*	ESS_TRUE to automatically configure the dimension storage (dense/sparse) when a block-storage outline is saved.
ESS_USHORT_T	*usNumAliasTables*	Number of alias tables. This is a read-only field and will be ignored in the **EssOtlSetOutlineInfo()** call.
ESS_ ALIASNAME_T	*pAliasTables*	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.
ESS_BOOL_T	*fEnableVaryingAttrs*	ESS_TRUE indicates the outline supports varying attributes.
ESS_BOOL_T	*fNonUniqueName*	Indicates whether the outline supports duplicate member names.
ESS_UCHAR_T	*ucImpliedShareSetting*	Implied Share setting: ● TRUE (default) means that Implied Share is ON ● FALSE means that Implied Share is OFF
ESS_BOOL_T	*fEnableMemberType*	ESS_TRUE indicates member types are enabled.

ESS_OUTLINEINFOEX_T

Contains information about the outline.

```
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_ALIASNAME_T      pAliasTables[1];
   ESS_BOOL_T           fEnableVaryingAttrs;
   ESS_UCHAR_T,         ucImpliedShareSetting
   ESS_BOOL_T,          fEnableMemberType;
   ESS_CHAR_T,          cSMDateFormatValue;
} ESS_OUTLINEINFOEX_T, *ESS_POUTLINEINFOEX_T, **ESS_PPOUTLINEINFOEX_T;
```

Data Type	Field	Description
ESS_BOOL_T	*fCaseSensitive*	Case-sensitive member names flag.
ESS_USHORT_T	*usOutlineType*	Type of the outline. It can be one of these: ● ESS_DBTYPE_NORMAL ● ESS_DBTYPE_CURRENCY
ESS_BOOL_T	*fAutoConfigure*	ESS_TRUE to automatically configure the dimension storage (dense/sparse) when a block-storage outline is saved.
ESS_BOOL_T	*fNonUniqueName*	Indicates whether the outline supports duplicate member names.
ESS_USHORT_T	*usNumAliasTables*	Number of alias tables. This is a read-only field and is ignored in the **EssOtlSetOutlineInfo()** call.
ESS_ALIASNAME_T	*pAliasTables*	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.
ESS_BOOL_T	*fEnableVaryingAttrs*	ESS_TRUE indicates the outline supports varying attributes.
ESS_UCHAR_T	*ucImpliedShareSetting*	Implied share setting for the outline. Possible values: ● ESS_IMPLIEDSHARE_DEFAULT ● ESS_IMPLIEDSHARE_DEFAULT_ON ● ESS_IMPLIEDSHARE_DEFAULT_OFF ● ESS_IMPLIEDSHARE_FORCE_ON ● ESS_IMPLIEDSHARE_FORCE_OFF
ESS_BOOL_T	*fEnableMemberType*	
ESS_UCHAR_T	*cSMDateFormatValue*	

ESS_PERSPECTIVE_T

Contains information about perspectives and validity sets.

```
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
```

Data Type	Field	Description
ESS_USHORT_T	*usValiditySetType*	How members are specified. Possible values: • ESS_VALIDITYSET_TYPE_MBRHDLS • ESS_VALIDITYSET_TYPE_MBRNAMS
ESS_USHORT_T	*usFiller*	Set to zero
ESS_STR_T	*szValiditySetExpr*	MDX expression spescified by the MDX type
ESS_INT32_T	*countOfIndepDims*	Size of each of the tuples
ESS_INT32_T	*countOfIndepRanges*	Number of tuple ranges
ESS_PVOID_T	*pIndepMbrs*	Array of member handles (ESS_HMEMBER_T) or member names (ESS_STR_T) depending on usValiditySetType

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;
```

Data Type	Field	Description
ESS_ULONG_T	*ulQuery*	Type of query. See EssOtlQueryMembers for more information.

Data Type	Field	Description
ESS_ULONG_T	*ulOptions*	Options dependent on the query type. See EssOtlQueryMembers for more information.
ESS_STR_T	*pszDimension*	Dimension name. See EssOtlQueryMembers for more information.
ESS_STR_T	*pszString1*	Input string value. See EssOtlQueryMembers for more information.
ESS_STR_T	*pszString2*	Input string value. See EssOtlQueryMembers for more information.

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;
```

Data Type	Field	Description
ESS_BOOL_T	*fCaseSensitive*	Case-sensitive member names flag.
ESS_USHORT_T	*usOutlineType*	Type of the outline. It can be one of the following: • ESS_DBTYPE_NORMAL • ESS_DBTYPE_CURRENCY
ESS_BOOL_T	*fNonUniqueName*	Indicates whether the outline supports duplicate member names.
ESS_USHORT_T	*usNumAliasTables*	Number of alias tables. This is a read-only field and will be ignored in the **EssOtlSetOutlineInfo()** call.
ESS_ALIASNAME_T	*pAliasTables*	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.

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;
```

```
   ESS_INT32_T, countOfIndepRanges;
   ESS_PVOID_T*, pIndepMbrs;
} ESS_VALIDITYSET_T; *ESS_PVALIDITYSET_T
```

Data Type	Field	Description
ESS_USHORT_T	*usValiditySetType*	How members are specified. Possible values: ● ESS_VALIDITYSET_TYPE_MBRHDLS ● ESS_VALIDITYSET_TYPE_MBRNAMS
ESS_USHORT_T	*usFiller*	Set to zero
ESS_STR_T	*szValiditySetExpr*	MDX expression spescified by the MDX type
ESS_INT32_T	*countOfIndepDims*	Size of each of the tuples
ESS_INT32_T	*countOfIndepRanges*	Number of tuple ranges
ESS_PVOID_T	*pIndepMbrs*	Array of member handles (ESS_HMEMBER_T) or member names (ESS_STR_T) depending on usValiditySetType

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.

11 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.

Function	Description
`EssOtlCreateAliasTable`()	Creates an empty alias table in the outline

Function	Description
EssOtlCopyAliasTable	Copies an alias table to another alias table
EssOtlRenameAliasTable	Renames an existing alias table
EssOtlClearAliasTable	Clears all entries from an existing alias table without deleting the alias table
EssOtlDeleteAliasTable	Deletes the alias table from the outline and clears all of its entries
EssOtlSetAliasTableLanguage	Sets a language code for the alias table. An alias table can have multiple language codes.
EssOtlGetAliasTableLanguages	Gets the set of language codes associated with the alias table.
EssOtlClearAliasTableLanguages	Clears the set of language codes from the alias table.

C Outline API Attributes Functions

These C Outline functions are for attributes.

See also "C Outline API Varying Attributes Functions" on page 719.

Function	Description
EssOtlAssociateAttributeDimension	Associates an attribute dimension with a base dimension
EssOtlAssociateAttributeMember	Associates an attribute member with a base dimension member
EssOtlDisassociateAttributeDimension	Disassociates an attribute dimension from a base dimension
EssOtlDisassociateAttributeMember	Disassociates an attribute member from a base dimension member
EssOtlFindAttributeMembers	Returns all base dimension members that are associated with an attribute member
EssOtlFreeStructure	Frees memory dynamically allocated for string type attribute information
EssOtlGetAssociatedAttributes	Returns all attribute dimension members that are associated with a base dimension member or base dimension
EssOtlGetAttributeInfo	Returns attribute information for a given attribute member or dimension
EssOtlGetAttributeSpecifications	Retrieves attribute specifications for the outline
EssOtlQueryAttributes	Queries the outline for member attribute information
EssOtlQueryAttributesEx	
EssOtlSetAttributeSpecifications	Sets attribute specifications for the outline

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.

Function	Description
EssOtlDeleteDTSMemberAlias	Deletes an alias name for a Dynamic Time Series member.
EssOtlEnableDTSMember	Enables a new Dynamic Time Series members for the outline.
EssOtlGetEnabledDTSMembers	Gets the defined Dynamic Time Series members for the outline.
EssOtlGetDTSMemberAlias	Gets an alias name for a Dynamic Time Series member.
EssOtlSetDTSMemberAlias	Sets an alias name for a Dynamic Time Series member.

C Outline API Generation Name Functions

These functions perform operations on generation names.

Function	Description
EssOtlGetGenName	Gets the generation name for the specified dimension and generation number
EssOtlGetGenNames	Retrieves all generation names specified for a particular dimension
EssOtlSetGenName	Sets the generation name for the specified dimension and generation number
EssOtlDeleteGenName	Deletes the generation name of the specified dimension and level number

C Outline API Level Name Functions

These functions perform operations on level names.

Function	Description
EssOtlGetLevelName	Gets the level name of the specified dimension
EssOtlGetLevelNames	Retrieves all level names specified for a particular dimension
EssOtlSetLevelName	Sets the level name of the specified dimension
EssOtlDeleteLevelName	Deletes the level name of the specified dimension

C Outline API Member Administration Functions

These functions assist in managing the members of an outline.

Function	Description
EssOtlAddMember	Adds a member
EssOtlDeleteMember	Deletes a member
EssOtlAddDimension	Adds a dimension
EssOtlDeleteDimension	Deletes a dimension
EssOtlRenameMember	Renames a member
EssOtlMoveMember	Moves a member
EssOtlFindMember	Finds a member
EssOtlGetMemberCommentEx	Gets the extended comment for a specified member
EssOtlGetMemberInfo	Gets member information
EssOtlSetMemberCommentEx	Sets the extended comment for a specified member
EssOtlSetMemberInfo	Sets member information
EssOtlGetMemberSolveOrder	Gets member solve order
EssOtlSetMemberSolveOrder	Sets member solve order
EssOtlGetDimensionSolveOrder	Gets dimension solve order
EssOtlSetDimensionSolveOrder	Sets dimension solve order

C Outline API Member Alias Functions

These functions perform operations on member aliases.

Function	Description
EssOtlFindAlias	Finds a member with the specified alias name
EssOtlGetMemberAlias	Gets the default member alias for a specific member in a specific alias table
EssOtlSetMemberAlias	Sets the default member alias for a specific member in a specific alias table
EssOtlDeleteMemberAlias	Deletes the default member alias for a specific member in a specific alias table

C Outline API Member Formula Functions

These functions perform operations on member formulas.

Function	Description
EssOtlGetMemberFormula	Gets the formula of the specified member
EssOtlGetMemberLastFormula	Returns the last formula used to calculate the member
EssOtlSetMemberFormula	Sets the formula for the specified member
EssOtlDeleteMemberFormula	Deletes the formula of the specified member

C Outline API Member Traversal Functions

These functions are used in traversing the outline tree.

Function	Description
EssOtlGetFirstMember	Returns a member handle to the first member in the outline; the first dimension defined in the outline
EssOtlGetChild	Returns a member handle to the child of a member
EssOtlGetParent	Returns a member handle to the parent of a member
EssOtlGetNextSibling	Returns a member handle to the next sibling of a member
EssOtlGetPrevSibling	Returns a member handle to the previous sibling of a member
EssOtlGetNextSharedMember	Returns a member handle to the next shared member of a real member
EssOtlQueryGetFirstDimension	Returns the dimension handle of the first dimension in the outline
EssOtlQueryGetNextDimension()	Returns the next dimension handle of the dimension in the outline opened in query mode

C Outline API Outline Administration Functions

These functions assist in managing outlines.

Function	Description
EssOtlGetOutlineInfo	Returns information about the outline file
EssOtlGetUpdateTime	Returns the timestamp for the specified outline
EssOtlSetOutlineInfo	Sets outline information
EssOtlVerifyOutline	Verifies that an outline is correct
EssOtlSortChildren	Sorts the children of an outline member
EssOtlGenerateCurrencyOutline	Generates a currency outline based on the existing outline
EssOtlGetASOCompressionDimension	Gets aggregate storage compression dimension

Function	Description
EssOtlSetASOCompressionDimension	Sets aggregate storage compression dimension

C Outline API Outline Query Functions

These functions assist in making outline queries.

Function	Description
EssOtlGetMemberField	Returns data for the specified field of a specified outline member
EssOtlOpenOutlineQuery	Opens an existing outline
EssOtlQueryMembers	Queries the outline, using a member handle
EssOtlQueryMembersByName	Queries the outline, using a member name string
EssOtlQueryMembersEx	Queries specific members and member fields, and returns an array of member handles
EssOtlQueryAttributes	Queries the outline for attribute information.
EssOtlQueryAttributesEx	
EssOtlFreeMembers	Frees the member array returned from **EssOtlQueryMembers()**

C Outline API Setup and Cleanup Functions

These functions start and finish editing operations on an outline.

Function	Description
EssOtlNewOutline	Creates a new outline
EssOtlOpenOutline	Opens an existing outline
EssOtlOpenOutlineEx	Opens an existing outline (for Unicode mode)
EssOtlWriteOutline	Writes the outline to the server
EssOtlWriteOutlineEx	Writes the outline to the server (for Unicode mode)
EssOtlRestructure	Restructures the database based on the newly saved outline
EssOtlCloseOutline	Frees resources associated with the outline

C Outline API Unicode Mode Functions

The following functions help you work with the Essbase Server outlines in Unicode mode.

Function	Description
EssOtlWriteOutlineEx	Writes the outline to the server, specifying whether to save in UTF-8 encoding or in non-Unicode encoding.
EssOtlOpenOutlineEx	Opens the outline of a Unicode-mode application.

C Outline API User-Defined Attributes Functions

These functions perform operations on user-defined attributes (UDAs).

Function	Description
EssOtlGetDimensionUserAttributes	Gets the UDAs of the specified dimension
EssOtlGetUserAttributes	Gets the UDAs of the specified member
EssOtlSetUserAttribute	Sets a UDA for the specified member
EssOtlDeleteUserAttribute	Deletes a UDA of the specified member

C Outline API User-Defined View Selection Functions

These functions define view selection criteria for aggregation of aggregate storage databases.

Function	Description
EssOtlSetAggLevelUsage	Applies view selection properties to stored hierarchies
EssOtlGetAggLevelUsage	Returns the applied view selection properties on stored hierarchies
EssOtlAddQueryHint	Adds a query hint to the outline to aid in view selection
EssOtlGetQueryHint	Returns specified query hint defined on an outline
EssOtlSetQueryHint	Sets a query hint
EssOtlGetNumQueryHints	Returns the number of query hints
EssOtlGetQueryHintSize	Returns the size (in number of members) of query hints
EssOtlDeleteQueryHint	Deletes specified query hint and decreases the number of hints by one

C Outline API Varying Attributes Functions

These C Outline functions are for varying attributes.

Function	Description
EssOtlQueryVaryingAttributes	Queries the outline for member varying attribute information

Function	Description
EssOtlDetailQueryVaryingAttributes	Similar to EssOtlQueryVaryingAttributes.
EssOtlVaryingAssociateAttribute	Associates a varying attribute member with a base dimension member
EssOtlVaryingAssociateAttributeDimension	Associates a varying attribute dimension with a base dimension
EssOtlVaryingDisassociateAttribute	Disassociates a varying attribute dimension from a base dimension
EssOtlVaryingGetAssociatedAttributes	Returns all varying attribute members that are associated with a base dimension member or base dimension
EssOtlVaryingGetAttributeIndepDims	Returns the independent dimensions, if any, for the dimension containing the specified varying attribute member

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:

```
Jan       Feb       Mar       Apr       May       Jun
5540      2190      1580      300       2455      3255
Jones     Jones     Jones     Jones     Smith     Smith
```

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

Term	Definition
independent dimensions	The dimension upon which varying attributes depend; in the above example, the Year dimension.
perspective	A combination of independent dimension members that is used when querying for associations. Defined in "ESS_PERSPECTIVE_T" on page 708.
validity set	The collection of independent dimension members for which an association is true. Defined in "ESS_VALIDITYSET_T" on page 710.

Outline Construction

Varying attributes are constructed in the API with the following flow:

Item	Outline API Call
1. Set the outline type to accept varying attributes	EssOtlSetOutlineInfo, where pOutlineinfo->fEnableVaryingAttrs = ESS_TRUE.
2. Identify the independent dimension	EssOtlSetMemberInfo, where pMemberInfo->fIndependentDim = ESS_TRUE
3. Associate the attribute dimension to the base dimension and identify independent dimensions	EssOtlVaryingAssociateAttributeDimension
4. Associate attribute dimension members independent dimension members with base dimension members	EssOtlVaryingAssociateAttribute
5. Save and restructure the outline.	The same as when making other outline changes.

Maintenance Tasks

Item	Outline API Call
Add a new association to independent members.	EssOtlVaryingAssociateAttribute
Remove independent member associations	EssOtlVaryingDisassociateAttribute
View existing independent dimension member associations	EssOtlQueryVaryingAttributes or EssOtlVaryingGetAssociatedAttributes
Disassociate attribute dimensions from base dimensions	EssOtlDisassociateAttributeDimension (disassociates all attribute dimensions).

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, pszDataMbr,
phMember);
```

Parameter	Data Type	Description
hOutline;	ESS_HOUTLINE_T	Outline context handle.
pMemberInfo;	"ESS_MBRINFO_T" on page 699	Member information structure defining the member and its attributes.
hPrevSibling	ESS_HMEMBER_T	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*.
pszDataMbr;	ESS_STR_T	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.
phMember	ESS_PHMEMBER_T	Handle of new member returned from the API.

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:

Field	Setting
usConsolidation	ESS_UCALC_NOOP
fTwoPass	ESS_FALSE
fExpense	ESS_FALSE
usConversion	ESS_CONV_NONE
usTimeBalance	ESS_TIMEBAL_NONE
usSkip	ESS_SKIP_NONE
usShare	ESS_SHARE_DYNCALCNOSTORE
usStorage	ESS_DIMTYPE_SPARSE

Field	Setting
usCategory	ESS_CAT_ATTRIBUTE
usStorageCategory	ESS_STORECAT_ATTRIBUTE
Attribute.*usDataType*	One of the following attribute member data types: ❍ ESS_ATTRMBRDT_BOOL ❍ ESS_ATTRMBRDT_DATETIME ❍ ESS_ATTRMBRDT_DOUBLE ❍ ESS_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
- OOTLAPI_ERR_ADDNAMEUSEDTLAPI_ERR_ADDDELETEDIMDYNAMICCALC
- 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

```
#include <essapi.h>
#include <essotl.h>

ESS_STS_T           sts = 0;
ESS_OUTLINEINFO_T  NewInfo;
```

```
ESS_HOUTLINE_T     hOutline;
ESS_MBRINFO_T      MbrInfo;
ESS_HMEMBER_T      hDimMeasures;

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

```
ESS_FUNC_M EssOtlAddQueryHint (hOutline, numMembers, pMemberArray);
```

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Outline context handle (input).
numMembers	ESS_SHORT_T	Number of members in the array provided - usually the number of real dimensions in the outline. (input)
pMemberArray	ESS_PHMEMBER_T	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.

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

```
   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");
   }
```

See Also

- EssOtlSetQueryHint
- EssOtlGetQueryHint
- EssOtlGetNumQueryHints
- EssOtlGetQueryHintSize
- EssOtlDeleteQueryHint

EssOtlAddMember

Adds a member to the outline and sets the member's attributes.

Syntax

```
ESS_FUNC_M EssOtlAddMember (hOutline, pMemberInfo, hParent, hPrevSibling, phMember);
```

Parameter	Data Type	Description
hOutline;	ESS_HOUTLINE_T	Outline context handle.
pMemberInfo;	"ESS_MBRINFO_T" on page 699	Member information structure defining the member and its attributes.
hParent;	ESS_HMEMBER_T	Handle of parent. This field is used only if the *hPrevSibling* field is ESS_NULL.
hPrevSibling;	ESS_HMEMBER_T	Handle of previous sibling.
phMember;	ESS_PHMEMBER_T	Handle of new member returned from the API.

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 **EssOtlSetMemberInfo**() to set the label tag of the label member.

- To add an attribute member, set the fields of ESS_MBRINFO_T as follows:

Field	Setting
usConsolidation	ESS_UCALC_NOOP
fTwoPass	ESS_FALSE
fExpense	ESS_FALSE
usConversion	ESS_CONV_NONE
usTimeBalance	ESS_TIMEBAL_NONE
usSkip	ESS_SKIP_NONE
usShare	ESS_SHARE_DYNCALCNOSTORE
usStorage	ESS_DIMTYPE_SPARSE
usCategory	ESS_CAT_ATTRIBUTE
usStorageCategory	ESS_STORECAT_ATTRIBUTE
Attribute.*usDataType*	For an attribute dimension or zero-level (leaf node) attribute member, set one of the following data types: - ESS_ATTRMBRDT_BOOL - ESS_ATTRMBRDT_DATETIME - ESS_ATTRMBRDT_DOUBLE - ESS_ATTRMBRDT_STRING 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_ATTRMBR_AUTO.

  - **Notes on Adding an Attribute Member:**
    - Adding a zero-level attribute member that is not of type ESS_ATTRMBRDT_STRING also sets the *szMember* field of the 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 *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 ESS_MBRINFO_T structure.
    - For a zero-level attribute member that is not of type ESS_ATTRMBRDT_STRING, do not set the *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 **EssOtlAddDimension**(). Do not call **EssOtlAddMember**().
    - To add a dimension that is not an attribute dimension, call either **EssOtlAddDimension**() 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 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

```
#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;

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, "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

```
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

Parameter	Data Type	Description
hOutline;	ESS_HOUTLINE_T	Handle to the outline
hStandardMember;	ESS_HMEMBER_T	Handle to the standard or base member
hAttributeMember;	ESS_HMEMBER_T	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 **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

```
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

```
#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

```
ESS_FUNC_M EssOtlClearAliasTableLanguages (hOutline, pszAliasTable);
```

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Outline context handle.
pszAliasTable	ESS_STR_T	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

```
#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

```
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

```
#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

```
ESS_FUNC_M EssOtlCompactOutline (hCtx, filename);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	API context acquired during login
filename	ESS_STR_T	Path and outline file to be compacted

Return Value

Returns 0 if successful. The compacted file is named the same with extension *.otn* and is available in the path specified.

Example

```
#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\\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
        };

        /* get appname and dbname from the argument list */
        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

```
ESS_FUNC_M EssOtlCopyAliasTable (hOutline, pszSourceAliasTable, pszDestAliasTable,
fMerge);
```

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Outline context handle.
pszSourceAliasTable	ESS_STR_T	Name of alias table to copy from. If this parameter is ESS_NULL, the default alias table is used.

Parameter	Data Type	Description
pszDestAliasTable	ESS_STR_T	Name of alias table to copy to. Cannot be the same as *pszSourceAliasTable*.
fMerge	ESS_BOOL_T	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.

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

```
#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);
}
```

See Also

- EssOtlCreateAliasTable
- EssOtlClearAliasTable
- EssOtlRenameAliasTable
- EssOtlDeleteAliasTable
- EssOtlSetAliasTableLanguage

EssOtlCreateAliasTable

Creates an empty alias table in the outline.

Syntax

```
ESS_FUNC_M EssOtlCreateAliasTable (hOutline, pszAliasTable);
```

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Outline context handle.
pszAliasTable	ESS_STR_T	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 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
hOutline	ESS_HOUTLINE_T	Outline handle (Edit mode only)
objType	ESS_OBJECT_TYPES	Object type with one of the following values: • OBJECT_SMARTLIST Object type is Text List (SmartList)/
name	ESS_STR_T	String identifying the object
phObjHandle	ESS_PHOBJECT_T	Returns the created object handle.

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

```
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;
```

```
    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);

        /* Free resources */
    if(ObjHandles)
        EssFree (hInst, 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

EssOtlDeleteAliasTable

Deletes the specified alias table from the outline, clearing all of its entries.

Syntax

```
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

```
#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);
```

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Outline context handle.
hMember	ESS_HMEMBER_T	Handle of member to delete.
pszDataMbr	ESS_STR_T	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.

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

```
#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);
```

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	The Essbase outline handle returned from the EssOtlOpenOutline call.
pszDTSMember	ESS_STR_T	Name of the DTS member which provides the alias.
pzsAliasTable	ESS_STR_T	Name of the alias table which provides the alias. If NULL, use the default alias 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

```
#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_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(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_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

- EssOtlSetDTSMemberAlias

EssOtlDeleteGenName

Deletes the name of a specific generation within a dimension.

Syntax

```
ESS_FUNC_M EssOtlDeleteGenName (hOutline, pszDimension, usGen);
```

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Outline context handle.
pszDimension;	ESS_STR_T	Name of the dimension that contains the generation.
usGen	ESS_USHORT_T	Number of generation 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

```
#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,
```

```
          GenNum);
}
```

See Also

- EssOtlGetGenName
- EssOtlSetGenName

EssOtlDeleteLevelName

Deletes the name for a specific level within a dimension.

Syntax

```
ESS_FUNC_M EssOtlDeleteLevelName (hOutline, pszDimension, usLevel);
```

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Outline context handle.
pszDimension	ESS_STR_T	Name of the dimension that contains the level name.
usLevel	ESS_USHORT_T	Number of level for which to delete name. Leaf members are level 0.

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

```
#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);

/**********  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;
```

```
    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

EssOtlDeleteMember

Deletes a member from the outline.

Syntax

```
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

```
#include <essapi.h>
#include <essotl.h>

ESS_STS_T          sts = 0;
ESS_OBJDEF_T       Object;
```

```
ESS_HOUTLINE_T     hOutline;
ESS_HMEMBER_T      hCOGS;
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, "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

```
ESS_FUNC_M EssOtlDeleteMemberAlias (hOutline, hMember, pszAliasTable);
```

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Outline context handle.
hMember	ESS_HMEMBER_T	Handle of member to delete the alias from.
pszAliasTable	ESS_STR_T	Alias table to delete the alias from. If this parameter is ESS_NULL, the default table is used.

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;
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 = 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

```
#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 *hintNum* greater than the deleted hint are renumbered to *hintNum* – 1.

Syntax

```
ESS_FUNC_M EssOtlDeleteQueryHint (hOutline, hintNum);
```

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Outline context handle (input).
hintNum	ESS_SHORT_T	Query hint number (input).

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

```
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

- EssOtlGetNumQueryHints
- EssOtlGetQueryHintSize

EssOtlDeleteUserAttribute

Deletes a user-defined attribute of a member.

Syntax

```
ESS_FUNC_M EssOtlDeleteUserAttribute (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          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          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
hOutline	ESS_HOUTLINE_T	Outline context handle (input)
pAttrQuery	ESS_PATTRIBUTEQUERY_T	Pointer to the structure that defines the query
pCount	ESS_PMBRCOUNTS_T	Pointer to the number of base members returned
pphReturnedMemberArray	ESS_PPHMEMBER_T	Pointer to the array of returned member handles
pphDetailMemberArray	ESS_PPHMEMBER_T	Pointer to the array of member details

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

```
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	Data Type	Description
hOutline	ESS_HOUTLINE_T	Outline context handle
pAttrQuery	ESS_PVARYING_ATTRIBUTEQUERY_T	Pointer to the structure that defines the query
pPerspective	ESS_PPERSPECTIVE_T	Pointer to a collection of independent members used when querying the client or server for associations
pCount	ESS_PMBRCOUNTS_T	Pointer to the number of base members returned
pphMembers	ESS_PPHMEMBER_T	Pointer to the array of attribute member handles
pphDetailMemberArray	ESS_PPHMEMBER_T	Pointer to the array of attribute member detail handles
usValiditySetType	ESS_USHORT_T	See "ESS_VALIDITYSET_T" on page 710.
**pppValiditySets	ESS_PVALIDITYSET_T	Pointer to the validity set array

Return Value

Returns:

- 0—If successful
- Error number—If unsuccessful

Example

```
void TestEssOtlDetaileQueryVaringAttributes()
{
    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_MBRCOUNTS_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";
    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];

    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

Parameter	Data Type	Description
hOutline;	ESS_HOUTLINE_T	Handle to the outline
hBaseDimension;	ESS_HMEMBER_T	Handle to the base dimension
hAttributeDimension;	ESS_HMEMBER_T	Handle to the attribute dimension

Notes

When you dissassociate an attribute dimension from a base dimension, you disassociate all members of the attribute dimension from members of the base dimension.

Example

```
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;

   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*);

Parameter	Data Type	Description
hOutline;	ESS_HOUTLINE_T	Handle to the outline
hBaseMember;	ESS_HMEMBER_T	Handle to the base member
hAttributeMember;	ESS_HMEMBER_T	Handle to the attribute member

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

```
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

- EssGetAttributeInfo
- EssGetAttributeSpecifications
- EssOtlAssociateAttributeDimension
- EssOtlAssociateAttributeMember
- EssOtlDisassociateAttributeDimension
- EssOtlFindAttributeMembers
- EssOtlFreeStructure
- EssOtlGetAssociatedAttributes
- EssOtlGetAttributeInfo
- EssOtlGetAttributeSpecifications
- EssOtlQueryAttributes
- EssOtlSetAttributeSpecifications

EssOtlEnableDTSMember

Enables a new DTS member for the outline.

Syntax

```
ESS_FUNC_M EssOtlEnableDTSMember (hOutline, pszDTSMember,
usGen, bEnable);
```

Parameter	Data Type	Description
hOutline;	ESS_HOUTLINE_T	The Essbase outline handle returned from the EssOtlOpenOutline call.
pszDTSMember;	ESS_STR_T	Name of the DTS member
usGen;	ESS_USHORT _T	Generation to assign to the DTS member
bEnable;	ESS_BOOL_T	Flag to enable the DTS member

Notes

This function also fills in the ESS_DTSMBRNAME_T structure passed to it.

Return Value

Returns zero if successful.

Example

```
#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;
```

```
        ESS_ULONG_T     ulErrors;
        ESS_ULONG_T     ulCount;
        ESS_POUTERROR_T pMbrErrors = NULL;

        strcpy(szAppName, "1Sample");
        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_TRUE, ESS_TRUE, &hOutline);
        if(sts)
        {
                printf("Could not open outline\n");
                return sts;
        }

        sts = EssOtlEnableDTSMember(hOutline, "H-T-D", 1, ESS_TRUE);
        if(sts)
        {
                printf("Could not enable DTS member alias\n");
        }

        sts = EssOtlVerifyOutline(hOutline, &ulErrors, &ulCount, &pMbrErrors);
        if(sts)
        {
                printf("Could not verify outline\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);
        {
                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);
```

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Outline context handle.
pszAlias	ESS_STR_T	Alias name to find. Can be a simple alias name or a qualified alias name (distinguishing this member from another member having the same name). For information about syntax used to specify a qualified alias name, see the *Oracle Essbase Database Administrator's Guide* section entitled "Creating and Working With Duplicate Member Outlines."
pszAliasTable	ESS_STR_T	Alias table to search in. Use ESS_NULL to search all alias tables. Use "Default" to search the default alias table.
phMember	ESS_PHMEMBER_T	Variable for the return of the member handle. ESS_NULL if the member is not found.

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

```
#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

Parameter	Data Type	Description
hOutline;	ESS_HOUTLINE_T	Handle to the outline
pszMember;	ESS_STR_T	Attribute short name
pszDimName;	ESS_STR_T	Attribute dimension name (optional)
pusCount;	ESS_PUSHORT_T	Number of base members returned
pphMembers;	ESS_PPHMEMBER_T	Pointer to an array of base member handles

Notes

- *pszMember* must be a short name.
- *pszDimName* is optional. You may enter NULL.

Example

```
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

- EssOtlAssociateAttributeDimension
- EssOtlAssociateAttributeMember
- EssOtlDisassociateAttributeDimension
- EssOtlDisassociateAttributeMember
- EssOtlFreeStructure
- EssOtlGetAssociatedAttributes
- EssOtlGetAttributeInfo
- EssOtlGetAttributeSpecifications
- EssOtlQueryAttributes
- EssOtlSetAttributeSpecifications

EssOtlFindMember

Finds a member with the specified name and returns a handle to the member.

Syntax

```
ESS_FUNC_M EssOtlFindMember (hOutline, pszMember, phMember);
```

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Outline context handle.
pszMember	ESS_STR_T	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 "Creating and Working With Duplicate Member Outlines."
phMember	ESS_PHMEMBER_T	Variable for the return of the member handle. ESS_NULL if the member is not found.

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

```
#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

```
ESS_FUNC_M EssOtlFindObject(hOutline, objType, objName, pObjHandle)
```

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Outline handle (Edit mode only)
objType	ESS_OBJECT_TYPES	Object type with one of the following values: • OBJECT_SMARTLIST Object type is Text List (SmartList)
objName	ESS_STR_T	String identifying the object
phObjHandle	ESS_PHOBJECT_T	Returns the found object handle.

Return Value

Returns:

- 0—If successful

  phObjHandle contains the object handle.

- Error number—If unsuccessful

  phObjHandle in NULL.

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;
    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);
```

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Essbase outline handle. This must have been returned from **EssOtlOpenOutlineQuery**().
ulCount	ESS_ULONG_T	The number of elements in the phMember array.
phMembers	ESS_PHMEMBER_T	An array of member handles to be freed.

Return Value

The return value is zero if the function was successful.

Example

See the example for EssOtlQueryMembers.

See Also

- EssOtlOpenOutlineQuery
- EssOtlQueryMembers
- EssOtlQueryMembersByName

EssOtlFreeSmartListInfo

Frees the Text List (SmartList) object obtained by EssOtlGetSmartListInfo.

Syntax

```
ESS_FUNC_M EssOtlFreeSmartListInfo(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

```
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);

    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)
```

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Outline handle (Query mode only)
count	ESS_ULONG_T	Count of object handles
objHandles	ESS_PHOBJECT_T	Array of object handles to be de allocated

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

- EssOtlListObjects
- EssOtlQueryObjects
- EssOtlSetMemberType
- EssOtlSetMemberTypeToSmartList

EssOtlFreeStructure

Frees memory dynamically allocated by **EssOtlGetAttributeInfo**() and **EssOtlGetMemberInfo**() for string type attribute information.

Syntax

Parameter	Data Type	Description
hOutline;	ESS_HOUTLINE_T	Handle to the outline
structId;	ESS_ULONG_T	One of the following constant identifiers for the structure: • ESS_DT_STRUCT_ATTRIBUTEINFO • ESS_DT_STRUCT_ATTRSPECS • ESS_DT_STRUCT_MBRINFO • ESS_DT_STRUCT_TIGENINFO
count;	ESS_ULONG_T	Number of structures
structPtr;	ESS_PVOID_T	Pointer to memory

Notes

Always call this function, **EssOtlFreeStructure**(), after you call **EssOtlGetMemberInfo**().

Example

```
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

```
ESS_FUNC_M EssOtlGenerateCurrencyOutline (hOutline, phCurOutline);
```

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Outline context handle.
phCurOutline	ESS_PHOUTLINE_T	Pointer to an outline context handle for the return of the currency outline.

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

```
#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);
```

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Outline context handle (input).
hMember	ESS_HMEMBER_T	Member handle (input).
pAgglevelUsage	ESS_PSHORT_T	One of the Level Usage Constants listed in EssOtlSetAggLevelUsage documentation (output).

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;
```

```
  /* code to assign hOutline variable omitted */
  /* code to assign hMember variable omitted */

  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_HIERARCHY_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_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

- EssOtlClearAliasTableLanguages
- EssOtlSetAliasTableLanguage

EssOtlGetAltHierarchyEnabled

Returns the dimension's multiple-hierarchy-enabled setting.

Syntax

```
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.

See Also

- EssOtlSetAltHierarchyEnabled
- EssOtlGetHierarchyType
- EssOtlSetHierarchyType

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

```
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

- EssOtlSetASOCompressionDimension

EssOtlGetAssociatedAttributes

Returns all attribute members that are associated with a base member or dimension.

Syntax

```
ESS_FUNC_M EssOtlGetAssociatedAttributes(hOutline,hMember,
pusCount,pphMemberArray);
```

Parameter	Data Type	Description
hOutline;	ESS_HOUTLINE_T	Handle to the outline
hMember;	ESS_HMEMBER_T	Handle to the base member or base dimension
pusCount;	ESS_PUSHORT_T	Number of attribute members returned
pphMemberArray;	ESS_PPHMEMBER_T	Pointer to an array of attribute member handles

Example

```
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);
```

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Outline context handle (input).
hDimMember	ESS_HMEMBER_T	Attribute or linked-attribute dimension member handle (input).
psLevel	ESS_PUSHORT_T	The attribute association level (output).

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

```
#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)
{
```

```
   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

Parameter	Data Type	Description
hOutline;	ESS_HOUTLINE_T	Handle to the outline
hAttribute;	ESS_HMEMBER_T	Handle to the attribute member or dimension
pAttributeInfo;	"ESS_ATTRIBUTEINFO_T" on page 118	Attribute information

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

```
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;
      }
   }
}
```

See Also

- EssCheckAttributes
- EssFreeStructure
- EssGetAssociatedAttributesInfo
- EssGetAttributeInfo
- EssGetAttributeSpecifications
- EssOtlAssociateAttributeDimension
- EssOtlAssociateAttributeMember
- EssOtlDisassociateAttributeDimension
- EssOtlDisassociateAttributeMember
- EssOtlFindAttributeMembers
- EssOtlFreeStructure
- EssOtlGetAssociatedAttributes
- EssOtlGetAttributeSpecifications
- EssOtlQueryAttributes
- EssOtlSetAttributeSpecifications

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

```
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");
      break;
   case ESS_GENNAMEBY_SUFFIX:
      printf("\n Prefix/Suffix   : Suffix");
      break;
   default:
      printf("\n Prefix/Suffix   : None");
      break;
}
switch(AttrSpecs->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(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

```
#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,
```

```
        &hMemberChild);
}
```

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

```
ESS_FUNC_M EssOtlGetDimensionNameUniqueness (hOutline, hDim, *pulDupCount);
```

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Outline context handle (input).
hDim	ESS_HMEMBER_T	Input dimension, returned by EssOtlQueryGetFirstDimension() or EssOtlQueryGetNextDimension().
*pulDupCount	ESS_ULONG_T	The number of members with duplicate names (output).

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

```
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

```
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

```
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);

    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("EssOtlGetSMDateFormatString 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

- EssOtlGetServerDateFormats
- EssOtlSetDateFormatString

EssOtlGetDimensionNameUniqueness

Returns the dimension's member-name uniqueness setting.

Syntax

```
ESS_FUNC_M EssOtlGetDimensionNameUniqueness (hOutline, hDim, pbNameUnique);
```

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Outline context handle (input).
hDim	ESS_HMEMBER_T	Member handle of the dimension root member (input).
pbNameUnique	ESS_PBOOL_T	The dimension member name uniqueness setting (output). If TRUE, the dimension cannot have duplicate member names.

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

```
ESS_FUNC_M EssOtlGetDimensionSolveOrder (hOutline, hMember, pOrder);
```

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Outline context handle (input).
hMember	ESS_PHMEMBER_T	Dimension handle (input).
pOrder	ESS_PUCHAR_T	Solve order (output).

Notes

Solve order is applicable only to 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_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

```
ESS_FUNC_M EssOtlGetDimensionUserAttributes (hOutline, pPredicate,
                                             pCounts, ppAttributeNames);
```

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Essbase outline handle. This must have been returned from **EssOtlOpenOutlineQuery**().
pPredicate	"ESS_PREDICATE_T" on page 709	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 ESS_DIMUSERATTRIBUTES. • **pszDimension**—Dimension to limit the scope of the query. Specify a valid dimension name.
pCounts	"ESS_MBRCOUNTS_T" on page 699	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
ppAttributeNames	ESS_PPMBRNAME_T	An array of attribute names returned from the 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 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

```
#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*);

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	The Essbase outline handle returned from the EssOtlOpenOutline call.
pszDTSMember	ESS_STR_T	Name of the DTS member which provides the alias.

Parameter	Data Type	Description
pszAliasTable	ESS_STR_T	Name of the alias table which provides the alias. If NULL, the default alias table is used.
ppszAlias	ESS_PSTR_T	Pointer to a pointer to a C string containing the alias name for the DTS member.

Return Value

If successful the return value is zero. Otherwise, one of the following is returned:

- OTLAPI_ERR_DTSMBRNOTDEFINED
- OTLAPI_BAD_ALIASTABLE

Example

```
#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 = EssOtlGetDTSMemberAlias(hOutline, pszDTSMember, pszAliasTable, &pszAlias);
        if(sts)
        {
                printf("Could not get DTS member alias\n");
                return sts;
```

```
        }
        printf("MEMBER %s is aliased to %s\n", pszDTSMember, pszAlias);
        EssOtlCloseOutline(hOutline);
        return sts;
}
```

See Also

- EssOtlDeleteDTSMemberAlias
- EssOtlEnableDTSMember
- EssOtlGetEnabledDTSMembers
- EssOtlSetDTSMemberAlias

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 theESS_DTSMBRNAME_T structure passed to it.

Return Value

If successful the return value is zero. Otherwise, returns the status of the **EssOtlQueryMembers**() call.

Example

```
#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);
```

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Outline context handle.
phMember	ESS_PHMEMBER_T	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.

Return Value

Returns 0 if successful.

Example

```
#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

```
ESS_FUNC_M EssOtlGetGenName (hOutline, pszDimension, usGen, ppszName);
```

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Outline context handle.
pszDimension	ESS_STR_T	Name of dimension that contains the generation.
usGen	ESS_USHORT_T	Number of generation for which to get a name. The dimension is generation 1.
ppszName	ESS_PSTR_T	Buffer for return of generation name, allocated by the API.

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

```
#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);
```

```
/************ Get Gen Name ***************/
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*);

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Outline context handle.
pszDimension	ESS_STR_T	Name of dimension that contains the generation.
usGen	ESS_USHORT_T	Number of generation for which to get a name. The dimension is generation 1.
ppszName	ESS_PSTR_T	Buffer for return of generation name, allocated by the API.
pbNameUnique	ESS_PBOOL_T	Member name uniqueness setting.

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

```
ESS_FUNC_M EssOtlGetGenNames (hOutline, pszDimension, ulOptions, pulCount, pNameArray);
```

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Essbase outline handle.
pszDimension	ESS_STR_T	The dimension to retrieve generation names for.
ulOptions	ESS_ULONG_T	This can be one of the following values: • ESS_GENLEV_ALL—Returns default and actual generation names. • ESS_GENLEV_ACTUAL—Returns only generation names that are actually defined. • ESS_GENLEV_DEFAULT—Returns all default generation names. This includes the default names for generations that have an actual name. • ESS_GENLEV_NOACTUAL—Returnsdefault generation names. This includes only the generations that don't have an actual generation name.
pulCount	ESS_PULONG_T	Return of the number of elements in the pNameArray. It is the number of generation names for the specified member.
pNameArray	"ESS_GENLEVELNAME_T" on page 698	An array of generation name structures for the specified dimension.

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

```
#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         GenOpt;
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";
   GenOpt = ESS_GENLEV_ALL;

   if (!sts)
   {
      sts = EssOtlGetGenNames(hOutline, Dimension,
            GenOpt, &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
- EssOtlGetLevelName
- EssOtlGetLevelNames
- EssOtlOpenOutline
- EssOtlOpenOutlineQuery

EssOtlGetHierarchyType

Gets the dimension's hierarchy type designation: Multiple hierarchies enabled, dynamic hierarchy, or stored hierarchy.

Syntax

```
ESS_FUNC_M EssOtlGetHierarchyType(hOutline, hMember, pType);
```

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Outline context handle (input).
hMember	ESS_HMEMBER_T	A dimension member (input).
pType	ESS_UCHAR_T	If *hMember* is a dimension member, one of the following values (output): • ESS_STORED_HIERARCHY—The dimension is a single, stored hierarchy. • ESS_DYNAMIC_HIERARCHY—The dimension is a single, dynamic hierarchy. • ESS_MULTIPLE_HIERARCHY_IS_ENABLED—The dimension is multiple-hierarchy enabled. 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);
```

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Outline context handle
&impliedShareSetting	ESS_USHORT	Address of an implied share setting.

Return Value

- 0—If successful
- Error number—If unsuccessful

The implied share setting value. See "Implied Share Setting (C)" on page 103.

Return Parameter

ESS_USHORT impliedShareSetting

See Also

- EssOtlSetImpliedShare

EssOtlGetLevelName

Gets the name for a specific level within a dimension.

Syntax

```
ESS_FUNC_M EssOtlGetLevelName (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 generation.
usLevel	ESS_USHORT_T	Number of level number for which to get a name. Leaf members are level 0.
pszName	ESS_PSTR_T	Buffer for return of the level of the specified dimension, allocated by the API.

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

- OTLAPI_ ERR_GENLEVELNAMEMBR

Example

```
#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	Data Type	Description
hOutline	ESS_HOUTLINE_T	Outline context handle.
pszDimension	ESS_STR_T	Name of dimension that contains the generation.
usLevel	ESS_USHORT_T	Number of level number for which to get a name. Leaf members are level 0.
pszName	ESS_PSTR_T	Buffer for return of the level of the specified dimension, allocated by the API (output).
pbNameUnique	ESS_PBOOL_T	The member-name uniqueness setting (output).

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

```
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

- 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
ulOptions	ESS_ULONG_T	This can be one of the following values: • ESS_GENLEV_ALL—Returns default and actual level names • ESS_GENLEV_ACTUAL—Returns only level names that are actually defined • ESS_GENLEV_DEFAULT—Returns all default level names. This includes the default names for levels that have an actual name. • ESS_GENLEV_NOACTUAL—Returns default level names. This includes only the levels that don't have an actual level name.
pulCount	ESS_PULONG_T	Return of the number of elements in the pNameArray. It is the number of level names for the specified member.
pNameArray	"ESS_GENLEVELNAME_T" on page 698	An array of level name structures for the specified dimension.

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

```
#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

```
ESS_FUNC_M EssOtlGetLinkedAttributeAttachLevel (hOutline, hDimMember, psLevel);
```

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Outline context handle (input).
hDimMember	ESS_HMEMBER_T	Linked attribute dimension member handle (input).
psLevel	ESS_PUSHORT_T	The linked attribute attachment level (output).

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

```
#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

```
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

```
#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
hOutline	ESS_HOUTLINE_T	Outline context handle
hMember	ESS_HMEMBER_T	Member handle.
pszCommentEx	ESS_PSTR_T	Variable for the return of the extended comment. This buffer is allocated by the API.

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          sts = 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);
```

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Essbase outline handle. This must have been returned from **EssOtlOpenOutlineQuery**().
hMember	ESS_HMEMBER_T	Member handle returned by **EssOtlQueryMembersEx**().
MbrFieldID	ESS_ULONG_T	A member field identifier constant. See Notes.
ppFieldElement	ESS_PPVOID_T	Returned pointer to the required field element.

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_NEXTNAME
  - 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	Data Type	Description
hOutline	ESS_HOUTLINE_T	Outline context handle
hMember	ESS_HMEMBER_T	Member handle.
ppszFormula	ESS_PSTR_T	Variable for the return of the member formula. This buffer is allocated by the API.

Notes

- Use **EssFree**() to free the formula buffer.

Return Value

Returns 0 if successful.

Example

```
#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)
```

```
{
   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

```
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

```
#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

```
ESS_FUNC_M EssOtlGetMemberInfoArray (hOutline, memberCount, hMemberArr, pInfoArr,
pStsArr);
```

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Outline handle.
memberCount	ESS_SHORT_T	Count of members in the input array.
hMemberArr	ESS_HMEMBER_T	Array of memberCount member handles.
pInfoArr	ESS_PPMBRINFO_T	Array of memberCount member information pointers.

Parameter	Data Type	Description
pStsArr	ESS_STS_T	Array of memberCount status return codes. If any errors occur, the function returns the value of the first error encountered.

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

```
#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);
```

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Outline context handle
hMember	ESS_HMEMBER_T	Member handle.
ppszFormula	ESS_PSTR_T	Variable for the return of the member formula. This buffer is allocated by the API.

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

```
#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

```
ESS_FUNC_M EssOtlGetMemberSmartList(hOutline, hMember, *phSmartlist);
```

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Outline handle (Edit mode only)
hMember	ESS_HMEMBER_T	Outline member handle
*phSmartlist	ESS_HSMARTLIST_T	Returns the associate Text List (SmartList) handle

Return Value

Returns:

- 0—If successful

  **phSmartlist* contains return value.

- Error number—If unsuccessful

  **phSmartlist* is NULL.

Example

```
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

- EssOtlFreeSmartListInfo
- EssOtlGetMemberSmartList
- EssOtlGetMemberType
- EssOtlGetObjectReferenceCount
- EssOtlGetObjectReferences
- EssOtlImportExportObject
- EssOtlListObjects
- EssOtlQueryObjects
- EssOtlSetMemberType
- EssOtlSetMemberTypeToSmartList

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 (ouput).

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

```
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

```
ESS_FUNC_M EssOtlGetMemberType(hOutline, hMember, *pusType)
```

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Outline handle
hMember	ESS_HMEMBER_T	Outline member handle
*pusType	ESS_USHORT_T	Type of the outline member: • ESS_MEMBERTYPE_NUMERIC Member type is a numeric. • ESS_MEMBERTYPE_SMARTLIST Member type is textual and has an associated Text List (SmartList) object. • ESS_MEMBERTYPE_DATE Member type is date typed.

Return Value

Returns:

- 0—If successful

  pusType contains a value.
- Error number—If unsuccessful

  pusType is NULL.

Example

```
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 */

    /* 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

```
ESS_FUNC_M EssOtlGetMemberUniqueName (hOutline, hMember, *szFullName);
```

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Outline context handle (input).
hMember	ESS_HMEMBER_T	Member handle (input).
*szFullName	ESS_STR_T	The returned member name or qualified member name (output).

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]`

```
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);
 }
```

```
 return sts;
}
```

Example 2

The following example shows this function used in query mode.

```
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*);

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Outline context handle.
hMember	ESS_HMEMBER_T	Handle of member to find the next shared member for.
phMember	ESS_PHMEMBER_T	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.

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

```
#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

```
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

```
#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

```
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

```
ESS_FUNC_M EssOtlGetObjectReferenceCount(hOutline, objHandle, pCount)
```

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Outline handle (Edit mode only)
objHandle	ESS_HOBJECT_T	Object handle to be imported or exported
pCount	ESS_ULONG_T*	Count of outline members

Return Value

Returns:

- 0—If successful

  pCount contains values.

- Error number—If unsuccessful

  pCount is NULL.

Example

```
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;

    sts = EssOtlOpenOutline(hCtx, &Object,
                ESS_TRUE, ESS_TRUE, &hOutline);

    /* Get count of an object that is referenced */
    objType = OBJECT_SMARTLIST;
    objName = "Smartlist1";
    sts = EssOtlFindObject(hOutline, objType,
                objName, &hObjHandle);
    printf("EssOtlFindObject sts: %ld\n",sts);

    sts = 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 deallocing *phMembers* using EssFree.

Syntax

ESS_FUNC_M EssOtlGetObjectReferences(*hOutline, objHandle, ulMaxCount, phMembers, pulNumMembers*)

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Outline handle (Edit mode only)
objHandle	ESS_HOBJECT_T	Object handle
ulMaxCount	ESS_ULONG_T	Count of max outline members that the client can handle
phMembers	ESS_HMEMBER_T*	Output array of outline members
pulNumMembers	ESS_ULONG_T*	Number of outline members returned.

Return Value

Returns:

- 0—If successful

  ulMaxCount, *phMembers*, and *pulNumMembers* contain values.

- Error number—If unsuccessful

  ulMaxCount, *phMembers*, and *pulNumMembers* are NULL.

Example

```
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("EssOtlGetObjectRefrences 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

- EssOtlDeleteObject
- EssOtlGetSmartListInfo
- EssOtlFindObject
- EssOtlFreeObjectArray
- EssOtlFreeSmartListInfo
- EssOtlGetMemberSmartList
- EssOtlGetMemberType
- EssOtlGetObjectReferenceCount
- EssOtlGetObjectReferences
- EssOtlImportExportObject
- EssOtlListObjects
- EssOtlQueryObjects
- EssOtlSetMemberType
- EssOtlSetMemberTypeToSmartList

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

```
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);
  }
```

```
 return sts;
}
```

See Also

- EssOtlSetOriginalMember

EssOtlGetOutlineInfo

Returns information about the outline file.

Syntax

```
ESS_FUNC_M EssOtlGetOutlineInfo (hOutline, ppInfo);
```

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Outline context handle.
ppInfo	"ESS_OUTLINEINFO_T" on page 706	Pointer to a pointer to a structure allocated by the API for storing outline information.

Notes

- Use EssFree() to free the information structure.

Return Value

Returns 0 if successful.

Example

```
#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;
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(pInfo)
{
 EssFree(hInst, pInfo);
}
```

See Also

- EssFree
- EssOtlSetOutlineInfo

EssOtlGetParent

Returns the parent of a member.

Syntax

```
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 *hMember* parameter.

Notes

- If there is no parent, **phMember* is set to ESS_NULL and the call returns 0. (The *hMember* is a dimension.)

Return Value

Returns 0 if successful.

Example

```
#include <essapi.h>
#include <essotl.h>

ESS_STS_T          sts = 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
hOutline	ESS_HOUTLINE_T	Outline context handle.
hMember	ESS_HMEMBER_T	Handle of member to retrieve the previous sibling of.
phMember	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>
```

```
ESS_STS_T           sts = 0;
ESS_HOUTLINE_T      hOutline;
ESS_HMEMBER_T       hMemberFeb;
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, "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

```
ESS_FUNC_M EssOtlGetQueryHint (hOutline, hintNum, numMembers, pMemberArray);
```

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Outline context handle (input).
hintNum	ESS_SHORT_T	Query hint number (input).
numMembers	ESS_SHORT_T	Number of members that the array provided is able to hold - usually the number of real dimensions in the outline (input)
pMemberArray	ESS_PHMEMBER_T	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)

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

```
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

```
ESS_FUNC_M EssOtlGetQueryHintSize (hOutline, pHintSize);
```

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Outline context handle (input).
pHintSize	ESS_SHORT_T	Query hint size (output).

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);
```

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Outline handle
hSmartlist	ESS_HSMARTLIST_T	Text List (SmartList) handle
**ppSmartListInfo	ESS_SMARTLISTINFO_T	Contains the Text List (SmartList) information structure.

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);
```

```
    if(SmartListInfo)
        sts = EssOtlFreeSmartListInfo(hOutline, SmartListInfo);

}
```

EssOtlGetServerDateFormats

This function returns the list of server date formats supported.

Syntax

```
ESS_FUNC_M EssOtlGetServerDateFormats(
    ESS_HCTX_T hCtx,
    ESS_STR_T localeStr,
    ESS_USHORT_T* pcount,
    ESS_STR_T** ppdateStrings,
    ESS_STR_T** ppformatStrings)
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	Server context handle
localeStr	ESS_STR_T	Locale in which the example date Strings to be generated. • If *localeStr* is empty, the default environment locale is used to generate thedate Strings • If *localeStr* is invalid, invalid error message is returned • If *localeStr* is null, error message is returned
pcount	ESS_USHORT_T*	Count of date formats supported
ppdateStrings	ESS_STR_T**	Returns the example current date in different date formats as an array (to be de allocated).
ppformatStrings	ESS_STR_T**	Returns the array of formats supported (to be de allocated).

Return Value

Returns:

- 0—If successful

  Values are contained in *ppdateStrings* and *ppformatStrings*.

- Error number—If unsuccessful

Example

```
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);

    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("EssOtlGetSMDateFormatString 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

Parameter	Data Type	Description
hOutline;	ESS_HOUTLINE_T	Outline handle
pOtlTimeStamp;	ESS_PTIME_T	Pointer to the timestamp for the outline

Notes

- The value for time, of type ESS_ULONG_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

```
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);
```

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Outline context handle.
hMember	ESS_HMEMBER_T	Handle of member for which to get the user-defined attribute.
pusCount	ESS_PUSHORT_T	Count of user attributes returned; defines the number of elements in the *ppAttributeList* array.
ppAttributeList	ESS_PPMBRNAME_T	Array of **pusCount* members. Each element of the array contains a single user-defined attribute string.

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

```
#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");
```

```
        for(ind = 0; ind < Count; ind++)
   {
   printf("%s\n",AttributeList[ind]);
   }
   EssFree(hInst, AttributeList);
}
```

See Also

- EssOtlDeleteUserAttribute
- EssOtlSetUserAttribute

EssOtlImportExportObject

Imports or exports the contents of the input object to the input file based on *bImport* being true or false.

Syntax

```
ESS_FUNC_M EssOtlImportExportObject(hOutline, objHandle, FileName, bImport)
```

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Outline handle (Edit mode only)
objHandle	ESS_HOBJECT_T	Object handle to be imported or exported
FileName	ESS_STR_T	Name of the file to which object needs to be exported or imported from
bImport	ESS_BOOL_T	• `true` Import • `false` Export

Return Value

Returns:

- 0—If successful
- Error number—If unsuccessful

Example

```
void TestImportExportObject()
{
    ESS_STS_T                sts = ESS_STS_NOERR;
    ESS_OBJDEF_T            Object;
    ESS_HOUTLINE_T         hOutline = ESS_NULL;
    ESS_HOBJECT_T         hObjHandle = ESS_NULL;
    ESS_PHOBJECT_T         hObjHandles;
    ESS_STR_T                 sFileName;
    ESS_BOOL_T             bImport;
    ESS_OBJECT_TYPES     objType;
    ESS_STR_T                 objName = "";
    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;

    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;
    sFileName = "F:\\testArea\\Smartlist\\ExpCSRatingsSL.txt";
    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

- EssOtlGetSmartListInfo
- EssOtlFindObject
- EssOtlFreeObjectArray
- EssOtlFreeSmartListInfo
- EssOtlGetMemberSmartList
- EssOtlGetMemberType
- EssOtlGetObjectReferenceCount
- EssOtlGetObjectReferences
- EssOtlImportExportObject
- EssOtlListObjects
- EssOtlQueryObjects
- EssOtlSetMemberType
- EssOtlSetMemberTypeToSmartList

EssOtlIsMemberNameNonUnique

Discovers if a member name is duplicate.

Syntax

```
ESS_FUNC_M EssOtlIsMemberNameNonUnique (hOutline, hMember, fNameNonUnique);
```

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Outline context handle (input).
hMember	ESS_HMEMBER_T	The member to query for non-uniqueness (input).
*fNameNonUnique	ESS_BOOL_T	TRUE if the member queried is a duplicate member name (output).

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

- EssOtlIsMemberNameUniqueWithinDim
- EssOtlIsMemberNameUniqueWithinDimAtGenLevel

EssOtlIsMemberNameUniqueWithinDim

Discovers if member names are all unique within a dimension.

Syntax

```
ESS_FUNC_M EssOtlIsMemberNameUniqueWithinDim (hOutline, hDim, *pbNameUnique);
```

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Outline context handle (input).
hDim	ESS_HMEMBER_T	Input dimension, returned by EssOtlQueryGetFirstDimension() or EssOtlQueryGetNextDimension().
*pbNameUnique	ESS_BOOL_T	TRUE if the dimension queried contains no duplicate member names; FALSE otherwise.

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);
```

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Outline context handle.
hDim	ESS_HMEMBER_T	Input dimension, returned by EssOtlQueryGetFirstDimension() or EssOtlQueryGetNextDimension().
bGen	ESS_BOOL_T	Input. If TRUE, *usGenLevel* is considered a generation number. If FALSE, *usGenLevel* is considered a level number.

Parameter	Data Type	Description
usGenLevel	ESS_USHORT_T	Input generation or level number.
*pbNameUnique	ESS_BOOL_T	Output. TRUE if the dimension queried contains duplicate member names at the generation or level specified; FALSE otherwise.

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_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)
```

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Outline handle (Edit mode only)
objType	ESS_OBJECT_TYPES	Object type with one of the following values: • OBJECT_SMARTLIST Object type is Text List (SmartList)
pCount	ESS_ULONG_T*	Count of object handles
pObjHandles	ESS_PPHOBJECT_T	Returns an array of object handles. Must be deallocated using EssFree.

Return Value

Returns:

- 0—If successful

  pCount and *pObjHandles* contain values.

- Error number—If unsuccessful

  pCount and *pObjHandles* are NULL.

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;
    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

```
ESS_FUNC_M EssOtlMoveMember (hOutline, hMember,  hNewParent, hNewPrevSibling);
```

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Outline context handle
hMember	ESS_HMEMBER_T	Handle of member to move
hNewParent	ESS_HMEMBER_T	Handle of new parent. Use this field only if the *hNewPrevSibling* field is ESS_NULL.
hNewPrevSibling	ESS_HMEMBER_T	Handle of new previous sibling

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_ATTRMBRDT_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

```
#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);
```

```
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

```
ESS_FUNC_M EssOtlNewOutline (hCtx, pNewInfo, phOutline);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	Essbase Context handle.
pNewInfo	"ESS_OUTLINEINFO_T" on page 706	Structure describing the new outline.
phOutline	ESS_PHOUTLINE_T	Pointer to ESS_HOUTLINE_T variable. This handle is set by the API and should be passed in to subsequent Outline API functions.

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.

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);
```

See Also

- EssOtlOpenOutline
- EssOtlWriteOutline
- EssOtlRestructure
- EssOtlCloseOutline
- EssOtlVerifyOutline

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

```
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

```
#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

```
ESS_FUNC_M EssOtlOpenOutlineEx(hCtx, pObject, fLock, fKeepTrans, pLocaleDescription,
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.
pLocaleDescription		The identifier used by GlobalC to identify the Locale. The LocaleDescription is in the form of: `[language]_[territory].[codepage]@[sort]` For example: `Japaness_japan.MS932@binary` 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.
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

- 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

- 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.

See Also

- EssOtlCloseOutline
- EssOtlGetMemberCommentEx
- EssOtlNewOutline
- EssOtlRestructure
- EssOtlSetMemberCommentEx
- EssOtlVerifyOutline
- EssOtlWriteOutlineEx

EssOtlOpenOutlineQuery

Opens an existing outline.

Syntax

```
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, `#OUTOFRANGE`, or the same as `pszMissingName` or `pszOutofRangeName`
- The number of entries len cannot be more than 1024.

Syntax

```
ESS_FUNC_M EssOtlPutSmartList(hOutline, hSmartList, len, *pIDs, *ppszText,
pszMissingName, pszOutofRangeName);
```

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	The source Essbase outline for the text list.
hSmartList	ESS_HSMARTLIST_T	Text list handle
len	ESS_UINT16	Number of items
*pIDs	ESS_UINT32_T	Integer IDs
*ppszText	ESS_STR_T	Enumerated text
pszMissingName	ESS_STR_T	Name of the missing smart text
pszOutofRangeName	ESS_STR_T	Name of the out of range smart text.

Return Value

Returns:

- 0—If successful

  pIDs and *ppszText* contain values.

- Error number—If unsuccessful

  pIDs and *ppszText* are NULL.

Example

```
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;
```

```
ESS_HSMARTLIST_T            hSmartList;
ESS_USHORT_T                    len;
ESS_SMARTLISTID_T           pIds[4];
ESS_STR_T                       ppszText[4];
ESS_STR_T                       pszMissingName;
ESS_STR_T                       pszOutOfRangeName;
ESS_ULONG_T                   Count, i;
ESS_STR_T                       smartListNames[3] =
        { "MainColors", "TempColors1", "TempColors2" };

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 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

Parameter	Data Type	Description
hOutline;	ESS_HOUTLINE_T	Handle to the outline
pAttributeQuery;	"ESS_ATTRIBUTEQUERY_T" on page 697	Structure that defines the query
pCount;	ESS_PULONG_T	Number of member handles returned
pphMemberArray;	ESS_PPHMEMBER_T	Pointer to an array of member handles returned

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_PHMEMBER_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)
    {
       printf("\n------- Query Results -------\n");
       for (index = 0; index < Count; index++)
       {
          sts = EssOtlGetMemberInfo(hOutlineQuery,phMemberArray[index],&pMbrInfo);
          printf("\t%s\n",pMbrInfo->szMember);
       }

       if (Count && phMemberArray)
       {
          sts = EssOtlFreeMembers(hOutlineQuery,Count, phMemberArray);
       }
    }
}
```

See Also

- EssCheckAttributes
- EssFreeStructure
- EssGetAssociatedAttributesInfo
- EssGetAttributeInfo
- EssGetAttributeSpecifications
- EssOtlAssociateAttributeDimension
- EssOtlAssociateAttributeMember
- EssOtlDisassociateAttributeDimension
- EssOtlDisassociateAttributeMember
- EssOtlFindAttributeMembers
- EssOtlFreeStructure
- EssOtlGetAssociatedAttributes
- EssOtlGetAttributeInfo
- EssOtlGetAttributeSpecifications
- EssOtlSetAttributeSpecifications

EssOtlQueryAttributesEx

Queries member information for a given attribute member or dimension.

Syntax

Parameter	Data Type	Description
hOutline;	ESS_HOUTLINE_T	Handle to the outline
pAttributeQuery;	"ESS_ATTRIBUTEQUERY_T" on page 697	Structure that defines the query
pCount;	ESS_PMBRCOUNTS_T	Number of member handles returned
pphMemberArray;	ESS_PPHMEMBER_T	Pointer to an array of member handles returned

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);
```

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Outline handle. This must have been returned from **EssOtlOpenOutlineQuery**().
szName	ESS_STS_T	Name of the top member of the date-time dimension
queryID	ESS_ULONG_T	Use the query identifier constant `ESS_OTLQRYTIDIM_TIMEPERIODS`
ppReturns	ESS_PPVOID_T	A pointer to the query information structure for this dimension.

Notes

The caller of **EssOtlQueryGenerationInfo**() should call **EssOtlFreeStructure**() with structure ID `ESS_DT_STRUCT_TIGENINFO` 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

```
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_TIGENINFO,
          1, pReturns);
       if (sts)
         goto exit;
       break;
```

```
    default:
      break;
  }
```

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

```
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

Parameter	Data Type	Description
hOutline;	ESS_HOUTLINE_T	Handle to the outline (input)
hDim;	ESS_HMEMBER_T	The dimension handle (input)
phNextDim;	ESS_PHMEMBER_T	The handle of the next dimension (output)

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

```
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

```
ESS_FUNC_M EssOtlQueryMembers (hOutline, hMember, pPredicate, pMbrCounts, phMemberArray);
```

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Essbase outline handle. This must have been returned from EssOtlOpenOutlineQuery.

Parameter	Data Type	Description
hMember	ESS_HMEMBER_T	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: • ESS_NAMEDGENERATION • ESS_NAMEDLEVEL • ESS_USERATTRIBUTE • ESS_SEARCH • ESS_WILDSEARCH
pPredicate	"ESS_PREDICATE_T" on page 709	Structure defining the query. The fields of this structure are described in Notes.
pMbrCounts	"ESS_MBRCOUNTS_T" on page 699	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	ESS_PPHMEMBER_T	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 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

```
#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)
```

```
    {
      sts = EssOtlQueryMembers(hOutline, hMember,
            &Predicate, &Counts, &phMemberArray);

      if (!sts && Counts.ulReturnCount)
      {
         sts = EssOtlFreeMembers(hOutline,
               Counts.ulReturnCount, phMemberArray);
      }
    }
}
```

See Also

- EssOtlFreeMembers
- EssOtlGetDimensionUserAttributes
- EssOtlOpenOutlineQuery
- EssOtlQueryMembersByName

EssOtlQueryMembersByName

Queries the outline.

Syntax

```
ESS_FUNC_M EssOtlQueryMembersByName (hOutline, pszMember, pPredicate,
                                     pMbrCounts, phMemberArray);
```

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Outline context handle. This must have been returned from **EssOtlOpenOutlineQuery**().
pszMember	ESS_STR_T	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: • ESS_NAMEDGENERATION • ESS_NAMEDLEVEL • ESS_USERATTRIBUTE • ESS_SEARCH • ESS_WILDSEARCH
pPredicate	"ESS_PREDICATE_T" on page 709	Structure defining the query. The fields of this structure are described in Notes.

Parameter	Data Type	Description
pMbrCounts	"ESS_MBRCOUNTS_T" on page 699	Structure defining information about member 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	ESS_PPHMEMBER_T	An array of member handles returned from the 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 **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_DTSMEMBERS
  - **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:

```
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
  - 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

```
#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;
```

```
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;

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)
```

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Essbase outline handle. This must have been returned from **EssOtlOpenOutlineQuery**().
pszFieldSelection	ESS_STR_T	The query string which defines the set of fields that will be returned for each member. The syntax of pszFieldSelection is shown in Notes.
pszMemberSelection	ESS_STR_T	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.
pMaxCount	ESS_PULONG_T	Input: A pointer to the maximum number of member handles to be returned. Output: A pointer to the number of member handles returned.
ppMemberArray	ESS_PPHMEMBER_T	Reference to a pointer to the first in an array of member handles returned.
ppqryErrorList	"ESS_OTLQUERYERRORLIST_T" on page 703	Reference to a pointer to a structure containing the list of errors in the query.

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 menmber 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)
```

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Outline handle returned from **EssOtlOpenOutlineQuery**().
pszFieldSelection	ESS_STR_T	Selects the member fields that the queries return. The same selections are used for all queries in the array.
queryCount	ESS_SHORT_T	Count of members in the input array.
pszMemberSelectionArr	ESS_STR_T	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.
pMaxCountArr	ESS_PULONG_T	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.

Parameter	Data Type	Description
pphMemberArr	ESS_PPHMEMBER_T	queryCount array of returned member handle arrays (each with pMaxCountArr[i] values).
ppqryErrorList	"ESS_OTLQUERYERRORLIST_T" on page 703	List of members with errors.

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 */
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.

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 EssOtlQueryMembersEx into just one call to EssOtlQueryMembersExArray. 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]*.

```
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 = EssOtlQueryMembersExArray(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 **EssOtlQueryMembersEx**(), **EssOtlGetMemberField**(), and **ESS_OTLQUERYERRORLIST_T**, and includes calls to **EssOtlFreeMembers**() and **EssFree**().

See Also

- EssOtlFreeMembers
- EssOtlGetDimensionUserAttributes

- 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)
```

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Outline handle (Query mode only)
objType	ESS_OBJECT_TYPES	Object type: • OBJECT_SMARTLIST Object type is Text List (SmartList)
objNames	ESS_PSTR_T	Array of object names to be queried
pcount	ESS_PULONG_T	Count of object names. If *pcount* is zero, this contains the number of Text List (SmartList) handles on execution.
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             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
- EssOtlListObjects
- EssOtlQueryObjects
- EssOtlSetMemberType
- EssOtlSetMemberTypeToSmartList

EssOtlQueryVaryingAttributes

Queries member information for a given attribute member or function, enabling specification of the perspective for varying attributes.

Syntax

```
ESS_FUNC_M EssOtlQueryVaryingAttributes (hOutline, pAttrQuery, pPerspective,
pCount, pphMembers);
```

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Outline context handle.
pAttrQuery	ESS_PATTRIBUTEQUERY_T	Pointer to the structure that defines the query. • If `pAttrQuery.bInputMemberIsHandle = ESS_TRUE`, make sure `pAttrQuery.uInputMember.hMember` is assigned a handle to a member. • If `pAttrQuery.bInputMemberIsHandle = ESS_FALSE`, make sure `pAttrQuery.uInputMember.szMember` is assigned a member name.
pPerspective	ESS_PPERSPECTIVE_T	Pointer to a collection of independent members used when querying the client or server for associations.
pCount	ESS_PMBRCOUNTS_T	Pointer to the number of base members returned.
pphMembers	ESS_PPHMEMBER_T	Pointer to the array of attribute member handles.

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

```
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;
```

```
    ESS_PMBRINFO_T    pMbrInfo = ESS_NULL;
    ESS_VARYING_ATTRIBUTEQUERY_T    pAttrQuery;
    ESS_MBRCOUNTS_T    Counts;
    ESS_HMEMBER_T    hIndepMbrHandlesArray[4];
    ESS_PERSPECTIVE_T Perspective;
    ESS_PHMEMBER_T    phMbrHandles;
    ESS_HMEMBER_T    hAttrMbr;
    ESS_HMEMBER_T    hAttrDim;
    ESS_PREDICATE_T Predicate;

    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("\n*** 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_ATTRMBRDT_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

```
#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,
```

```
          "Alias Table 2","2nd alias table");
}
```

See Also

- EssOtlCreateAliasTable
- EssOtlCopyAliasTable
- EssOtlClearAliasTable
- EssOtlDeleteAliasTable
- EssOtlSetAliasTableLanguage

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

```
#include <essapi.h>
#include <essotl.h>

ESS_STS_T          sts = 0;
```

```
ESS_OBJDEF_T       Object;
ESS_HOUTLINE_T     hOutline;
ESS_HMEMBER_T      hMemberJan;
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 = 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);
```

Parameter	Data Type	Description
hCtx	ESS_HCTX_T	Server login context handle. This must be the server on which the outline was saved using **EssOtlWriteOutline**().

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

```
#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);
```

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Outline context handle (input) Value
hMember	ESS_HMEMBER_T	A hierarchy member (input). 0

Parameter	Data Type	Description
sAgglevelUsage	ESS_SHORT_T	One of the Level Usage Constants (input).

Constant	Value	Description
ESS_AGGLEVELUSAGE_DEFAULT	11	On primary hierarchies, consider all levels. Do not aggregate secondary hierarchies unless alternate rollups are enabled.
ESS_AGGLEVELUSAGE_ALL	12	Consider all levels for aggregation. This is same as default for primary hierarchies, but not for secondary hierarchies.
ESS_AGGLEVELUSAGE_NOAGGREGATION	13	Do not aggregate along this hierarchy. All views selected are at the input level.
ESS_AGGLEVELUSAGE_BOTTOMONLY	14	Applies only to secondary hierarchies. Consider only lowest level of this hierarchy for aggregation.
ESS_ AGGLEVELUSAGE_TOPONLY	15	Applies only to primary hierarchies. Consider only topmost level of this hierarchy for aggregation.
ESS_ AGGLEVELUSAGE_BOTTOMTOP	16	Applies to primary hierarchies. Select top and bottom levels only.

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

```
   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
```

```
{
    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);
```

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Outline context handle.
pszAliasTable	ESS_STR_T	Name of the alias table for which to set a language code.
pszLanguageCode	ESS_STR_T	A language code to assign to the alias table specified in *pszAliasTable*. The language code should be a middle-tier language tag from an ApplCore session. Language codes are not case-sensitive.

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

```
#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
- EssOtlClearAliasTableLanguages

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

```
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_ATTRRENAMENAMEUSED error is returned.

Example

```
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:\n\t\t0. ESS_GENNAMEBY_PREFIX\n\t\t1. ESS_GENNAMEBY_SUFFIX
\n\nChoice: ");
```

```
  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\t\t0.  ESS_USENAMEOF_NONE\n\t\t1.
ESS_USENAMEOF_PARENT");
  printf("\n\t\t2.  ESS_USENAMEOF_GRANDPARENTANDPARENT\n\t\t3.
ESS_USENAMEOF_ALLANCESTORS");
  printf("\n\t\t4.  ESS_USENAMEOF_DIMENSION\n\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\t\t0.  ESS_DELIMITER_UNDERSCORE\n\t\t1.
ESS_DELIMITER_PIPE");
  printf("\n\t\t2.  ESS_DELIMITER_CARET\n\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:\n\t\t0.  ESS_DATEFORMAT_MMDDYYYY\n\t\t1.
 ESS_DATEFORMAT_DDMMYYYY\n\nChoice: ");
   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("\n\nInvalid choice.\n\n");
    }

    printf("Enter BucketingType:\n\t\t0.  ESS_UPPERBOUNDINCLUSIVE\n\t\t1.
 ESS_LOWERBOUNDINCLUSIVE");
   printf("\n\t\t2.  ESS_UPPERBOUNDNONINCLUSIVE\n\t\t3.  ESS_LOWERBOUNDNONINCLUSIVE\n
\nChoice:  ");
   test = atoi(gets(buffer[0]));
   switch (test)
    {
      case 0:
         AttrSpecs.usBucketingType=ESS_UPPERBOUNDINCLUSIVE;
         break;
      case 1:
         AttrSpecs.usBucketingType=ESS_LOWERBOUNDINCLUSIVE;
         break;
      default:
         printf("\n\nInvalid choice.\n\n");
    }

   printf("\nEnter 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("\nEnter 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("\nEnter your default attribute calculation dimension name (or
'ESS_DEFAULT_ATTRIBUTECALCULATIONS'):\n");
   gets(buffer[2]);
   if (buffer[2] == "ESS_DEFAULT_ATTRIBUTECALULATIONS")
      AttrSpecs.pszDefaultAttrCalcDimName="";
   else
      AttrSpecs.pszDefaultAttrCalcDimName=buffer[2];
```

```
    printf("\nEnter 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("\nEnter 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("\nEnter 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("\nEnter 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("\nEnter 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

- EssFreeStructure
- EssGetAssociatedAttributesInfo
- EssGetAttributeInfo
- EssGetAttributeSpecifications
- EssOtlAssociateAttributeDimension
- EssOtlAssociateAttributeMember
- EssOtlDisassociateAttributeDimension
- EssOtlDisassociateAttributeMember
- EssOtlFindAttributeMembers
- EssOtlFreeStructure
- EssOtlGetAssociatedAttributes
- EssOtlGetAttributeInfo
- EssOtlGetAttributeSpecifications
- EssOtlQueryAttributes

EssOtlSetDateFormatString

This function sets the outline property date format String.

Syntax

```
ESS_FUNC_M EssOtlSetDateFormatString(
    ESS_HOUTLINE_T hOutline,
    ESS_STR_T    formatString)
```

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Outline for the Smartlist.
formatString	ESS_STR_T	Returns the outline date format string to this argument.

Return Value

Returns:

- 0—If successful

  formatString constains a date format string.

- Error number—If unsuccessful

Example

```
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);

    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("EssOtlGetSMDateFormatString 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

- EssOtlGetServerDateFormats
- EssOtlGetDateFormatString

EssOtlSetDimensionNameUniqueness

Sets the dimension to prohibit duplicate (non-unique) member names.

Syntax

```
ESS_FUNC_M EssOtlSetDimensionNameUniqueness (hOutline, hMember, bNameUnique);
```

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Outline context handle (input).
hMember	ESS_HMEMBER_T	Member handle of the dimension root member (input).
bNameUnique	ESS_BOOL_T	The dimension member-name uniqueness setting (input). If set to TRUE, then the dimension cannot have duplicate member names.

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

- EssOtlGetDimensionNameUniqueness

EssOtlSetDimensionSolveOrder

Sets the solve order of a dimension.

Syntax

```
ESS_FUNC_M EssOtlSetDimensionSolveOrder (hOutline, hMember, cOrder);
```

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Outline context handle (input).
hMember	ESS_HMEMBER_T	Dimension handle (input).
cOrder	ESS_UCHAR_T	Solve order (input). 0 - 127

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

```
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 = 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

```
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.

Parameter	Data Type	Description
pszDTSMember	ESS_STR_T	Name of the DTS member which provides the alias.
pszAlias	ESS_STR_T	Pointer to a C string containing the alias name for the DTS member.
pzsAliasTable	ESS_STR_T	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

```
#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_ULONG_T     ulErrors;
        ESS_ULONG_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

EssOtlSetGenName

Sets the name for a specific generation within a dimension.

Syntax

```
ESS_FUNC_M EssOtlSetGenName (hOutline, pszDimension, usGen, pszName);
```

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Outline context handle.
pszDimension	ESS_STR_T	Name of dimension that contains the generation.
usGen	ESS_USHORT_T	Number of generation for which to set a name. The dimension itself is generation 1.
pszName	ESS_STR_T	Name to give the generation.

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_GENLEVELEXISTS
- OTLAPI_ERR_GENLEVELVALUE
- OTLAPI_ERR_NOTADIM
- OTLAPI_ ERR_GENLEVELNAMEMBR

Example

```
#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 = EssOtlSetGenName(hOutline, Dimension,
         GenNum, GenName);
}
```

See Also

- EssOtlDeleteGenName

EssOtlSetGenNameEx

Sets the generation name and member uniqueness setting for the specified generation number.

Syntax

```
ESS_FUNC_M EssOtlSetGenNameEx (hOutline, pszDimension, usGen, pszName, bUniqueName);
```

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Outline context handle.
pszDimension	ESS_STR_T	Dimension name.
usGen	ESS_USHORT_T	The number of the generation for which to set a name.
pszName	ESS_STR_T	The name to give the generation.
bUniqueName	ESS_BOOL_T	If TRUE, members at generation *usGen* in dimension *pszDimension* cannot have duplicate names.

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 EssOtlGetGenName 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

```
ESS_FUNC_M EssOtlSetHierarchyType(hOutline, hMember, cType);
```

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Outline context handle (input).
hMember	ESS_HMEMBER_T	A dimension member (input).
cType	ESS_UCHAR_T	If *hMember* is a dimension member, one of the following values (input): • ESS_STORED_HIERARCHY—The dimension is a single, stored hierarchy. • ESS_DYNAMIC_HIERARCHY—The dimension is a single, dynamic hierarchy. • ESS_MULTIPLE_HIERARCHY_IS_ENABLED—The dimension is multiple-hierarchy enabled (same as using EssOtlSetAltHierarchyEnabled). 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, *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.

See Also

- EssOtlGetHierarchyType
- EssOtlSetAltHierarchyEnabled
- EssOtlGetAltHierarchyEnabled

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_GENLEVELEXISTS
- OTLAPI_ERR_NOTADIM
- OTLAPI_ ERR_GENLEVELNAMEMBR

Example

```
#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

```
ESS_FUNC_M EssOtlSetLevelNameEx (hOutline, pszDimension, usLevel, pszName, bUniqueName);
```

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.
bUniqueName	ESS_BOOL_T	If TRUE, members at level *uslevel* in dimension *pszDimension* cannot have duplicate names. If FALSE, duplicate names are allowed.

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

```
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);
```

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Outline context handle.
hMember	ESS_HMEMBER_T	Handle of member to set the alias for.
pszAliasTable	ESS_STR_T	Alias table to set the alias for. If this parameter is ESS_NULL, the default alias table is used.
pszAlias	ESS_STR_T	Alias.

Notes

Use EssOtlDeleteMemberAlias() to remove an alias.

Return Value

Returns 0 if successful; otherwise one of the following::

- OTLAPI_BAD_ALIASTABLE
- OTLAPI_ERR_ILLEGALDEFALIAS
- OTLAPI_ERR_ILLEGALCOMBOALIAS
- OTLAPI_ERR_ILLEGALALIASSTRING
- OTLAPI_ERR_DUPLICATEALIAS

Example

```
#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

```
ESS_FUNC_M EssOtlSetMemberCommentEx (hOutline, hMember, pszCommentEx);
```

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Outline context handle
hMember	ESS_HMEMBER_T	Member handle.
pszCommentEx	ESS_STR_T	Buffer containing the extended comment.

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_MBRCOMMENTEXLEN if the comment is too long.

Example

```
#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          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;
```

```
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

```
ESS_FUNC_M EssOtlSetMemberFormula (hOutline, hMember, pszFormula);
```

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Outline context handle.
hMember	ESS_HMEMBER_T	Member handle.
pszFormula	ESS_STR_T	Buffer containing the member formula.

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

```
#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;

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);
```

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Outline context handle
hMember	ESS_HMEMBER_T	Handle to member to set attributes for
pInfo	"ESS_MBRINFO_T" on page 699	Member information structure

Notes

Attributes

- Three fields of the ESS_MBRINFO_T structure are for attributes only:

Data Type	Field	Description
ESS_BOOL_T	*fAttributed*	Indicates whether a base member has attributes associated with it: either ESS_TRUE or ESS_FALSE.
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.
union ESS_BOOL_T ESS_STR_T ESS_DATETIME_T ESS_DOUBLE_T	*Attribute.value* *bData* *strData* *dtData* *dblData*	One of the following attribute member values: ❍ Boolean value ❍ String value ❍ Date and time value ❍ Double value

- Values for two fields of the ESS_MBRINFO_T structure are for attributes only:

Data Type	Field	Description
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

- 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

```
#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)
```

```
{
   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);
```

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Outline context handle (input).
hMember	ESS_HMEMBER_T	Dimension handle (input).
cOrder	ESS_UCHAR_T	Solve order (input). 0 - 127

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

```
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
hOutline	ESS_HOUTLINE_T	Outline handle (Edit mode only)

Parameter	Data Type	Description
hMember	ESS_HMEMBER_T	Outline member handle
usType	ESS_USHORT_T	Type of the outline member: • ESS_MEMBERTYPE_NUMERIC Member type is a numeric. • ESS_MEMBERTYPE_DATE Member type is date typed.

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

```
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);
```

```
    printf("EssOtlSetMemberType sts: %d\n",sts);

    /* 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

- EssOtlFreeSmartListInfo
- EssOtlGetMemberSmartList
- EssOtlGetMemberType
- EssOtlGetObjectReferenceCount
- EssOtlGetObjectReferences
- EssOtlImportExportObject
- EssOtlListObjects
- EssOtlQueryObjects
- EssOtlSetMemberType
- EssOtlSetMemberTypeToSmartList

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	SmarList 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);
```

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Outline context handle (input).
hMember	ESS_HMEMBER_T	Member name (input). This member will be set as extended shared.
pszOriginalMbr	ESS_STR_T	The original member name intended to share with (input).

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 `EssOtlOpenOutline` 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

```
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;
```

```
 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

```
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 = EssOtlRestructure(hCtx, ESS_DOR_ALLDATA);
 }



 if (pInfo)
 {
  EssFree(hInst, pInfo);
 }

}
```

See Also

- EssOtlGetOutlineInfo

EssOtlSetQueryHint

Changes the contents (*pMemberArray*) of an existing query hint; applies only to Release 9.3 or higher aggregate storage databases.

Syntax

```
ESS_FUNC_M EssOtlSetQueryHint (hOutline, hintNum, numMembers, pMemberArray);
```

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Outline context handle (input).
hintNum	ESS_SHORT_T	Query hint number (input).
numMembers	ESS_SHORT_T	Number of members in the array provided - usually the number of real dimensions in the outline (input???)
pMemberArray	ESS_PHMEMBER_T	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)

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

```
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);
}
```

See Also

- EssOtlAddQueryHint
- EssOtlGetQueryHint
- EssOtlGetNumQueryHints
- EssOtlGetQueryHintSize
- EssOtlDeleteQueryHint

EssOtlSetUserAttribute

Sets a user-defined attribute for a member.

Syntax

```
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

```
#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          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

```
ESS_FUNC_M EssOtlSortChildren (hOutline, hParent, usType, fpCompare, pUserData);
```

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Outline context handle.
hParent	ESS_HMEMBER_T	Handle of parent of the children to sort. If this is ESS_NULL, the dimensions are sorted.
usType	ESS_USHORT_T	Sort type. This can be one of the following: • ESS_SORT_ASCENDING • ESS_SORT_DESCENDING • ESS_SORT_USERDEFINED

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:
		`(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

```
#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 && pMbrInfo1)
   sts = EssOtlGetMemberInfo(hOutline,
         hMember2, &pMbrInfo2);

   if (!sts && pMbrInfo2)
   {
     nLen1 = strlen(pMbrInfo1->szMember);
     nLen2 = strlen(pMbrInfo2->szMember);
     if (nLen1 < nLen2)
         nRet = -1;
     else if (nLen1 > 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

```
ESS_FUNC_M EssOtlVaryingAssociateAttribute (hOutline, hBaseMember, hAttrMember, mode,
pValiditySet);
```

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Outline context handle
hBaseMember	ESS_HMEMBER_T	Handle to the base member
hAttrMember	ESS_HMEMBER_T	Handle to the attribute member
mode	ESS_INT_T	Association mode. Possible values: • MODE_OVERWRITE • MODE_NOOVERWRITE
pValiditySet	"ESS_VALIDITYSET_T" on page 710	Pointer to the validity set that defines the independent dimension for which the association is valid

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):

  ```
  Jan   Feb   Mar   Apr   May
  12    12    16    16    20
  ```

  - 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:

    ```
    Jan   Feb   Mar    Apr    May
    12    12    12     12     12
    ```

- 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:

  ```
  Jan   Feb   Mar    Apr    May
  12    12    12     12     20
  ```

Return Value

Returns 0 if successful.

Example

```
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\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("EssOtlFindMember 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");
    printf("\nGet handle for attribute member:\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:");
    usValiditySetType =  ESS_VALIDITYSET_TYPE_MBRHDLS;
    DisplayVaryingAttributes(hOutline, hBaseMbr, hAttrDim, ESS_NULL, usValiditySetType);

    mode = ESS_ASSOCIATE_MODE_NOOVERWRITE;
    sts = 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

- EssOtlVaryingGetAttributeIndepDims

EssOtlVaryingAssociateAttributeDimension

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

```
ESS_FUNC_M EssOtlVaryingAssociateAttributeDimension (hOutline, hBaseDim, hAttrDim,
countOfIndepDims, *pIndepDims, *pucIndependentTypes);
```

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Outline context handle
hBaseDim	ESS_HMEMBER_T	Base dimension handle
hAttrDim	ESS_HMEMBER_T	Attribute dimension handle
countOfIndepDims	ESS_INT32_T	The number of independent dimensions that control the varying attribute
*pIndepDims	ESS_HMEMBER_T	Pointer to an array of member handles for the independent dimensions
*pucIndependentTypes	ESS_UCHAR_T	Pointer to an array of independent types contained in *pIndepDims*. The independent types supported are: • ESS_ASSOCIATE_TYPE_DISCRETE • ESS_ASSOCIATE_TYPE_CONTINUOUS

Return Value

Returns 0 if successful.

Example

```
void TestEssOtlVaryingAssociateAttributeDimenion()
{
    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

EssOtlVaryingDisassociateAttribute

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

```
ESS_FUNC_M EssOtlVaryingDisassociateAttribute (hOutline, hBaseMember, hAttrDim, mode,
pValiditySet);
```

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Outline context handle
hBaseMember	ESS_HMEMBER_T	Base member handle
hAttrDim	ESS_HMEMBER_T	Attribute member handle
mode	ESS_INT_T	Association mode. Possible values: • MODE_OVERWRITE • MODE_NOOVERWRITE • MODE_EXTEND
pValiditySet	"ESS_VALIDITYSET_T" on page 710	Pointer to the set of independent members where the disassociation occurs

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):

  ```
  Jan   Feb   Mar   Apr   May
  12    12    16    16    12
  ```

  - 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
    ```

  - 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
    ```

- 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.

  ```
  Jan   Feb   Mar   Apr    May
  12    12    12    12     12
  ```

Return Value

Returns 0 if successful.

Example

```
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("EssOtlFindMember 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");
    printf("\nGet handle for attribute member:\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:");
    usValiditySetType =  ESS_VALIDITYSET_TYPE_MBRHDLS;
    DisplayVaryingAttributes(hOutline, hBaseMbr, hAttrDim, ESS_NULL, usValiditySetType);

    mode = ESS_ASSOCIATE_MODE_NOOVERWRITE;
    sts = EssOtlVaryingAssociateAttribute(hOutline, hBaseMbr, hAttrMbr, 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);
```

```
    /* 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

```
ESS_FUNC_M EssOtlVaryingGetAssociatedAttributes (hOutline, hBaseMember, hAttrDim,
pPerspective, pusCount,
pphMembers, usValiditySetType, **pppValiditySets);
```

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Outline context handle
hBaseMember	ESS_HMEMBER_T	Handle to the member of the base dimension
hAttrDim	ESS_HMEMBER_T	Attribute member handle

Parameter	Data Type	Description
pPerspective	"ESS_PERSPECTIVE_T" on page 708	Pointer to the collection of independent members used when querying the client or server for associations
pusCount	ESS_PUSHORT_T	Pointer to the number of varying attribute members returned
pphMembers	ESS_PPHMEMBER_T	Pointer to the array of attribute member handles
usValiditySetType	ESS_USHORT_T	Type of validity set assigned to independent members: • 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). • ESS_VALIDITYSET_TYPE_MBRNAMS—As an XRange of member names
**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

```
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_PHMEMBER_T            phAttrMbrs;
    ESS_PVALIDITYSET_T  *ppValiditySets;
    ESS_PMBRINFO_T            pMemberInfo1, pMemberInfo2;

    sts = EssOtlVaryingGetAssociatedAttributes(hOutline,
            hBaseMbr,
            hAttrDim,
            pPerspective,
            &Count,
            &phAttrMbrs,
            usValiditySetType,
            &ppValiditySets);
    printf("\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");

                            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");

                            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");
                    }
                    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]);
            }
            printf("\n");
        }
        else
            printf("\n\tNo 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

```
ESS_FUNC_M EssOtlVaryingGetAttributeIndepDims (hOutline, hAttrMember,
*pCountOfIndepDims, **ppIndepDims, **ppucIndependentTypes);
```

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Outline context handle
hAttrMember	ESS_HMEMBER_T	Attribute member handle
*pCountOfIndepDims	ESS_INT32_T	Pointer to the number of independent dimensions that control the varying attributes
**ppIndepDims	ESS_HMEMBER_T	Pointer to an array of member handles for the independent dimensions
**ppucIndependentTypes	ESS_UCHAR_T	Pointer to an array of independent types contained in *pIndepDims

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

```
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;
```

```
    ESS_INT32_T    pCountOfIndepDims;
    ESS_HMEMBER_T    *ppIndepDims;
    ESS_PMBRINFO_T    pMemberInfo;
    ESS_UCHAR_T    *pucIndependentTypes;


    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\t%s", pMemberInfo->szMember);
            switch(pucIndependentTypes[i])
            {
                case ESS_ASSOCIATE_TYPE_CONTINUOUS:
                    printf(" - (Continuous)");
                    break;
                case ESS_ASSOCIATE_TYPE_DISCRETE:
                    printf(" - (Discrete)");
                    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\n",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);
```

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Outline context handle.
hCtx	ESS_HCTX_T	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.
FormulaString	ESS_STR_T	The syntactic formula expression.
pErrorNumber	ESS_PULONG_T	Pointer to the count of errors.
pErrorLine	ESS_PULONG_T	Pointer to the error line number.
MemberName	ESS_STR_T	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.
ErrorBufferLength	ESS_ULONG_T	The size of the error buffer.
ErrorMessage	ESS_STR_T	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.

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

```
ESS_FUNC_M EssOtlVerifyOutline (hOutline, pulErrors, pulCount, pMbrErrors);
```

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Outline context handle.
pulErrors	ESS_PULONG_T	Pointer to bitmask destination for global outline errors. Currently, this field has only one value: ESS_OUTERROR_CURTOOMANYDIMS
pulCount	ESS_PULONG_T	Count of members with errors. This defines the number of elements of the *pMbrErrors* array.
pMbrErrors	"ESS_OUTERROR_T" on page 704	Pointer to an array with **pulCount* members. Each element of the array contains the errors for a single member.

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

```
#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

```
ESS_FUNC_M EssOtlVerifyOutlineEx (hOutline, pulErrors, pulCount, pMbrErrors);
```

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Outline context handle.
pulErrors	ESS_PULONG_T	Pointer to bitmask destination for global outline errors. If the outline had formula errors the only field with a value is: ESS_OUTERROREX_OUTLINEHASFORMULAERROR
pulCount	ESS_PULONG_T	Count of members with errors. This defines the number of elements 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.
pMbrErrors	"ESS_OUTERROR_T" on page 704	Pointer to an array with **pulCount* members. Each element of the array contains the errors for a single member.

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

```
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.argn < 2)
   {
      hOutlineChoice = ishOutlineMenu(cmdctxp);
   }
   else
   {
      hOutlineChoice = atoi(*(cmdctxp->cmdbuf.args + 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],
                                                                                pMbrE
rrors[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)
                                {
```

```
                                        fprintf(cmdctxp->output,"
  ESS_OUTERROR_NOTIMEDIM\n");
                                }

                                if (pMbrErrors[ind].ulErrors &
ESS_OUTERROR_DUPLICATEALIAS)
                                {
                                        fprintf(cmdctxp->output,"
  ESS_OUTERROR_DUPLICATEALIAS\n");
                                }

                                if (pMbrErrors[ind].ulErrors & ESS_OUTERROR_MEMBERCALC)
                                {
                                        fprintf(cmdctxp->output,"
  ESS_OUTERROR_MEMBERCALC\n");
                                }

                                if (pMbrErrors[ind].ulErrors &
ESS_OUTERROR_SHARENOTLEVEL0)
                                {
                                        fprintf(cmdctxp->output,"
  ESS_OUTERROR_SHARENOTLEVEL0\n");
                                }

                                if (pMbrErrors[ind].ulErrors &
ESS_OUTERROR_NOSHAREPROTO)
                                {
                                        fprintf(cmdctxp->output,"
  ESS_OUTERROR_NOSHAREPROTO\n");
                                }

                                if (pMbrErrors[ind].ulErrors & ESS_OUTERROR_TIMESPARSE)
                                {
                                        fprintf(cmdctxp->output,"
  ESS_OUTERROR_TIMESPARSE\n");
                                }

                                if (pMbrErrors[ind].ulErrors & ESS_OUTERROR_LEAFLABEL)
                                {
                                        fprintf(cmdctxp->output,"
  ESS_OUTERROR_LEAFLABEL\n");
                                }

                                if (pMbrErrors[ind].ulErrors &
ESS_OUTERROR_ALIASSHARED)
                                {
                                        fprintf(cmdctxp->output,"
  ESS_OUTERROR_ALIASSHARED\n");
                                }

                                if (pMbrErrors[ind].ulErrors & ESS_OUTERROR_BADTIMEBAL)
                                {
                                        fprintf(cmdctxp->output,"
  ESS_OUTERROR_BADTIMEBAL\n");
                                }

                                if (pMbrErrors[ind].ulErrors & ESS_OUTERROR_BADSKIP)
```

```
                                {
                                        fprintf(cmdctxp->output,"   ESS_OUTERROR_BADSKIP
\n");
                                }

                                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_DUPGENLEVNAME)
                                {
                                        fprintf(cmdctxp->output,"
  ESS_OUTERROR_DUPGENLEVNAME\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_ILLEGALORDER\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_CHILDCOUNT)
                               {
                                        fprintf(cmdctxp->output,"
  ESS_OUTERROR2_CHILDCOUNT\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)
```

```
                                {
                                        fprintf(cmdctxp->output,"
  ESS_OUTERROR2_ILLEGALATTRVALUE\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");
                       }

                       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\n", sts);

   return(sts);
}
```

See Also

- EssOtlNewOutline
- EssOtlOpenOutline
- EssOtlWriteOutline
- EssOtlVerifyOutline
- EssOtlVerifyFormula

EssOtlWriteOutline

Writes the existing outline information to disk.

Syntax

```
ESS_FUNC_M EssOtlWriteOutline (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 **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.

Example

```
#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

```
ESS_FUNC_M EssOtlWriteOutlineEx(hOutline, pObject, iOtlType);
```

Parameter	Data Type	Description
hOutline	ESS_HOUTLINE_T	Outline context handle.
pObject	ESS_POBJDEF_T	Outline object to write.
iOtlType		Whether the outline is saved in Unicode mode or non-Unicode mode. The valid values are: • ESS_OUTLINE_UTF8 0x0002—Encoded in UTF-8. • ESS_OUTLINE_NONUNICODE 0x0003—Not Unicode-encoded.

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`

12 C Outline API Examples

In This Chapter

Example of Traversing an Outline 981
Extended Member Query Code Example 982

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.

```
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

```
#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;
   ESS_BOOL_T*  pCurrConv = NULL;
   ESS_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_PARENTNAME, (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
   {
```

```
      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    ------------------>\n"); }
   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

13 Using the C Grid API

In This Chapter

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

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). 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:

```
#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]`.

14 C Grid API Declarations

In This Chapter

C Grid API Constants 997
C Grid API Data Types 1004
C Grid API Structures 1006

C Grid API Constants

The following constants are defined in the Essbase Grid API.

Returned on successful API call.

Constant	Definition
ESSG_STS_NOERR	0

Defines the version of the EGAPI API. Changes each time the API is modified.

Constant	Definition
ESSG_VERSION	0x00040000

Define the maximum number of rows and columns supported

Constant	Definition
WINDOWS	
ESSG_MAXROWS	0xFFFFFFFF / sizeof(ESSG_PDATA_T)
ESSG_MAXCOLUMNS	0xFFFFFFFF / sizeof(ESSG_DATA_T)
OTHER	
ESSG_MAXROWS	0xFFFF / sizeof(ESSG_PDATA_T)
ESSG_MAXCOLUMNS	0xFFFF / sizeof(ESSG_DATA_T)

Used by the *pAttributes* member of the "ESSG_DATA_T" on page 1006 structure.

Constant	Definition
ESSG_CA_READONLY	0x00000001
ESSG_CA_READWRITE	0x00000002
ESSG_CA_LINKEDOBJ	0x00000004
ESSG_CA_LINKPARTITION	0x00000008
ESSG_CA_LINKCELLNOTE	0x00000010
ESSG_CA_LINKWINAPP	0x00000020
ESSG_CA_LINKURL	0x00000040
ESSG_CA_AISDT	0x00000080
ESSG_CA_GLDT	0X00000400

Used for the *pAttributes* for members in the "ESSG_DATA_T" on page 1006 structure.

Constant	Definition
ESSG_MA_DIMTOP	0x00000001
ESSG_MA_ZOOMINABLE	0x00000002
ESSG_MA_NEVERSHARE	0x00000004
ESSG_MA_LABELONLY	0x00000008
ESSG_MA_STOREDATA	0x00000010
ESSG_MA_EXPSHARE	0x00000020
ESSG_MA_IMPSHARE	0x00000040
ESSG_MA_DYNCALC	0x00000080
ESSG_MA_FORMULA	0x00000100
ESSG_MA_ATTRIBUTE	0x00000200
ESSG_MA_DIMNUMBITS	0xF8000000 (the last 5 bits contain the dimension number)

Used by the *usType* member in the "ESSG_DATA_T" on page 1006 structure.

Constant	Definition
ESSG_DT_UNUSED	0
ESSG_DT_STRING	1
ESSG_DT_LONG	2

Constant	Definition
ESSG_DT_DOUBLE	3
ESSG_DT_BLANK	4
ESSG_DT_RESERVED	5
ESSG_DT_ERROR	6
ESSG_DT_MISSING	7
ESSG_DT_ZERO	8
ESSG_DT_NOACCESS	9
ESSG_DT_MEMBER	10
ESSG_DT_FORMULA	11
ESSG_DT_ZEROwFORMULA	12
ESSG_DT_DOUBLEwFORMULA	13
ESSG_DT_BLANKwFORMULA	14
ESSG_DT_STRINGwFORMULA	15
ESSG_DT_MISSINGwFORMULA	16
ESSG_DT_NOACCESSwFORMULA	17
ESSG_DT_STRINGEX	18
ESSG_DT_MEMBEREX	19
ESSG_DT_STRINGEXwFORMULA	20
ESSG_DT_FORMULAEX	21
ESSG_DT_MEMBERwKEY (see "ESSG_DT_MEMBERwKEY Example" on page 1109)	23
ESSG_DT_SMARTLIST	24
ESSG_DT_MNGLESS	25
ESSG_DT_DATE	26

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**

Constant	Definition
ESSG_RET_RETRIEVE	0
ESSG_RET_RETRIEVELOCK	1
ESSG_RET_LOCKONLY	2

Used by the ulOptions parameter of **EssGBeginUpdate**

Constant	Definition
ESSG_RET_REQUIRELOCK	0
ESSG_RET_LOCKIFNEEDED	1

This bitmask contant is used by the ulOptions parameter of **EssGBeginConditionalRetrieve**, **EssGBeginConditionalZoomIn**, **EssGBeginReport**, and **EssGBeginReportFile**

Constant	Definition
ESSG_NOATTRIBUTES	0x00001000

These bitmask constants are used by the ulOptions parameter of **EssGBeginZoomIn** and **EssGBeginConditionalZoomIn**

Constant	Definition
ESSG_ZOOM_DOWN	0x00000080
ESSG_ZOOM_ACROSS	0x00000100

Describe the connect options

Constant	Definition
ESSG_CONNECT_DEFAULT	0
ESSG_CONNECT_NODIALOG	1

Describe the various zoom levels

Constant	Definition
ESSG_OPTIONS	0
ESSG_NEXTLEVEL	1
ESSG_ALLLEVELS	2
ESSG_BOTTOMLEVEL	3

Constant	Definition
ESSG_SIBLEVEL	4
ESSG_SAMELEVEL	5
ESSG_SAMEGENERATION	6
ESSG_CALCLEVEL	7
ESSG_PARENTLEVEL	8
ESSG_TOPLEVEL	9

Used for setting and retrieving grid options

Constant	Definition
ESSG_OP_DRILLLEVEL	1
ESSG_OP_INCSEL	2
ESSG_OP_SELONLY	3
ESSG_OP_SELGROUP	4
ESSG_OP_INDENT	5
ESSG_OP_SUPMISSING	6
ESSG_OP_SUPZEROS	7
ESSG_OP_SUPUNDER	8
ESSG_OP_UPDATEMODE	9
ESSG_OP_ALIASNAMES	10
ESSG_OP_ALIASTABLE	11
ESSG_OP_USERGRIDDATA	12
ESSG_OP_RETAINTHREAD	20
ESSG_OP_EMPTYGRIDERROR	21
ESSG_OP_DRILLONLEAF	22
ESSG_OP_DATALESS	23
ESSG_OP_SPANHYBRIDANALYSIS	24
ESSG_OP_UNIQUENAMEONLY	32
ESSG_OP_MEMBERANDUNIQUENAME (see "ESSG_OP_MEMBERANDUNIQUENAME Example" on page 1107)	33

Constant	Definition
ESSG_OP_GET_ME_CELLS	36 Return #ME (meaningless) value for cells with no base member-attribute combination Default: off
ESSG_OP_GET_FORMATTED_VALUE	38 Include formatting values for formatted cells Default: Return only cell values
ESSG_OP_GET_VALUE	39 Requests original values for cells with non-numeric types Default: on
ESSG_OP_GET_FORMATTED_MISSING	40 Include formatting values for cells with missing values Default: off
ESSG_OP_GET_DRILLTHRU_URLS	41 Populates the drill through flag for cells

Describe the various indent styles

Constant	Definition
ESSG_INDENTNONE	1
ESSG_INDENTSUBITEMS	2
ESSG_INDENTTOTALS	3

Used by get results calls to determine the process state

Constant	Definition
ESSG_STATE_DONE	1
ESSG_STATE_INPROGRESS	2

Buffer length constants (including terminating null)

Constant	Definition
ESSG_USERNAMELEN	31
ESSG_PASSWORDLEN	101

Constant	Definition
ESSG_SERVERLEN	31
ESSG_APPLICATIONLEN	9
ESSG_DATABASELEN	9

Constants used by Grid API Drill-Through functions (`EssGDT*xxx*()`)

Constant	Definition
ESSG_DESCRIPTION_LEN	Maximum buffer length (255) used for report data
ESSG_DTINPUTOPTION_PROMPT_HISNAME	*uInputOption* value in **ESSG_DTINFO_T**, meaning that users have all the default values needed to connect to Essbase Studio and start a drill-through session
ESSG_DTINPUTOPTION_PROMPT_LOGIN	*uInputOption* value in **ESSG_DTINFO_T**, meaning that users must set the password to connect to Essbase Studio and start a drill-through session
ESSG_DTREPORT_NAME	Maximum string length (80) used for drill-through
ESSG_ERR_INVALIDDTHANDLE	Error message constant returned if the given drill-through instance handle is invalid
ESSG_ERR_NODTREPORTS	Error message constant returned if no drill-through report is defined for the given drill-through instance handle
ESSG_FIELDLEN	Maximum string length (30) used for drill-through
ESSG_HISDT	Value (5) used for drill-through entry

Used by LRO API calls in the structure ESSG_LRODESC_T

Constant	Definition
ESSG_PARTITIONTYPE	1
ESSG_CELLNOTETYPE	2
ESSG_WINAPPTYPE	3
ESSG_URLTYPE	4

Grid Perspective Types

Used by `EssGGetGridPerspective` and `EssGSetGridPerspective`.

Constant	Definition
ESSG_PERSP_EXPLICIT	0 Requires tuple specification

Constant	Definition
ESSG_PERSP_REALITY	1 Uses reality context for attribute dimension

Text List (SmartList) Types

Text List (SmartList) attributes.

Constant	Definition
ESSG_CA_MISSINGCELL	0x00000100 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.
ESSG_CA_OUTOFRANGE	0x00000200 Set when a SmartList-type cell with a numeric value is out of range in the context of that text list

Unicode Mode Types

Used as values for *usApiType* field of ESSG_INIT_T for Unicode Mode.

Constant	Definition	Description
ESSG_API_UTF8	0x0003	This value enables Essbase Server to create or migrate Unicode-mode applications.
ESSG_API_NONUNICODE	0x0002	This value disables the creation and migration of Unicode-mode applications on Essbase Server.

C Grid API Data Types

Data Type	Essbase Type
typedef char ESSG_APPLICATION_T[ESSG_APPLICATIONLEN];	ESSG_APPLICATION_T
typedef unsigned char ESSG_BOOL_T;	ESSG_BOOL_T
typedef char ESSG_CHAR_T;	ESSG_CHAR_T
typedef char ESSG_DATABASE_T[ESSG_DATABASELEN];	ESSG_DATABASE_T
typedef double ESSG_DOUBLE_T;	ESSG_DOUBLE_T
typedef ESSG_PVOID_T ESSG_DTHINST_T, *ESSG_PDTHINST_T	ESSG_DTHINST_T, ESSG_PDTHINST_T
typedef float ESSG_FLOAT_T;	ESSG_FLOAT_T
typedef ESSG_PVOID_T ESSG_HANDLE_T, *ESSG_PHANDLE_T;	ESSG_HANDLE_T, ESSG_PHANDLE_T

Data Type	Essbase Type
typedef ESSG_PVOID_T ESSG_HGRID_T,*ESSG_PHGRID_T;	ESSG_HGRID_T, ESSG_PHGRID_T
typedef long ESSG_LONG_T;	ESSG_LONG_T
typedef char ESSG_PASSWORD_T[ESSG_PASSWORDLEN];	ESSG_PASSWORD_T
typedef char *ESSG_PSTR_T;	ESSG_PSTR_T
typedef ESSG_VOID_T *ESSG_PVOID_T;	ESSG_PVOID_T
typedef char ESSG_SERVER_T[ESSG_SERVERLEN];	ESSG_SERVER_T
typedef short ESSG_SHORT_T;	ESSG_SHORT_T
typedef char *ESSG_STR_T;	ESSG_STR_T
typedef long ESSG_STS_T;	ESSG_STS_T
typedef unsigned char ESSG_UCHAR_T;	ESSG_UCHAR_T
typedef unsigned long ESSG_ULONG_T;	ESSG_ULONG_T
typedef char ESSG_USERNAME_T[ESSG_USERNAMELEN];	ESSG_USERNAME_T
typedef unsigned short ESSG_USHORT_T;	ESSG_USHORT_T
typedef void ESSG_VOID_T;	ESSG_VOID_T
typedef unsigned short ESSG_WORD_T;	ESSG_WORD_T

ESSG_PFUNC_T, ESSG_PFUNC_M

These types define the prototype for a user's message callback function.

```
#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,
  SSG_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:

```
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;
```

Data Type	Field	Description
ESSG_SERVER_T	*Server*	Name of the server
ESSG_APPLICATION_T	*Application*	Name of the application
ESSG_DATABASE_T	*Database*NNN	Name of the Essbase database
ESSG_USERNAME_T	*Username*	User's name
ESSG_PASSWORD_T	*Password*	User's password

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.

```
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);
```

Data Type	Field	Description
ESSG_PVOID_T	*pAttributes*	One of the long integer constants listed below indicating the cell type or member type (OUT)
ESSG_DATA_VALUE_T	*Value*	The value of the returned grid string
ESSG_USHORT_T	*usType*	One of the tag constants listed below indicating the data type (IN/OUT)
ESSG_PVOID_T	*pCellProps*	Stores cell properties; for example, whether or not cell is associated with a drill-through URL

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_MA_STOREDATA
ESSG_MA_EXPSHARE
```

```
ESSG_MA_IMPSHARE
ESSG_MA_DYNCALC
ESSG_MA_FORMULA
ESSG_MA_ATTRIBUTE
ESSG_MA_DIMNUMBITS
```

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_MEMBEREX
ESSG_DT_STRINGEXwFORMULA
ESSG_DT_FORMULAEX
ESSG_DT_MEMBERwKEY
```

The following constants are additional values for the *usType* field, to work in Unicode mode.

Constant	Definition	Description
ESSG_DT_STRINGEX	0x0018	This value specifies a string extended for Unicode mode.
ESSG_DT_MEMBEREX	0x0019	This value specifies a member name extended for Unicode mode.
ESSG_DT_STRINGEXwFORMULA	0x0020	This value specifies a formula string extended for Unicode mode.
ESSG_DT_FORMULASEX	0x0021	This value specifies a formula extended for Unicode mode.

ESSG_DRILLDATA_T

Contains information associating linked objects with specific cell addresses. The fields are described as follows:

```
typedef struct ESSG_DRILLDATA_T
{
   ESSG_HLRO_T       hLRO;
   ESSG_USHORT_T     usLinkObjType;
   ESSG_LINKOBJDESC  Description;
   ESSG_PSTR_T       pMbrCombos;
```

```
   ESSG_ULONG_T      ulNumMbrCombos;
} ESSG_DRILLDATA_T, * ESSG_PDRILLDATA_T, ** ESSG_PPDRILLDATA_T;
```

Data Type	Field	Description
ESSG_HLRO_T	*hLRO*	A unique handle to a linked object
ESSG_USHORT_T	*usLinkObjType*	Object type
ESSG_LINKOBJDESC	*Description)OOo*	Object description
ESSG_PSTR_T	*pMbrCombos*	An array of member names
ESSG_ULONG_T	*ulNumMbrCombos*	Number of member names in *pMbrCombos*

ESSG_DTDATA_T

Defines a report data cell.

```
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;
```

Data Type	Field	Description
ESSG_ULONG_T	*row*	0-indexed row number for the given data block
ESSG_ULONG_T	*column*	0-indexed column number for the given data block
ESSG_CHAR_T	*data* [ESSG_DESCRIPTION_LEN + 1]	Data value for the given data block

ESSG_DTHEADER_T

Defines header information for a specific column.

```
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;
```

Data Type	Field	Description
ESSG_ULONG_T	*colIndex*	0-based index of the column position
ESSG_CHAR_T	*viewName*[ESSG_DESCLEN_+ 1]	
ESSG_CHAR_T	*data*[ESSG_DESCLEN_+ 1]	Heading text for the given column of data

Data Type	Field	Description
ESSGDTREPORTDATATYPE	*dataType*	One of the constants listed below indicating the data type of the given column of data

Constants for ESSG_DTHEADER_T

The following constants are used by thedataType field of the ESSG_DTHEADER_T structure:

```
ESSGDTINT
ESSGDTFLOAT
ESSGDTSTRING
```

ESSG_DTINFO_T

Defines the connection information for a range of data cells.

```
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];
   ESSG_USHORT_T inputOption;
} ESSG_DTINFO_T, *ESSG_PDTINFO_T, **ESSG_PPDTINFO_T;
```

Data Type	Field	Description
ESSG_CHAR_T	*hisName* [ESSG_FIELDLEN + 1]	
ESSG_CHAR_T	*dataSource* [ESSG_FIELDLEN + 1]	(read only)
ESSG_CHAR_T	*username* [ESSG_FIELDLEN + 1]	
ESSG_CHAR_T	*password* [ESSG_FIELDLEN + 1]	(write only)
ESSG_USHORT_T	*inputOption*	(read only)

ESSG_DTREPORT_T

Defines a report definition.

```
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;
```

Data Type	Field	Description
ESSG_LONG_T	*reportId*	
ESSG_CHAR_T	*name* [ESSG_DESCLEN + 1]	
ESSG_LONG_T	*customize*	
ESSG_LONG_T	*rowGoverner*	
ESSG_LONG_T	*timeGoverner*	

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;
```

Data Type	Field	Description
ESSG_ULONG_T	*ulVersion*	This should be set to ESSG_VERSION
ESSG_ULONG_T	*ulMaxRows*	Maximum number of rows for the grid Limit: 65535 rows
ESSG_ULONG_T	*ulMaxColumns*	Maximum number of columns for the grid Limit: 256 columns
ESSG_PFUNC_T	*pfnMessageFunc*	Pointer to the user-defined message callback function
ESSG_PVOID_T	*pUserdata*	Pointer to user data passed to message callback
ESSG_UGSHORT_T	*usApiType*	Encoding type of the Grid API. For valid values, see "Unicode Mode Types" on page 1004.

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
```

```
      {
         ESSG_LROINFO_T  lroInfo;
         ESSG_CHAR_T     Note[ESSG_LRONOTELEN];
      } lro;
} ESSG_LRODESC_T, *ESSG_LPLRODESC_T;
```

Data Type	Field	Description
ESSG_ULONG_T	*usLinkObjType*	Object type
ESSG_USERNAME_T	*userName*	Name of the last user to modify the object
ESSG_TIME_T	*LastUpdate*	Last date the object was modified ESSG_TIME_T is defined as an unsigned long
ESSG_LROINFO_T	*lroInfo*	LRO information structure, associated by union
ESSG_CHAR_T	*Note*[ESSG_LRONOTELEN]	A cell note, associated by union 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:

```
typedef struct ESSG_LROINFO_T
{
   ESSG_CHAR_T  ObjName[ESSG_ONAMELEN];
   ESSG_CHAR_T  Desc[ESS_DESCLEN];
} ESSG_LROINFO_T, *ESSG_LPLROINFO_T;
```

Data Type	Field	Description
ESSG_ CHAR_T	*objName*[ESSG_ ONAMELEN]	Source file name of object linked to a data cell ESSG_ONAMELEN specifies the maximum length of an object name; the default value is 511.
ESSG_ CHAR_T	*Desc*[ESS_DESCLEN]	Description of an object linked to a data cell ESS_DESCLEN specifies the maximum length of the description; the default value is 79.

ESSG_RANGE_T

Describes the extent of the data being sent or received.

```
typedef struct
{
    ESSG_ULONG_T ulRowStart;
    ESSG_ULONG_T ulColumnStart;
    ESSG_ULONG_T ulNumRows;
    ESSG_ULONG_T ulNumColumns;
} ESSG_RANGE_T, *ESSG_PRANGE_T;
```

Data Type	Field	Description
ESSG_ULONG_T	*ulRowStart*	First Row in the report (zero based)
ESSG_ULONG_T	*ulColumnStart*	First Column in the report (zero based)
ESSG_ULONG_T	*ulNumRows*	Number of rows in the report (maximum 16370)
ESSG_ULONG_T	*ulNumColumns*	Number of columns in the report (maximum 256)

15 C Grid API Function Reference

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

```
ESSG_FUNC_M EssGBeginConditionalRetrieve (hGrid, pszConditions,
ulOptions);
```

Parameter	Data Type	Description
hGrid	ESSG_HGRID_T	Handle passed back from **EssGNewGrid**.
pszConditions	ESSG_STR_T	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.
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) 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 **ESSG_STS_NOERR**.

Access

None.

Example

```
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,
          &rDataRangeOut, &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

```
ESSG_FUNC_M EssGBeginConditionalZoomIn (hGrid, pZoomCell,
pszConditions, ulOptions);
```

Parameter	Data Type	Description
hGrid	ESSG_HGRID_T	Handle passed back from **EssGNewGrid**.
pZoomCell	"ESSG_RANGE_T" on page 1012	Describes the cell to be zoomed in upon. This must be a single cell for conditional zoomin.
pszConditions	ESSG_STR_T	String (no greater than 64K) containing Essbase report specification commands relating to the conditions for the zoom-in. Do not use Report Writer member/alias/unique name handling formatting commands for the pszConditions parameter. Use the options available in EssGSetGridOption.

Parameter	Data Type	Description
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 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;
```

```
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);
```

```
      /* free the returned data */
      EssGFreeRows(hGrid, &rDataRangeOut, ppDataOut);
   }

if(sts == 0)
   {
      EssGEndOperation(hGrid, 0);
      EssGDisconnect(hGrid, 0);
   }

}
```

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

```
ESSG_FUNC_M EssGBeginCreateLRO (hGrid, usCells, pCells, pLroDesc,
ulOption);
```

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 create the link.
pLroDesc;	"ESSG_LRODESC_T" on page 1011	LRO description information for the new object.
ulOption;	ESSG_ULONG_T	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).

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

- EssGGetCellLinkResults
- EssGGetLRODesc
- EssGGetLRO
- EssGUpdateLRO

EssGBeginDataPoint

Begins a data point operation.

Syntax

```
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

```
ESSG_VOID_T ESSG_BeginDataPoint(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:");
        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

Begins 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

Begins 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

```
ESSG_FUNC_M EssGBeginDrillOrLink (hGrid, usCells, pDrillCells,
ulOptions);
```

Parameter	Data Type	Description
hGrid;	ESSG_HGRID_T	Grid handle returned by **EssGNewGrid**().
usCells;	ESSG_USHORT_T	The number of cell ranges in the array of ranges specified in *pDrillCells*.
pDrillCells;	"ESSG_RANGE_T" on page 1012	Array of cell ranges to query for links.
ulOptions;	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
- EssGBeginDrillAcross
- EssGFreeCellLinkResults
- EssGGetCellLinkResults

EssGBeginKeepOnly

Begins a keep-only operation to isolate cells to keep, removing all others.

Syntax

```
ESSG_FUNC_M EssGBeginKeepOnly (hGrid, usCells, pKeepCells, 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 *pKeepCells* (the size of array).
pKeepCells	"ESSG_RANGE_T" on page 1012	Describes the cells to be kept. The members to be kept applies only to one dimension. That is, if the user decides to keep, for example, "Qtr1", then all other members of the Time dimension will be removed and the only representative of the Time dimension will be "Qtr1". 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.
ulOptions	ESSG_ULONG_T	Reserved for future use. Should be set to zero.

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

```
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)
```

```
  {
      EssGEndOperation(hGrid, 0);
     EssGDisconnect(hGrid, 0);
  }
}
```

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

```
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

```
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);
     EssGDisconnect(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

```
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

```
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

Begins a remove-only operation, isolating the cells to be removed.

Syntax

```
ESSG_FUNC_M EssGBeginRemoveOnly (hGrid, usCells, pRemoveCells, 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 *pRemoveCells* (the size of array).

Parameter	Data Type	Description
pRemoveCells	"ESSG_RANGE_T" on page 1012	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.
ulOptions	ESSG_ULONG_T	Reserved for future use. Should be set to zero.

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

```
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)
```

Parameter	Data Type	Description
hGrid	ESSG_HGRID_T	Handle passed back from **EssGNewGrid**.
pszReportIn	ESSG_STR_T	String (no greater than 64K) containing an Essbase report specification.
ulOptions	ESSG_ULONG_T	A bitmask which describes returned grid options. Valid values are: **ESSG_NOATTRIBUTES** returns grid without pAttributes values.

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;
```

```
   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);
```

Parameter	Data Type	Description
hGrid	ESSG_HGRID_T	Handle passed back from **EssGNewGrid**.
pszReportName	ESSG_STR_T	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.
bLocal	ESSG_BOOL_T	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.
ulOptions	ESSG_ULONG_T	A bitmask which describes returned grid options. Valid values are: **ESSG_NOATTRIBUTES** returns grid without pAttributes values

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 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 **ESSG_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 **ESSG_STS_NOERR**.

Access

None.

Example

```
ESSG_VOID_T ESSG_BeginReportFile (ESSG_HGRID_T hGrid)

{
  ESSG_FUNC_M     sts = ESS_STS_NOERR;
  ESSG_PPDATA_T         ppDataOut;
  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

```
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

```
#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**.

Access

None.

See Also

- "Using the C Grid API Functions" on page 994
- "C Grid API Structures" on page 1006
- EssGBeginZoomIn
- EssGBeginConditionalZoomIn
- EssGBeginConditionalZoomIn

EssGBeginUpdate

Begins an update of data at the server. This function returns no data to the caller.

Syntax

```
ESSG_FUNC_M EssGBeginUpdate (hGrid, ulOptions);
```

Parameter	Data Type	Description
hGrid	ESSG_HGRID_T	Handle passed back from **EssGNewGrid**.
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 suffcient to call **EssGSendRows** and **EssGPerformOperation**.

Return Value

If successful, returns **ESSG_STS_NOERR**.

Access

None.

Example

```
ESSG_VOID_T ESSG_BeginUpdate (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_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

```
ESSG_FUNC_M EssGBeginZoomIn (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).

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);
```

```
  }
}
```

See Also

- "Using the C Grid API Functions" on page 994
- "C Grid API Structures" on page 1006

EssGBeginZoomOut

Begins a zoom-out.

Syntax

```
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

```
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;
  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);
```

```
    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

- "C Grid API Structures" on page 1006

EssGCancelOperation

Cancels an operation at any stage during an operation.

Syntax

```
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

```
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

```
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

```
ESSG_VOID_T ESSG_Cell (ESSG_HGRID_T hGrid)

{
  ESSG_FUNC_M    sts = ESS_STS_NOERR;
```

```
  ESSG_USHORT_T        usCount;
  ESSG_DATA_T          DataCell;
  ESSG_CHAR_T          *pszMbrs[5] = { "Actual", "Jan",
                                       "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);
}
```

See Also

- "Using the C Grid API Functions" on page 994
- "C Grid API Structures" on page 1006

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);
```

Parameter	Data Type	Description
pszMember	ESSG_STR_T	Member name (input).
pszKey	ESSG_STR_T	Member key (input).
*pszOutStr	ESSG_STR_T	Output string of the format: `<member-name length><member-name><key length ><key>` where the length elements are 2 bytes in size. Note that <member-name> is null-terminated.

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);
```

Parameter	Data Type	Description
hGrid	ESSG_HGRID_T	Handle passed back from EssGNewGrid.
Server	ESSG_SERVER_T	Network server name string. The server name can be expressed as *hostname* or *hostname*:*port*.
Username	ESSG_USERNAME_T	Name of valid user at server.
Password	ESSG_PASSWORD_T	Password of user.
Application	ESSG_APPLICATION_T	Name of a valid application on server.
Database	ESSG_DATABASE_T	Name of a valid database for application on server.
ulOptions	ESSG_ULONG_T	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.

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 utilitized. 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

```
#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

```
ESSG_FUNC_M EssGConnectEx (hGrid, Server, Token, Application, Database, ulOptions);
```

Parameter	Data Type	Description
hGrid	ESSG_HGRID_T	Handle passed back from EssGNewGrid.
Server	ESSG_SERVER_T	Network server name string. The server name can be expressed as *hostname* or *hostname*:*port*.

Parameter	Data Type	Description
Token	ESSG_TOKEN_T	The token representing the username and password of an authenticated user.
Username	ESSG_USERNAME_T	Name of valid user at server.
Password	ESSG_PASSWORD_T	Password of user.
Application	ESSG_APPLICATION_T	Name of a valid application on server.
Database	ESSG_DATABASE_T	Name of a valid database for application on server.
ulOptions	ESSG_ULONG_T	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.

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 utilitized. 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

```
ESSG_FUNC_M EssGDeleteLRO (hGrid, hLRO);
```

Parameter	Data Type	Description
hGrid;	ESSG_HGRID_T	Grid handle returned by **EssGNewGrid**().
hLRO;	ESSG_HLRO_T	Handle to the linked object (returned in a DRILLDATA structure by the **EssGGetCellLinkResults**() function).

Notes

To delete *all* objects linked to a particular range of cells, use `EssGBeginDeleteLROs`.

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`
- `EssGFreeCellLinkResults`
- `EssGGetCellLinkResults`
- `EssGGetLRODesc`
- `EssGGetLRO`
- `EssGUpdateLRO`

EssGDestroyGrid

Destroys a grid instance.

Syntax

ESSG_FUNC_M **EssGDestroyGrid** (*hGrid*)

Parameter	Data Type	Description
hGrid	ESSG_HGRID_T	Handle passed back from **EssGNewGrid**.

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

```
#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);

/* destroys a grid instance */
if(!sts)
   sts = 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

```
ESSG_FUNC_M EssGDisconnect (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**.

Example

```
#include <essapin.h>
#include <essgapin.h>

ESSG_FUNC_M     sts = 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

- "C Grid API Structures" on page 1006
- EssGDTEndDrillThrough
- EssGDTGetData
- EssGDTGetHeader
- EssGDTGetInfo
- EssGDTGetReportData
- EssGInit
- EssGDTListReports
- EssGDTReportCount

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_PDTHINST_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 Constants" on page 997
- "C Grid API Structures" on page 1006
- EssGDTEndDrillThrough
- EssGDTExecuteReport
- EssGDTGetData
- EssGDTGetHeader
- 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

```
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

```
ESSG_FUNC_M EssGDTGetData (pDTInst, ppData, pulCount);
```

Parameter	Data Type	Description
pDTInst;	ESSG_PDTHINST_T	Initialized drill-through instance handle.
ppData;	"ESSG_DTDATA_T" on page 1009	Array of report data structures for given data cells.
pulCount;	ESSG_PULONG_T	Count of data blocks in the *ppData* array.

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
- EssGDTExecuteReport
- EssGDTGetHeader
- EssGDTGetInfo
- EssGDTListReports
- EssGDTRequestDrillThrough
- EssGDTSetInfo
- EssFree

EssGDTGetHeader

Retrieves the report data header information for the given drill-through instance handle.

Syntax

```
ESSG_FUNC_M EssGDTGetHeader (pDTInst, ppHeader, pulCount);
```

Parameter	Data Type	Description
pDTInst;	ESSG_PDTHINST_T	Initialized drill-through instance handle.
ppHeader;	"ESSG_DTHEADER_T" on page 1009	Array of header information structures for given columns.

Parameter	Data Type	Description
pulCount;	ESSG_PULONG_T	Count of data blocks in the *ppHeader* header information array.

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

```
ESSG_FUNC_M EssGDTGetInfo (pDTInst, pDTInfo);
```

Parameter	Data Type	Description
pDTInst;	ESSG_PDTHINST_T	Initialized drill-through instance handle
pDTInfo;	"ESSG_DTINFO_T" on page 1010	Pointer to a structure of connection information for a given range of data cells

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

```
ESSG_FUNC_M EssGDTGetReportData (hGrid, hDAGrid, usCells, pCells);
```

Parameter	Data Type	Description
hGrid;	ESSG_HGRID_T	Handle of the original grid returned by **EssGNewGrid**().
hDAGrid;	ESSG_HGRID_T	Handle of the new grid to receive drill-through report data.
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.

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

```
ESSG_FUNC_M EssGDTListReports (pDTInst, ppReports, pulCount);
```

Parameter	Data Type	Description
pDTInst;	ESSG_PDTHINST_T	Initialized drill-through instance handle
ppReports;	"ESSG_DTREPORT_T" on page 1010	An array of report structures for the given drill-through instance handle
pulCount;	ESSG_PULONG_T	Number of blocks in the *ppReports* header information array

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

```
ESSG_FUNC_M EssGDTReportCount (hGrid, usCells, pCells, uspReportNum);
```

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.
uspReportNum;	ESSG_PUSHORT_T	Number of reports defined for the given data cell range(s).

See Also

- "C Grid API Constants" on page 997
- "C Grid API Structures" on page 1006
- EssGDTEndDrillThrough
- EssGDTGetData

- EssGDTGetHeader
- EssGDTGetInfo
- EssGDTGetReportData
- EssGInit
- EssGDTListReports
- EssGDTRequestDrillThrough

EssGDTRequestDrillThrough

Returns the drill-through instance handle for the given data cell range.

Syntax

```
ESSG_FUNC_M EssGDTRequestDrillThrough (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).

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
- EssGDTExecuteReport
- EssGDTGetData
- EssGDTGetHeader
- EssGDTGetInfo
- EssGDTListReports
- EssGDTSetInfo
- EssOtlGetMemberCommentEx
- EssOtlSetMemberCommentEx

EssGDTSetInfo

Sets drill-through connection information for a given drill-through handle.

Syntax

```
ESSG_FUNC_M EssGDTSetInfo (pDTInst, pDTInfo);
```

Parameter	Data Type	Description
pDTInst;	ESSG_PDTHINST_T	Initialized drill-through instance handle
pDTInfo;	"ESSG_DTINFO_T" on page 1010d	Pointer to a structure of connection information for a given range of data cells

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

```
ESSG_FUNC_M EssGEndOperation (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 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);
```

Parameter	Data Type	Description
hGrid;	ESSG_HGRID_T	Grid handle returned by **EssGNewGrid**().
pDrillData;	"ESSG_DRILLDATA_T" on page 1008	Reference to an array of ESSG_DRILLDATA_T structures containing information about the linked objects.

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`

- EssGBeginDrillOrLink
- EssGBeginRemoveOnly
- EssGDeleteLRO
- EssGGetCellLinkResults
- EssGGetLRO
- EssGUpdateLRO

EssGFreeMemberInfo

Frees any data returned by any call that returns member information, including **EssGGetMemberInfo**, and **EssGGetDataPointResults**.

Syntax

```
ESSG_FUNC_M EssGFreeMemberInfo (hGrid, ulMembers, pszMembers);
```

Parameter	Data Type	Description
hGrid	ESSG_HGRID_T	Handle passed back from **EssGNewGrid**.
ulMembers	ESSG_ULONG_T	Describes the number of elements in the *ppszMembers* array to be freed.
pszMembers	ESSG_PSTR_T	Pointer to a one-dimensional array of member names to be freed.

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

```
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

```
ESSG_FUNC_M 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

```
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

- EssGCreateMemberwKeyStr
- EssGGetFromMemberwKey

EssGFreeRows

Frees data that has been returned via **EssGGetRows**.

Syntax

```
ESSG_FUNC_M EssGFreeRows (hGrid, pRange, ppData);
```

Parameter	Data Type	Description
hGrid	ESSG_HGRID_T	Handle passed back from **EssGNewGrid**.
pRange	"ESSG_RANGE_T" on page 1012	Describes the extent of the data.
ppData	"ESSG_DATA_T" on page 1006	A two-dimensional array of data to be freed.

Return Value

If successful, returns **ESSG_STS_NOERR**.

Access

None.

Example

```
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

```
ESSG_FUNC_M EssGGetAPIContext (hGrid, pEssHctx);
```

Parameter	Data Type	Description
hGrid	ESSG_HGRID_T	Handle passed back from **EssGNewGrid**.
pEssHctx	ESSG_PPVOID_T	Variable for the return of the API context handle of the connected grid.

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

```
#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

EssGGetAPIInstance

Gets the API initialization instance handle.

Syntax

```
ESSG_FUNC_M EssGGetAPIInstance (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);
```

Parameter	Data Type	Description
hGrid;	ESSG_HGRID_T	Grid handle returned by **EssGNewGrid**().
pfCanDrill;	ESSG_PBOOL_T	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.
pNumLROs;	ESSG_PULONG_T	Returns the number of links retrieved.
ppDrillData;	"ESSG_DRILLDATA_T" on page 1008	Returns references to an array of ESSG_DRILLDATA_T structures containing information about the linked objects.
pRangeOut;	"ESSG_RANGE_T" on page 1012	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**().
pState;	ESSG_PUSHORT_T	Returns one of the following states of operation: • In progress—Not all cells have been retrieved • Done—All cells have been retrieved

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

```
ESSG_FUNC_M EssGGetDataPointResults (hGrid, pulMembers, ppszMembers,
pState);
```

Parameter	Data Type	Description
hGrid	ESSG_HGRID_T	Handle passed back from EssGNewGrid.
pulMembers	ESSG_PULONG_T	Count of members being returned.

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

```
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

```
ESS_FUNC_M EssGGetFormattedValue(hGrid, pData, *fmtVal)
```

Parameter	Data Type	Description
hGrid	ESSG_HGRID_T	Grid handle
pData	ESSG_PDATA_T	Pointer to the ESSG_DATA_T structure of the cell.
*fmtVal	ESSG_STR_T	Pointer to formatted value for this cell

Notes

- 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	Data Type	Description
pszOutStr ;	ESSG_STR_T	Input string of the format: <member-name length><member-name><key length ><key>, where the length elements are 2 bytes in size. Note that <member-name> is null-terminated. The string is returned from the API or can be created using EssGCreateMemberwKeyStr.
pszMember;	ESSG_STR_T	Member name (output).
pszKey;	ESSG_STR_T	Member key (output).

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

```
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

- EssGCreateMemberwKeyStr
- EssGFreeMemberwKeyStr

EssGGetGridOption

Gets individual grid options.

Syntax

```
ESSG_FUNC_M EssGGetGridOption (hGrid, sOption, pOption);
```

Parameter	Data Type	Description
hGrid	ESSG_HGRID_T	Handle passed back from **EssGNewGrid**().
sOption	ESSG_SHORT_T	Number indicating what option is being retrieved.
pOption	ESSG_PVOID_T	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.

Return Value

If successful, returns ESSG_STS_NOERR.

Example

```
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);
     EssGDisconnect(hGrid, 0);
  }
}
```

See Also

- "Using the C Grid API Functions" on page 994
- "C Grid API Structures" on page 1006

- EssGSetGridOption

EssGGetGridPerspective

Returns the perspective for a grid.

Syntax

```
ESSG_FUNC_M EssGGetGridPerspective(hGrid, sAttrdim, *pPerspectiveType,
*pPerspectiveString)
```

Parameter	Data Type	Description
hGrid	ESSG_HGRID_T	Handle passed back from EssGNewGrid().
sAttrdim	ESSG_STR_T	Attribute dimension name for which the perspective is queried
*pPerspectiveType	ESSG_SHORT_T	Type of perspective. See "Grid Perspective Types" on page 1003.
*pPerspectiveString	ESSG_STR_T	Pointer to perspective tuple set being returned. • Null for perspective types other than ESSG_PERSP_EXPLICIT. • For ESSG_PERSP_EXPLICIT, this value should be explicitly freed.

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

```
ESS_FUNC_M EssGGetIsCellDrillable (hGrid, pData, pIsDrillable);
```

Parameter	Data Type	Description
hGrid	ESSG_HGRID_T	Grid handle returned by **EssGNewGrid**()
pData	ESS_PDATA_T	Pointer to the ESSG_DATA_T structure of the cell
pIsDrillable	ESS_PBOOL_T	True, if the cell is associated with a drill-through URL; False otherwise

Return Value

- If successful, sets *pIsDrillable* accordingly.
- If unsuccessful, returns an error code.

Example

```
#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

```
ESSG_FUNC_M EssGGetLinkedPartitionDesc (hGrid, hLRO, pConnectInfo);
```

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 must specify a linked object of type ESSG_PARTITIONTYPE.
pConnectInfo;	"ESSG_CONNECTINFO_T" on page 1006	Returns connection information for the linked partition.

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`

- EssGGetCellLinkResults
- EssGGetGridOption
- EssGPerformOperation
- EssGSetGridOption
- EssGBeginCreateLRO
- EssGBeginDrillAcross
- EssGBeginDeleteLROs
- EssGBeginDrillOrLink
- EssGDeleteLRO
- EssGFreeCellLinkResults
- EssGGetCellLinkResults
- EssGGetLRODesc
- EssGGetLRO
- EssGUpdateLRO

EssGGetLRO

Retrieves a LRO from an Essbase database.

Syntax

```
ESSG_FUNC_M EssGGetLRO (hGrid, hLRO, szTargetFile, ulOption);
```

Parameter	Data Type	Description
hGrid	ESSG_HGRID_T	Grid handle returned by **EssGNewGrid**().
hLRO	ESSG_HLRO_T	Handle to the linked object (returned in a DRILLDATA structure by the **EssGGetCellLinkResults**() function).
szTargetFile	ESSG_STR_T	The name of the target file, including path, into which the object is retrieved.
ulOption	ESSG_ULONG_T	Option specifying whether to retrieve the object, its catalog entry, or both. Use one of the following: • ESS_LRO_OBJ_API to retrieve only the object. • ESS_LRO_CATALOG_API to retrieve only the catalog entry. • ESS_LRO_BOTH_API to retrieve object and catalog entry.

Notes

To retrieve a cell note, use EssGGetLRODesc. **EssGGetLRO** does not retrieve cell note information.

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
- EssGGetGridOption
- EssGPerformOperation
- EssGSetGridOption
- EssGBeginCreateLRO
- EssGBeginDrillAcross
- EssGBeginDeleteLROs
- EssGBeginDrillOrLink
- EssGDeleteLRO
- EssGFreeCellLinkResults
- EssGGetCellLinkResults
- EssGGetLRODesc
- EssGGetLinkedPartitionDesc
- EssGUpdateLRO

EssGGetLRODesc

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

- EssGFreeCellLinkResults
- EssGGetCellLinkResults
- EssGGetLRO
- EssGUpdateLRO
- EssGUpdateLRO
- EssGUpdateLRO
- EssGUpdateLRO
- EssGUpdateLRO

EssGGetMemberInfo

Returns member relationship information from within one dimension.

Syntax

```
ESSG_FUNC_M EssGGetMemberInfo (hGrid, pszMbrName, sAction,
bAliases, pulMembers, ppszMembers);
```

Parameter	Data Type	Description
hGrid ;	ESSG_HGRID_T	Handle passed back from EssGNewGrid.
pszMbrName;	ESSG_STR_T	Name of the member for which relationship information will be obtained.
sAction;	ESSG_SHORT_T	Number indicating what type of relationship information will be returned. The following values are valid for this parameter and are 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_PARENTLEVEL Parent of member • ESSG_TOPLEVEL Dimension member belongs to
bAliases ;	ESSG_BOOL_T	Indicates whether alias names will be returned.
pulMembers;	ESSG_PULONG_T	Count of members being returned.
*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.

Notes

- *pszMbrName* cannot be null.
- Free the *ppszMembers* parameter using **EssGFreeMemberInfo**.

Return Value

If successful, returns **ESSG_STS_NOERR**.

Access

None.

Example

```
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_NEXTLEVEL;
  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);
```

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.
pRangeOut	"ESSG_RANGE_T" on page 1012	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 **EssGGetRows**.
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

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);
```

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.
pRangeRequested	"ESSG_RANGE_T" on page 1012	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.
pRangeOut	"ESSG_RANGE_T" on page 1012	Describes the extent of the data returned.
*pppDataOut	"ESSG_DATA_T" on page 1006	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**.

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

```
ESS_FUNC_M EssGGetSmartlistforCell (hGrid, pData, *pSmartlistname)
```

Parameter	Data Type	Description
hGrid	ESSG_HGRID_T	Grid handle
pData	ESSG_PDATA_T	Pointer to the cell ESSG_DATA_T structure of the cell.
*pSmartlistname	ESSG_STR_T	Pointer to name of the TextList object the cell is associated with

Return Value

- 0—If successful
- Error number—If unsuccessful

See Also

- EssOtlFreeSmartListInfo
- EssOtlGetMemberSmartList
- EssOtlGetSmartListInfo
- EssOtlPutSmartList

EssGInit

Initializes the Grid API.

Syntax

```
ESSG_FUNC_M EssGInit (pInitStruct, pHandle);
```

Parameter	Data Type	Description
pInitStruct	"ESSG_INIT_T" on page 1011	Pointer to a structure containing useful information for the EGAPI.
pHandle	ESSG_PHANDLE_T	Pointer to the handle sent back from the EGAPI.

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

```
#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

Parameter	Data Type	Description
hGrid;	ESSG_HGRID_T	Handle from EssGNewGrid()
Server;	ESSG_SERVER_T	Name of a valid server
Username;	ESSG_USERNAME_T	Name of a valid user on the server
Password;	ESSG_PASSWORD_T	User's password
NewPassword;	ESSG_PASSWORD_T	User's new password

Return Value

If successful, returns ESSG_STS_NOERR.

Example

```
#include
#include
```

```
{
  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

```
ESSG_FUNC_M EssGNewGrid (Handle, phGrid);
```

Parameter	Data Type	Description
Handle; /* IN */	ESSG_HANDLE_T	Handle passed back from EssGInit.
phGrid; /* OUT */	ESSG_PHGRID_T	Pointer to the grid-specific handle sent back from the EGAPI.

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 EssGBegin*Operation*().

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_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 serve rusing **EssGBeginXxx** and **EssGSendRows**.

Syntax

```
ESSG_FUNC_M EssGPerformOperation (hGrid, ulOptions);
```

Parameter	Data Type	Description
hGrid;	ESSG_HGRID_T	Handle passed back from **EssGNewGrid**().
ulOptions;	ESSG_ULONG_T	Reserved for future use. Set to zero.

Notes

After this call is made, call **EssGGetResults** to get information about the data returned.

Return Value

If successful, returns **ESSG_STS_NOERR**.

Example

```
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

```
ESSG_FUNC_M EssGSendRows (hGrid, pRangeIn, ppDataIn);
```

Parameter	Data Type	Description
hGrid	ESSG_HGRID_T	Handle passed back from **EssGNewGrid**.
pRangeIn	"ESSG_RANGE_T" on page 1012	Describes the extent of the data in *ppDataIn*.
ppDataIn	"ESSG_DATA_T" on page 1006	A two-dimensional array of cells describing the data.

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

```
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

```
ESSG_FUNC_M EssGSetGridOption (hGrid, sOption, pOption);
```

Parameter	Data Type	Description
hGrid	ESSG_HGRID_T	Handle passed back from **EssGNewGrid**().
		Description
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:

Value	Description	Data Type Expected	Default
ESSG_OP_ALIASNAMES	Alias names	ESSG_BOOL_T	ESSG_FALSE
ESSG_OP_ALIASTABLE	Alias names table	ESSG_STR_T	
ESSG_OP_DATALESS	Enable dataless navigation.	ESSG_BOOL_T	ESSG_FALSE

Value	Description	Data Type Expected	Default
ESSG_OP_DRILLLEVEL	Drill-level	ESSG_SHORT_T	ESSG_NEXTLEVEL
ESSG_OP_EMPTYGRIDERROR	If FALSE, don't issue error on queries which result in no data and return only the grid header.	ESSG_BOOL_T	ESSG_TRUE
ESSG_OP_INCSEL	Include selection	ESSG_BOOL_T	ESSG_FALSE
ESSG_OP_INDENT	Indent style	ESSG_SHORT_T	ESSG_INDENTTOTALS
ESSG_OP_LATEST	Turn on the ability to specify the latest member.	ESSG_BOOL_T	ESSG_FALSE
ESSG_OP_LATESTMEMBER	Specify the latest member.	ESSG_STR_T	NULL
ESSG_OP_REPEATMBRNAMES	Repeat member names.	ESSG_BOOL_T	ESSG_FALSE
ESSG_OP_RETAINTHREAD	If set to TRUE, don't disconnect from server thread at end of grid operation. May improve performance when submitting several operations in sequence.	ESSG_BOOL_T	ESSG_FALSE
ESSG_OP_SELGROUP	Remove Unselected Groups. Zooms on all occurrences of selected member but removes any other members from the same dimension, including the selected member itself.	ESSG_BOOL_T	ESSG_FALSE
ESSG_OP_SELONLY	Within Selected Group. Zooms on only the exact instance of the member that is selected.	ESSG_BOOL_T	ESSG_FALSE
ESSG_OP_SUPMISSING	Suppress missing rows.	ESSG_BOOL_T	ESSG_FALSE
ESSG_OP_SUPUNDER	Replace underscores with spaces.	ESSG_BOOL_T	ESSG_FALSE
ESSG_OP_SUPZEROS	Suppress zero rows.	ESSG_BOOL_T	ESSG_FALSE
ESSG_OP_UPDATEMODE	Update mode	ESSG_BOOL_T	ESSG_FALSE
ESSG_OP_USEBOTHFORROWDIMS	Use both member names and aliases for the row dimensions.	ESSG_BOOL_T	ESSG_FALSE
ESSG_OP_USERGRIDDATA	Pointer to user data	ESSG_PVOID_T	NULL
ESSG_OP_RETAINTHREAD	Retain threads	ESSG_BOOL_T	ESSG_FALSE
ESSG_OP_EMPTYGRIDERROR	Issue an empty grid error	ESSG_BOOL_T	ESSG_FALSE
ESSG_OP_DATALESS	Navigate without data	ESSG_BOOL_T	ESSG_FALSE
ESSG_OP_SPANHYBRIDANALYSIS	Span drill to relational source	ESSG_BOOL_T	ESSG_FALSE

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)
```

Parameter	Data Type	Description
hGrid	ESSG_HGRID_T	Handle passed back from EssGNewGrid().
sAttrdim	ESSG_STR_T	Attribute dimension name for which perspective has to set
sPerspectiveType	ESSG_SHORT_T	Type of perspective. See "Grid Perspective Types" on page 1003.

Parameter	Data Type	Description
pPerspectiveString	ESSG_STR_T	`pPerspectiveString (m1,m2,m3,.....)` This is perspective tuple which should be applied for the given attribute dimension. Level-0 members from one or more "Independent" dimensions (for attrDim) will be the part of the input tuple. If a member from one "independent" dimension is not present in the perspective tuple, the member of the same dimension from the current query/calculation context will be used. In case of an explicit perspective missing for an attribute dimension, the default usage for perspective is ESSG_PERSP_REALITY. This argument can be NULL for sPerspectiveType other than ESSG_PERSP_EXPLICIT, which requires a valid tuple.

Return Value

- 0—If successful
- Error number—If unsuccessful

See Also

- EssGGetGridPerspective

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

```
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

```
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

```
#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);

}
```

See Also

- "Using the C Grid API Functions" on page 994
- "C Grid API Structures" on page 1006

EssGUpdateLRO

Updates the description and contents of a linked object.

Syntax

```
ESSG_FUNC_M EssGUpdateLRO (hGrid, hLRO, pLroDesc, ulOption);
```

Parameter	Data Type	Description
hGrid;	ESSG_HGRID_T	Grid handle returned by **EssGNewGrid**().
hLRO;	ESSG_HLRO_T	Handle to the linked object (returned in a DRILLDATA structure by the **EssGGetCellLinkResults**() function).
pLroDesc;	"ESSG_LRODESC_T" on page 1011	A structure containing information about the LRO to be updated.
ulOption;	ESSG_ULONG_T	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).

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);
```

Parameter	Data Type	Description
pulVersion	ESSG_PULONG_T	Pointer to the current version number of the Grid API.

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

16 C Grid API Examples

In This Chapter

C Grid API Example 1101
C Grid API Drill-Through Example 1105
ESSG_OP_MEMBERANDUNIQUENAME Example 1107
ESSG_DT_MEMBERwKEY Example 1109
BuildTable Example Function 1111
DisplayOutput Example Function 1112
FreeTwoDim Example Function 1114

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.

		Year	Product	Market
Actual	Sales	123.45		

The following code fragment shows how the data structures are setup and the function calls that are needed to perform the retrieval.

```
/* 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,
                        rRangeDateOut, ppDataOut);
        }
        if (sts == 0)
        {
                /* Interate 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

```
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);
```

```
            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.

```
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);
 }
}
```

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");
}
```

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;
```

```
   ppTable[1][2].usType = ESSG_DT_DOUBLE;
   ppTable[1][2].Value.dblData = 123.45;
   ppTable[1][3].usType = ESSG_DT_BLANK;
   ppTable[1][4].usType = ESSG_DT_BLANK;

   return (ppTable);
}
```

DisplayOutput Example Function

The following function examples call this example function:

```
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\n",
            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)
```

```
                printf(",");
            }
               printf("\n");
      }
}
```

FreeTwoDim Example Function

```
ESSG_VOID_T FreeTwoDim(ESSG_PPDATA_T ppDataToFree,
                       ESSG_ULONG_T ulRows)
{
   ESSG_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);
}
```

delete

17 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.

Error Code	Value
ESSG_ERR_INITREQUIRED	1100001
ESSG_ERR_CONNECTREQUIRED	1100002
ESSG_ERR_INVALIDHANDLE	1100003
ESSG_ERR_INVALIDGRID	1100004
ESSG_ERR_CANNOTINIT	1100005
ESSG_ERR_CANNOTCONNECT	1100006
ESSG_ERR_CANNOTCREATEGRID	1100007
ESSG_ERR_INVALIDVERSION	1100008
ESSG_ERR_CANNOTGETAPIINST	1100009
ESSG_ERR_CANNOTGETAPICTX	1100010
ESSG_ERR_INVALIDOPTION	1100011
ESSG_ERR_INVALIDRANGE	1100012
ESSG_ERR_INVALIDDATA	1100013

Error Code	Value
ESSG_ERR_INVALIDROWORCOLMAX	1100014
ESSG_ERR_NULLARGUMENT	1100015
ESSG_ERR_CELLSREQUIRED	1100016
ESSG_ERR_RANGEREQUIRED	1100017
ESSG_ERR_INVALIDACTION	1100018
ESSG_ERR_INVALIDGRIDOPTION	1100019
ESSG_ERR_INVALIDFUNCTION	1100020
ESSG_ERR_MEMORY	1100021
ESSG_ERR_INVALIDROW	1100022
ESSG_ERR_INVALIDCOLUMN	1100023
ESSG_ERR_INVALIDPARM	1100024
ESSG_ERR_INVALIDCSLVERSION	1100025
ESSG_ERR_RANGEOVERLAP	1100026
ESSG_ERR_OPERATIONFAILED	1100027
ESSG_ERR_CANNOTSETOPTION	1100028
ESSG_ERR_INVALIDOPTIONVALUE	1100029
ESSG_ERR_EMPTYARGUMENT	1100030
ESSG_ERR_INVALIDLROHANDLE	1100031
ESSG_ERR_NOLROSAVAILABLE	1100032
ESSG_ERR_INVALIDLROTYPE	1100033
ESSG_ERR_GCINITFAIL	1100034
ESSG_ERR_GCSETLOCALEFAIL	1100035

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

18 Using the Visual Basic Main API

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 Function 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.

C Declaration	Visual Basic Declaration	Variable to Use When Calling
`Pointer to string (LPSTR)`	`ByVal S As String`	Any String or Variant variable
`Pointer to integer (LPINT)`	`I As Integer`	Any Integer or Variant variable
`Pointer to long (LPDWORD)`	`L As Long`	Any Long or Variant variable

C Declaration	Visual Basic Declaration	Variable to Use When Calling
`Pointer to struct (such as, LPRECT)`	`S As Rect`	Any variable of that user-defined type
`Integer (INT, UINT, WORD, BOOL)`	`ByVal I As Integer`	Any Integer or Variant variable
`Handle (hWnd, hDC, hMenu, etc.)`	`ByVal h As Integer`	Any Integer or Variant variable
`Long (DWORD, LONG)`	`ByVal L As Long`	Any Long or Variant variable
`Pointer to integers`	`I As Integer`	The first element of the array, for example I(0)
`Pointer to void (void *)`	`As Any`	Any variable (use ByVal with strings)
`Void (function return value)`	`Sub procedure`	not applicable

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

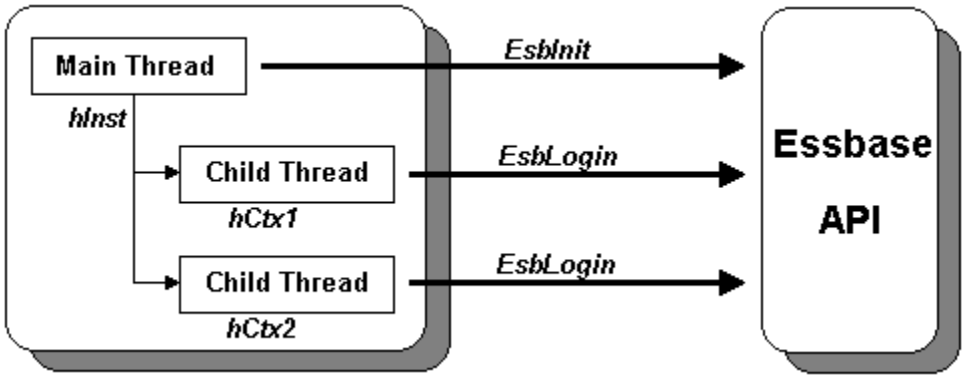
A single instance of an API program may make multiple calls to **EsbLogin**(), using the same user name or different user names on one or more Essbase Servers. Each call to **EsbLogin**() returns a unique context handle, and your program must keep track of each context handle returned. You may have up 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 alocal 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:

```
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
   Init.vbCallbackFuncAddress = GetProcAddress(AddressOf EsbErrorHandler)

   sts = EsbInit(Init, hInst)
   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
  ' [ YOUR CODE GOES HERE ]
  MsgBox " Info " & MsgNum & ": Level: " & Level & ": " & uLog & ": " & uMsg
End Function
```

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`.

```
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 calc operation has completed successfully or has been canceled. After **EsbGetProcessState**() 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 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.

might assist you in identifying and solving problems.

Table 8 Visual Basic Main API Common Problems and Solutions

Problem	Solution
Your program is generating protection faults.	If you are having this problem with a Visual Basic program, check the declared indirection level of any pointers being passed to the API.
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 **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. ● "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 **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, "Integrating Essbase With Your Product."
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 the header file `ESSERROR.H` to find the corresponding message text.

Drill-through Visual Basic API Example

```
Attribute VB_Name = "Module3"
Dim sts As Long
   Dim hInst As Long
   Dim hDestInst As Long
   Dim hCtx As Long
   Dim hDestCtx 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

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)
         Debug.Print "EsbGetMessage: sts = " & sts
       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"
   'Dim LevelName As String * ESB_MBRNAMELEN
   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

   'Next

      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
     Debug.Print "Access: " & UserDb.Access
   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
     Debug.Print "Access: " & UserDb.Access
   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
     Debug.Print "Access: " & UserDb.Access
   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
     Debug.Print "Access: " & UserDb.Access
   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
     Debug.Print "Access: " & UserDb.Access
   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
     Debug.Print "Access: " & UserDb.Access
   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" & _
   '          "Cola" & vbCrLf & "Utah" & _
   '          "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
```

```
   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
```

```
            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
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
```

```
   ESB_Term
End Sub
```

19 Visual Basic Main API Declarations

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:

Constant	Definition
ESB_USERNAMELEN	Maximum length of a user or group name
ESB_PASSWORDLEN	Maximum length of a user password
ESB_SVRNAMELEN	Maximum length of a server name
ESB_APPNAMELEN	Maximum length of an application name
ESB_DBNAMELEN	Maximum length of a database name

Constant	Definition
ESB_OBJNAMELEN	Maximum length of an object name
ESB_MBRNAMELEN	Maximum length of a member name
ESB_FTRNAMELEN	Maximum length of a filter name
ESB_ALIASNAMELEN	Maximum length of an alias table name
ESB_PATHLEN	Maximum length of a file path name
ESB_LINELEN	Maximum length of a line in a report
ESB_DESCLEN	Maximum length of application or database description

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.

Constant	Definition
ESB_DBREQFLAG_CALCDEF	Default flag for `DbReqFlags` field. Used the default calc script. Value: 1.
ESB_DBREQFLAG_CALCDSCR	Custom calc script flag for `DbReqFlags` field. Used a custom calc script. Value: 2.

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.

Constant	Definition
ESB_INDEXCACHEMIN_SIZE	Minimum index cache size for the *MaxMemIndex* field of the ESB_DBSTATE_T structure. Value: 1048576. No maximum value is defined.
ESB_INDEXPAGEMAX_SIZE	Maximum index page size for the *IndexPageSize* field of the ESB_DBSTATE_T structure. Value: 8192
ESB_INDEXPAGEMIN_SIZE	Minimum index page size for the *IndexPageSizeMin* field of the ESB_DBSTATE_T structure. Value: 1024

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.

Constant	Definition
ESB_TTYPE_NONE	No dimension type. Value for *DimTag* field of ESB_DIMENSIONINFO_T.

Constant	Definition
ESB_TTYPE_CATEGORY	Accounts: Currency ACCOUNTS tag. Value for *DimTag* type of ESB_DIMENSIONINFO_T.
ESB_TTYPE_CNAME	Country: Currency COUNTRY tag. Value for *DimTag* field of ESB_DIMENSIONINFO_T.
ESB_TTYPE_CTIME	Time: Currency TIME tag. Value for *DimTag* field of ESB_DIMENSIONINFO_T.
ESB_TTYPE_TYPE	Type: Currency TYPE tag. Value for *DimTag* field of ESB_DIMENSIONINFO_T.
ESB_TTYPE_PARTITION	Currency PARTITION tag. Value of *DimTag* field of ESB_DIMENSIONINFO_T.

Constant 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.

Constant	Value	Definition
ESB_LRODESCLEN_API	79	Maximum length of an object description
ESB_LRONOTELEN_API	599	Maximum length of a cell note
ESB_ONAMELEN_API	511	Length of an object name consisting of file name and path
ESB_STORE_OBJECT_API	&H0010	Value specifying to store a linked object on the server
ESB_NOSTORE_OBJECT_API	&H0001	Value specifying not to store a linked object on the server
ESB_LRO_OBJ_API	1	Value specifying to update only the linked object file
ESB_LRO_CATALOG_API	2	Value specifying to update only the object's catalog entry
ESB_LRO_BOTH_API	3	Value specifying to update both the object file and the catalog entry
ESB_LROTYPE_CELLNOTE_API	0	Value specifying that a linked object is a cell note
ESB_LROTYPE_WINAPP_API	1	Value specifying that a linked object is a Windows application
ESB_LROTYPE_URL_API	2	Value specifying that a linked object is a URL

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:

Data Type	Field	Description
Long	*cellOffset*	Cell offset within a data block
Double	*blkOffset*	Block offset
Double	*segment*	Segment number

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:

Data Type	Field	Description
Integer	*ObjType*	The object type
Integer	*status*	The catalog entry status
Long	*memCount*	The number of member names in the member combination identifying the data cell
ESB_LROHANDLE_API_T	*LinkID*	A link to the object's identification structure
Long	*updateDate*	The last date the object was modified
Integer	*accessLevel*	The access level for the data cell associated with the linked object
String * ESB_USERNAMELEN	*userName*	The name of the last user to modify the object
String	*memComb*	The member combination that identifies the data cell associated with the linked object
String * ESB_LRONOTELEN_API	*note*	A cell note, associated by union
ESB_LROINFO_API_T	*lroInfo*	The LRO information structure, associated by union

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:

Data Type	Field	Description
ESB_CELLADDR_API_T	*cellKey*	Cell address
Long	*hObject*	Internal object handle

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` save your changes on the server. The fields are described as follows:

Data Type	Field	Description
String * ESB_ONAMELEN_API	*objName*	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.
String * ESB_LRODESCLEN_API	*objDesc*	Description of an object linked to a data cell. ESB_LRODESCLEN_API specifies the maximum length of the description; the default value is 79.

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

Data Type	Essbase Type
typedef char	ESB_CHAR_T
typedef short	ESB_SHORT_T
typedef long	ESB_LONG_T
typedef unsigned char	ESB_UCHAR_T
typedef unsigned short	ESB_USHORT_T
typedef unsigned long	ESB_ULONG_T
typedef float	ESB_FLOAT_T
typedef double	ESB_DOUBLE_T

Table 10 Other Data Types

Data Type	Essbase Type	Description
typedef unsigned char	ESB_BOOL_T	boolean
typedef char	*ESB_STR_T	string (array of char)
typedef void	*ESB_HINST_T	API instance handle
typedef void	*ESB_HCTX_T	API context handle
typedef void	ESB_VOID_T	void
typedef size_t	ESB_SIZE_T	size of a memory block
typedef unsigned short	ESB_ACCESS_T	Essbase access level
typedef unsigned long	ESB_LOGINID_T	Essbase login id

Table 11 Pointer Types

Data Type	Essbase Type	Description
typedef char	*ESB_PCHAR_T	pointer to char
typedef short	*ESB_PSHORT_T	pointer to short
typedef long	*ESB_PLONG_T	pointer to long
typedef unsigned char	*ESB_PUCHAR_T	pointer to unsigned char
typedef unsigned short	*ESB_PUSHORT_T	pointer to unsigned short
typedef unsigned long	*ESB_PULONG_T	pointer to unsigned long
typedef float	*ESB_PFLOAT_T	pointer to float

Data Type	Essbase Type	Description
typedef double	*ESB_PDOUBLE_T	pointer to double
typedef ESB_BOOL_T	*ESB_PBOOL_T	pointer to boolean
typedef ESB_STR_T	*ESB_PSTR_T	pointer to string
typedef ESB_VOID_T	*ESB_PVOID_T	pointer to void
typedef ESB_SIZE_T	*ESB_PSIZE_T	pointer to size of a memory block
typedef ESB_HINST_T	*ESB_PHINST_T	pointer to VB API instance handle
typedef ESB_HCTX_T	*ESB_PHCTX_T	pointer to VB API context handle
typedef ESB_ACCESS_T	*ESB_PACCESS_T	pointer to security access level
typedef ESB_LOGINID_T	*ESB_PLOGINID_T	pointer to login id

Table 12 Miscellaneous Types

Data Type	Essbase Type	Description
typedef long	ESB_STS_T	return value from API functions
typedef ESB_STS_T	(*ESB_FUNC_T)()	pointer to function

Table 13 API Definitions

Constant	Value
#define ESB_TRUE	1
#define ESB_FALSE	0
#define ESB_NULL	NULL

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 14 Simple Data Types

Data Type	Essbase Type
As String * 1	ESB_CHAR_T
As Integer	ESB_SHORT_T
As Long	ESB_LONG_T
As String * 1	ESB_UCHAR_T
As Integer	ESB_USHORT_T
As Long	ESB_ULONG_T
As Long	ESB_FLOAT_T
As Long	ESB_DOUBLE_T
As Long	ESB_TIME_T
As Long	ESB_DATE_T

Table 15 Bitmask Data Types

Data Type	Essbase Type	Description
As Integer	ESB_ACCESS_T	Security access level. Possible bit values are: ● ESB_PRIV_NONE - 0x0000 - no privilege ● ESB_PRIV_READ - 0x0001 - read data ● ESB_PRIV_WRITE - 0x0002 - write data ● ESB_PRIV_CALC - 0x0004 - calculate data ● ESB_PRIV_DBLOAD - 0x0010 - load and unload databases ● ESB_PRIV_DBDESIGN - 0x0020 - design databases ● ESB_PRIV_DBCREATE - 0x0040 - create, delete, and edit databases ● ESB_PRIV_APPLOAD - 0x0100 - load and unload applications ● ESB_PRIV_APPDESIGN - 0x0200 - design applications ● ESB_PRIV_APPCREATE - 0x0400 - create, delete, and edit applications ● ESB_PRIV_USERCREATE - 0x1000 - create, delete, and edit users The access types are combinations of privileges. The valid values are: ● ESB_ACCESS_NONE - 0x0000 ● ESB_ACCESS_READ - 0x0111 ● ESB_ACCESS_WRITE - 0x0113 ● ESB_ACCESS_CALC - 0x0117 ● ESB_ACCESS_DBDESIGN - 0x0137 ● ESB_ACCESS_DBCREATE - 0x0177 ● ESB_ACCESS_APPDESIGN - 0x0377 ● ESB_ACCESS_APPCREATE - 0x0777 ● ESB_ACCESS_FILTER - 0x0110 ● ESB_ACCESS_DBALL - 0x00ff - full database access ● ESB_ACCESS_APPALL - 0x0fff - full application/database access ● ESB_ACCESS_SUPER - 0xffff - Administrator (unrestricted access)

Data Type	Essbase Type	Description
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

Data Type	Essbase Type	Description
ByVal As String * 1	*ESB_BOOL_T*	boolean
ByVal As String	**ESB_STR_T*	string (array of char)
ByVal As Long	**ESB_HINST_T*	API instance handle
ByVal As Long	**ESB_HCTX_T*	API context handle
As Any	*ESB_VOID_T*	void
ByVal As Long	*ESB_SIZE_T*	size of a memory block

Data Type	Essbase Type	Description
ByVal As Integer	*ESB_ACCESS_T*	Essbase access level
ByVal As Long	*ESB_LOGINID_T*	Essbase login id

Table 17 Pointer Types

Data Type	Essbase Type	Description
As Integer	**ESB_PSHORT_T*	pointer to short
As Long	**ESB_PLONG_T*	pointer to long
As Integer	**ESB_PUSHORT_T*	pointer to unsigned short
As Long	**ESB_PULONG_T*	pointer to unsigned long
As Long	**ESB_PFLOAT_T*	pointer to float
As Long	**ESB_PDOUBLE_T*	pointer to double
As Any	**ESB_PVOID_T*	pointer to void
As Long	**ESB_PSIZE_T*	pointer to size of a memory block
As Long	**ESB_PHINST_T*	pointer to VB API instance handle
As Long	**ESB_PHCTX_T*	pointer to VB API context handle
As Integer	**ESB_PACCESS_T*	pointer to security access level
As Long	**ESB_PLOGINID_T*	pointer to Essbase login id

Table 18 Miscellaneous Types

Data Type	Essbase Type	Description
ByVal As Long	*ESB_STS_T*	return value from API functions
As Long	*ESB_FUNC_T*	pointer to function

Table 19 Boolean Flags

Data Type	Essbase Type	Description
chr$(1)	ESB_TRUE	Boolean TRUE to be used in data structures
chr$(0)	ESB_FALSE	Boolean FALSE to be used in data structures
1	ESB_YES	YES flag to be used in the list of the VB API function parameters
0	ESB_NO	NO flag to be used in the list of the VB API function parameters
ByVal 0&	NULL	Null

Visual Basic API Attributes Terminology

Table 20 VB API Attributes Terminology

Term	Definition
Bucketing Type	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.
ESB_ATTRIBUTE_ DIMENSION and ESB_ATTRIBUTE_ MEMBER	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`.
ESB_ATTRIBUTED_ MEMBER	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`.
ESB_BASE_ DIMENSION and ESB_BASE_ MEMBER	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`.
ESB_STANDARD_ DIMENSION and ESB_STANDARD_ MEMBER	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`.
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

   AppName      As String * ESB_APPNAMELEN
   DbName       As String * ESB_DBNAMELEN
End Type
```

VB Data Type	Field	Description
As String * ESB_APPNAMELEN	*AppName*	Application name
As String * ESB_DBNAMELEN	*DbName*	Database name

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

   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
```

VB Data Type	Field	Description
As String * ESB_APPNAMELEN	*Name*	The application name
As String * ESB_SVRNAMELEN	*Server*	The server name

VB Data Type	Field	Description
As Integer	*Status*	Application load status. The values are as follows: ● ESB_STATUS_NOTLOADED ● ESB_STATUS_LOADING ● ESB_STATUS_LOADED ● ESB_STATUS_UNLOADING
As Integer	*AppType*	The type of application. Valid values are: ● 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. ● ESB_APP_NONUNICODE - 0x0002 - The program is a non-Unicode mode client program.
As Integer	*nConnects*	The number of users currently connected to the application
As Integer	*nDbs*	Number of databases in this application
As Long	*ElapsedAppTime*	Elapsed number of seconds since application loading
As Integer	*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
As String	*AppLocale*	The application locale description, of type ESB_LOCALESTRING_LENGTH.

ESB_APPINFOEX_T

This structure is used in **EsbGetApplicationInfoEx**(). The fields are:

```
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
```

VB Data Type	Field	Description
As String * ESB_ APPNAMELEN	*Name*	Application name

VB Data Type	Field	Description
As String * ESB_ SVRNAMELEN	*Server*	Network server name
As Integer	*AppType*	The type of application. Valid values are: ● 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. ● ESB_APP_NONUNICODE - 0x0002 - The program is a non-Unicode mode client program.
As String * ESB_ LOCALESTRING_ LENGTH	*AppLocale*	The application locale description, of type ESB_LOCALESTRING_LENGTH.
As Integer	*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
As Integer	*Status*	Application load status
As Integer	*nConnects*	Number of users connected
As Long	*ElapsedApp Time*	Elapsed application time: number of seconds the application has been loaded

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:

```
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
```

VB Data Type	Field	Description
As String * ESB_DESCLEN	*Description*	Application description (up to 80 characters)
As String * 1	*Loadable*	Flag to indicate whether the application can be loaded (ESB_TRUE if the application is loadable).
As String * 1	*Autoload*	Flag to indicate whether the application is loaded automatically when Essbase Server is started (ESB_TRUE if the application will be automatically loaded).
As Integer	*Access*	Default access to databases in the application (lowest possible level of access for all users). Values: ● ESB_PRIV_NONE ● ESB_PRIV_DBDESIGN ● ESB_PRIV_CALC ● ESB_PRIV_WRITE ● ESB_PRIV_READ
As String * 1	*Connects*	Flag to indicate whether users can connect to the application (ESB_TRUE if users can connect).
As String * 1	*Commands*	Flag to indicate whether users can issue commands to the application (ESB_TRUE if the application is accepting user commands).
As String * 1	*Updates*	Flag to indicate whether users can update data in the application (ESB_TRUE if the application is accepting user update commands).
As Long	*LockTimeout*	Time-out period (in seconds) after which block-level locks are automatically removed. This field does not apply to aggregate storage databases.
As Long	*lroSizeLimit*	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.
As String * 1	*Security*	Flag to indicate whether application security is enabled (ESB_TRUE if security is enabled).

ESB_ATTRIBUTEINFO_T

This structure has information on attributes.

```
Type ESB_ATTRIBUTEINFO_T
   MbrName   As String * ESB_MBRNAMELEN
   DimName   As String * ESB_MBRNAMELEN
   Attribute As Variant
End Type
```

VB Data Type	Field	Description
As String * ESB_MBRNAMELEN	*MbrName*	Attribute member name from "ESB_MEMBERINFO_T" on page 1182 or "ESB_MBRINFO_T" on page 1460, including a long name
As String * ESB_MBRNAMELEN	*DimName*	Attribute dimension name

VB Data Type	Field	Description
As Variant	*Attribute*	Attribute value

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.

```
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	Field	Description
As String * ESB_ MBRNAMELEN	*DefaultTrueString*	The string used with the boolean attribute to indicate TRUE. The default value is ESB_ DEFAULT_TRUESTRING ("True").
As String * ESB_ MBRNAMELEN	*DefaultFalseString*	The string used with the boolean attribute to indicate FALSE. The default value is ESB_ DEFAULT_FALSESTRING ("False").
As String * ESB_ MBRNAMELEN	*DefaultAttrCalcDimName*	The name of the attribute calculations (aggregate) dimension. The default value is ESB_ DEFAULT_ATTRIBUTECALCULATIONS ("Attribute Calculations").
As String * ESB_ MBRNAMELEN	*DefaultSumMbrName*	The name used with the attribute calculations (aggregate) dimension to indicate SUM. The default value is ESB_DEFAULT_SUM ("Sum").
As String * ESB_ MBRNAMELEN	*DefaultCountMbrName*	The name used with the attribute calculations (aggregate) dimension to indicate COUNT. The default value is ESB_DEFAULT_COUNT ("Count").
As String * ESB_ MBRNAMELEN	*DefaultAverageMbrName*	The name used with the attribute calculations (aggregate) dimension to indicate AVERAGE. The default value is ESB_DEFAULT_AVERAGE ("Average").
As String * ESB_ MBRNAMELEN	*DefaultMinMbrName*	The name used with the attribute calculations (aggregate) dimension to indicate MINIMUM. The default value is ESB_DEFAULT_MIN ("Min").
As String * ESB_ MBRNAMELEN	*DefaultMaxMbrName*	The name used with the attribute calculations (aggregate) dimension to indicate MAXIMUM. The default value is ESB_DEFAULT_MAX ("Max").

VB Data Type	Field	Description
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
```

VB Data Type	Field	Description
As String * ESB_ APPNAMELEN	*AppName*	Application name

VB Data Type	Field	Description
As String * ESB_ DBNAMELEN	*DbName*	Database Name
As String * ESB_ FILENAMELEN	*FilePath*	File path
As Long	*FileSize*	File size in bytes
As Long	*FileSequenceNum*	The 1-based sequence number of the file within the set of files of its *FileType* for the specified database
As Long	*FileCount*	Number of files of its *FileType* returned
As Integer	*FileType*	One of the following types of files: ● ESB_FILETYPE_INDEX ● ESB_FILETYPE_DATA
As Integer	*FileOpen*	Flag indicating whether the file is open: 0 if the file is closed, nonzero if the file is open

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:

```
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

VB Data Type	Field	Description
As long	*DataCacheSize*	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.
As long	*IndexCacheSize*	Run-time index cache size (in KB) currently in use by database
As long	*IndexPageSize*	Run-time index page size (in KB) currently in use by database
As Long	*nDims*	The number of dimensions in database
As Integer	*DbType*	Database type (normal or currency). This field can contain the following values : ● ESB_DBTYPE_NORMAL ● ESB_DBTYTPE_CURRENCY
As Integer	*Status*	The Ddatabase load status (loaded or not loaded) - one of: ● ESB_STATUS_NOTLOADED ● ESB_STATUS_LOADING ● ESB_STATUS_LOADED ● ESB_STATUS_UNLOADING
As Integer	*nConnects*	Number of users currently connected to the database
As Integer	*nLocks*	The Nnumber of data blocks currently exclusively locked
As Integer	*Data*	Flag to indicate the loading state of the data in 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: ● ESB_DBDATA_NONE /* no data */ ● ESB_DBDATA_LOADNOCALC /*data loaded without calc */ ● ESB_DBDATA_CLEAN /* data has been calculated */
As String * ESB_ APPNAMELEN	*AppName*	Associated application name
As String * ESB_ DBNAMELEN	*Name*	Database name
As String * ESB_ MBRNAMELEN	*Country*	The currency country dimension member, if any. If none, this field is an empty string.
As String * ESB_ MBRNAMELEN	*Time*	The currency time dimension member, if any. If none, this field is an empty string.
As String * ESB_ MBRNAMELEN	*Category*	The currency category dimension member, if any. If none, this field is an empty string.
As String * ESB_ MBRNAMELEN	*Type*	The currency type dimension member (currency databases only) . If none exists, this field is an empty string.
As String * ESB_ MBRNAMELEN	*CrPartition*	The currency partition member (non-currency databases only)

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:

```
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
```

Data Type	Field	Description
As Long	*DbReqType*	Type of database request
As Long	*DbReqFlags*	Bit map of information flags
As "ESB_TIMERECORD_T" on page 1191	*StartTimeRec*	Request to start time record
As "ESB_TIMERECORD_T" on page 1191	*EndTimeRec*	Request to end time record
As String * ESB_USERNAMELEN	*User*	User name

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.

```
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
```

```
   CalcTwoPass          As Integer
   CalcCreateBlock      As Integer
   CrDbName             As String * ESB_DBNAMELEN
   CrTypeMember         As String * ESB_MBRNAMELEN
   CrConvType           As Integer
   DataCompress         As Integer
   RetrievalBuffer      As Long
   RetrievalSortBuffer As Long
   TimeOut              As Long
   CommitBlocks         As Long
   CommitRows           As Long
   nVolumes             As Long
   DataCompressType     As Integer
   IsolationLevel       As Integer
   PreImage             As Integer
End Type
```

The fields are:

VB Data Type	Field	Description
As String * ESB_ DESCLEN	*Description*	Database description (up to 80 characters)
As Integer	*Loadable*	Flag to indicate whether the database may be loaded (ESB_TRUE if the database is loadable)
As Integer	*Autoload*	Flag to indicate whether the database will automatically be loaded when the application is started (ESB_TRUE if the database will be automatically loaded)
As Integer	*Access*	Default access level to the database. See Table 15 for a list of values this field can contain.
As Integer	*IndexType*	Database index type (array or tree). Values: ● ESB_INDEXTYPE_ARRAY ● ESB_INDEXTYPE_AVL **Note:** For API Releases 4 and later, the *IndexType* field is obsolete.
As Long	*MaxMem*	Maximum memory reserved for non-compressed data blocks in the database (in bytes)
As Long	*MaxCompMem*	Maximum memory reserved for compressed data blocks in database (in bytes)
As Long	*MaxMemIndex*	Minimum index cache size. Value: 1048576. Set using the constant ESB _ INDEXCACHEMIN_SIZE
As Long	*IndexPageSize*	Size of index page in which buffer pool is constructed in (in bytes). Minimum index page size. Value: 1024. Set using the constant ESB_INDEXPAGEMIN_SIZE Maximum page size for the IndexPageSize field. Value: 8192. Set using the constant ESB_ INDEXPAGEMAX_SIZE
As Integer	*CalcNoAgg Missing*	Flag to suppress aggregation of members if all their children are missing (ESB_TRUE if missing values are not aggregated)
As Integer	*CalcNoAvg Missing*	Flag to suppress inclusion of missing members in calculating averages (ESB_TRUE if missing values are not included)

VB Data Type	Field	Description
As Integer	*CalcTwoPass*	Flag to force two pass calculation when running full calculation of database (ESB_TRUE if two pass calculation is enabled)
As Integer	*CalcCreate Block*	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.
As String * ESB_ DBNAMELEN	*CrDbName*	The name of associated currency database (valid in non-currency databases)
As String * ESB_ MBRNAMELEN	*CrTypeMember*	The name of Currency Conversion type member (valid in non-currency databases)
As Integer	*CrConvType*	Currency Conversion type (whether currency conversions are calculated by multiplication or division). Values: ● ESB_CRCTYPE_DIV ● ESB_CRCTYPE_MULT
As Integer	*DataCompress*	Optional Flag to determine whether to compress blocks for this database.
As Long	*RetrievalBuffer*	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.
As Long	*RetrievalSortBuffer*	Specifies the size, in kilobytes, of the server buffer that holds the data to be sorted during a retrieval. The default is 10240 bytes.
As Long	*TimeOut*	The timeout interval in seconds. This can only be set for COMMITTED access. ● -1 is Indefinite wait. ● 0 is Immediate access, no wait (the default). ● *n* is the specified interval in seconds.
As Long	*CommitBlocks*	The number of data blocks modified before performing the explicit commit (only used when isolation level is UNCOMMITTED).
As Long	*CommitRows*	The number of rows of the input file to data load before performing the explicit commit (only used when isolation level is UNCOMMITTED).
As Long	*nVolumes*	The number of disk volume settings for this database.
As Integer	*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.
As Integer	*IsolationLevel*	The isolation level: ● COMMITTED - write locks on all affected data blocks restrict access until the transaction commits. ● UNCOMMITTED - (default) write locks are acquired and released as needed during the transaction.

VB Data Type	Field	Description
As Integer	*PreImage*	The flag to read previously committed data during read-only requests. This flag can only be set for COMMITTED access. The default is YES.

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 1171structure. The fields are:

```
Type ESB_DBSTATS_T

   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	Field	Description
As Integer	*IndexType*	The database index type (array or tree). This field can contain the following values: ● ESB_INDEXTYPE_ARRAY ● ESB_INDEXTYPE_AVL
As Long	*nDims*	Number of dimensions
As Long	*DeclaredBlockSize*	The declared data block size
As Long	*ActualBlockSize*	The actual data block size
As Double	*DeclaredMax Blocks*	The declared maximum number of blocks in the database

VB Data Type	Field	Description
As Double	*ActualMaxBlocks*	The actual maximum number of blocks in the database
As Double	*NonMissingLeaf Blocks*	The number of non-missing leaf (lowest level) blocks in the database
As Double	*NonMissingNon LeafBlocks*	The number of non-missing, non-leaf (upper level) blocks in the database
As Double	*NonMissing Blocks*	The total number of non-missing blocks in the database
As Double	*PagedOutBlocks*	The number of database blocks currently paged out to disk
As Double	*PagedInBlocks*	The total number of database blocks currently paged into memory
As Double	*TotalBlocks*	Total number of existing data blocks (not the maximum)
As Double	*NonExclusive LockCount*	The number of database blocks currently non-exclusively locked
As Double	*ExclusiveLock Count*	The number of database blocks currently exclusively locked
As Double	*TotMemBlocks*	The total memory used for all database blocks
As Double	*TotMemIndex*	The total memory used for the database index
As Double	*TotMemPagedIn Blocks*	The total memory used for all paged-in (uncompressed) database blocks
As Double	*BlockDensity*	The average database block density (calculated using all currently loaded blocks)
As Double	*SparseDensity*	Average density of the sparse dimensions in the database
As Double	*CompressionRatio*	Average data block compression ratio on the disk
As Double	*InMemCompBlocks*	The number of database blocks currently paged into compressed memory
As Double	*TotMemInMemCompBlocks*	The total memory used for database blocks currently paged into compressed memory
As Double	*ClusterRatio*	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.

ESB_DIMENSIONINFO_T

This structure is used in **EsbGetDimensionInfo**(). The fields are:

```
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
```

Data Type	Field	Description
As String * ESB_MBRNAMELEN	*DimName*	Dimension name
As Long	*DimNumber*	Dimension number of the member
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_TTYPE_CATEGORY ● ESB_TTYPE_CNAME ● ESB_TTYPE_CTIME ● ESB_TTYPE_NONE ● ESB_TTYPE_PARTITION ● ESB_TTYPE_TYPE
As Long	*DeclaredDimSize*	Declare dimension size
As Long	*ActualDim Size*	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:

```
Type ESB_DIMSTATS_T

   DeclaredDimSize   As Long
   ActualDimSize     As Long
   DimType           As Integer
   DimName           As String * ESB_MBRNAMELEN
End Type
```

VB Data Type	Field	Description
As String * ESB_ MBRNAMELEN	*DimName*	The dimension member name
As Integer	*DimType*	The dimension type (sparse or dense). This field can contain the following values: ● ESB_DIMTYPE_SPARSE ● ESB_DIMTYPE_DENSE
As Long	*DeclaredDimSize*	The declared dimension size (the number of members declared in the specified dimension, including any label only or shared members in that dimension)
As Long	*ActualDim Size*	The actual dimension size (the number of members in the specified dimension, excluding any label only or shared members in that dimension)

ESB_DURLINFO_T

A data structure used to capture URL information. The fields are:

```
Type ESB_DURLINFO_T
   bIsLevel0    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
```

Visual Basic Data Type	Field	Description
As Integer	*bIsLevel0*	If 1, then URL definition is restricted to level-0 data; if 0, there is no restriction
As Integer	*iURLXMLSize*	Size of URL XML
As String * 1024	*cpURLName*	Name of URL definition
As String * 8192	*cpURLXML*	Content of URL XML

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:

```
Type ESB_GLOBAL_T

   Security        As Integer
   Logins          As Integer
   Access          As Integer
   Validity        As Integer
   Currency        As Integer
   PwMin           As Integer
```

```
   InactivityTime  As Long
   InactivityCheck As Long
End Type
```

VB Data Type	Field	Description
As String * 1	*Security*	Flag to indicate whether global security is enabled (default is ESB_TRUE, security enabled)
As String * 1	*Logins*	Flag to indicate whether user logins are enabled (default is ESB_TRUE, indicating logins are enabled)
As Integer	*Access*	The default access level for newly-created applications (default is ESB_ACCESS_NONE). For a list of possible values, see Table 15.
As Integer	*Validity*	The default password validity period (default is 365 days)
As String * 1	*Currency*	Flag to indicate whether currency option is supported (this flag is read only). Set to ESB_TRUE if the currency option is enabled.
As Integer	*PwMin*	The minimum permitted password length (default is 6 characters)
As Long	*InactivityTime*	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.
As Long	*Inactivity Check*	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.

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
```

VB Data Type	Field	Description
As Long	*Version*	Version of Essbase API used to compile the application. Used for backward compatibility.
As Integer	*MaxHandles*	The maximum number of simultaneous context handles required by the application (between 1 and 10)
As String * ESB_PATHLEN	*LocalPath*	The default local path name to use for file and object operations on the client

VB Data Type	Field	Description
As String * ESB_PATHLEN	*MessageFile*	The qualified path name of the message database file, ESSBASE.MDB
As String * ESB_PATHLEN	*HelpFile*	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. See Chapter 3, "Integrating Essbase With Your Product." 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.
As String * 1	*ClientError*	ESB_FALSE to use a default error handler, ESB_TRUE to use EsbGetMessage to retrieve messages
As Integer	*ErrorStack*	A size of the message stack used by EsbGetMessage, default is 100
As Long	*vbCallbackFuncAddress*	AddressOf custom Visual Basic callback function. For more information, see "Visual Basic API Message Handling" on page 1123.

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
End Type
```

VB Data Type	Field	Description
As String * ESB_USERNAMELEN	*UserName*	The user name
As Integer	*nLocks*	The number of blocks exclusively locked by this user
As Integer	*Time*	The maximum time (in seconds) that blocks have been exclusively locked
As Long	*LoginId*	The user login identification tag

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
```

VB Data Type	Field	Description
As String * ESB_MBRNAMELEN	*MbrName*	The member name
As String * ESB_MBRNAMELEN	*AltName*	The associated alias name (ESB_MBRNAME_T)

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
```

VB Data Type	Field	Description
As String * ESB_USERNAMELEN	*User*	SQL database user name
As String * ESB_PASSWORDLEN	*Password*	SQL database user password

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
```

VB Data Type	Field	Description
As String * ESB_ MBRNAMELEN	*CrMbrName*	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
As String * ESB_ MBRNAMELEN	*MbrName*	The member name
As String * ESB_ MBRNAMELEN	*DimName*	The member's dimension name (ESB_MBRNAMELEN)
As String * ESB_ MBRNAMELEN	*ParentMbr Name*	The specified member's parent name or an empty string if the member has no parent
As String * ESB_ MBRNAMELEN	*ChildMbrName*	The specified member's first child member name
As String * ESB_ MBRNAMELEN	*PrevMbrName*	The specified member's previous sibling member name
As String * ESB_ MBRNAMELEN	*NextMbrName*	The specified member's next sibling member name
As String * ESB_ DESCLEN	*Description*	The member description
As Long	*MbrNumber*	The member number
As Long	*DimNumber*	The member's dimension number
As Integer	*Status*	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: ● ESB_MBRSTS_NOTSET ● ESB_MBRSTS_NEVER ● ESB_MBRSTS_LABEL ● ESB_MBRSTS_REFER ● ESB_MBRSTS_REFNME ● ESB_MBRSTS_SHARE ● ESB_MBRSTS_VIRTSTORE ● ESB_MBRSTS_VIRTNOSTORE
As Integer	*Level*	The member level number (zero-based), counting up from the lowest descendent of the specified member
As Integer	*Generation*	The member generation number (one-based), counting down from the specified member's dimension member
As Integer	*UnaryCalc*	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
As Integer	*MbrTagType*	A 16 bit mask for member's tagged types (masks are of the form ESB_ATYPE_xxx)
As String *1	*CurrConvert*	Currency Conversion. This field can contain the values ESB_TRUE and ESB_FALSE

VB 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: ● Boolean (True|False) ● Date ("09/19/2006") ● Double (3.14) ● String ("Hello") For any attribute member, but not an attribute dimension: ESB_ATTRMBRDT_NONE = Everything else including empty.
As Integer	*IsAttributed*	Indicates whether the member has attributes associated with it. (ESB_TRUE if attributes are associated.)

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
End Type
```

VB Data Type	Field	Description
As Long	*hCtx*	VB API context handle
As Long	*Type*	Object type. See Table 15, “Bitmask Data Types,” on page 1161for a list of object types.
As String * ESB_APPNAMELEN	*AppName*	Application name
As String * ESB_DBNAMELEN	*DbName*	Database name
As String * ESB_PATHLEN	*FileName*	Object's file name. A local file name when the following apply: ● *hCtx* is a local context handle ● *AppName* and *DbName* are empty strings ● *FileName* points to the full path name of a local file.

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:

```
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
```

VB Data Type	Field	Description
As String * ESB_OBJNAMELEN	*Name*	Object name
As Long	*Type*	Object type. See Table 15, “Bitmask Data Types,” on page 1161 for a list of object types.
As String * ESB_APPNAMELEN	*AppName*	Application name
As String * ESB_DBNAMELEN	*DbName*	Database name
As Long	*FileSize*	Object's allocated file size (in bytes)
As String * 1	*Locked*	Flag to indicate whether object is locked (ESB_TRUE indicates the object is locked)
As String * ESB_USERNAMELEN	*User*	Name of the user who has the object locked (if locked), otherwise undefined
As Long	*TimeStamp*	Date and time object was locked (if locked), otherwise undefined
As ESB_TIMERECORD_T	*Time Modified*	Date and time of last modification

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
```

Data Type	Field	Description
String	*HostName*	Host name.
String	*AppName*	Application name.
String	*DbName*	Database name.

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
```

Data Type	Field	Description
Integer	*usType*	One of the Operation Type constants listed below.
Integer	*usDirection*	One of the Direction constants listed below.
"ESB_PART_CONNECT_INFO_T" on page 1185	*HostDatabase*	The host server.

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
```

```
   DbName             As String * ESB_DBNAMELEN
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.
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		
Long	*LastRefreshTime*	Last time data at target was refreshed.
Integer	*AreaUpdatable*	Are changes allowed to replicated data?
The following fields only apply to replication data sources		
Integer	*IncrRefreshAllowed*	Can we refresh only the changed data?
Long	*LastUpdateTime*	Time of last change to data in the partition.
The following fields only apply to remote connections.		
String	*SvrName*	Host for the other side of the partition definition.
String	*AppName*	Application for the other side of the partition definition.
String	*DbName*	Database for other side of the partition definition; meta data change information.

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)
```

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

   AreaCount    As Long
   UpdatedOnly  As Integer
End Type
```

Data Type	Field	Description
Long	*AreaCount*	Number of partitions to refresh from (-1 == ALL)
Integer (Boolean)	*UpdatedOnly*	Refreshes only the cells modified at the source since the last refresh operation.
"ESB_PART_CONNECT_INFO_T" on page 1185	*pHostDatabase*	Array of partition specifications.

ESB_PARTOTL_QRY_FILTER_T

This structure further defines the meta data retrieval criteria.

```
Type ESB_PARTOTL_QRY_FILTER_T

   TimeStamp           As Long
   DimFilter           As Long
   MbrFilter           As Long
   MbrAttrFilter       As Long
End Type
```

Data Type	Field	Description
Long	*TimeStamp*	Query meta change happens after this time.
Long	*DimFilter*	Bitfield to select dimension changes.
Long	*MbrFilter*	Bitfield to select member changes.
Long	*MbrAttrFilter*	Bitfield to select member attribute changes.

Member Attribute Change Constants (MbrAttrFilter)

```
#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_PARTITION_OTLMBRATTR_ALIAS |
                                                    ESB_PARTITION_OTLMBRATTR_UCALC |
                                                    ESB_PARTITION_OTLMBRATTR_ATYPE |
                                                    ESB_PARTITION_OTLMBRATTR_CCONVERT |
                                                    ESB_MBRATTR_CRMBR_NAME |
                                                    ESB_PARTITION_OTLMBRATTR_UDA |
                                                    ESB_PARTITION_OTLMBRATTR_CALC |
                                                    ESB_PARTITION_OTLMBRATTR_LEVEL |
                                                    ESB_PARTITION_OTLMBRATTR_GENERATION)
#define 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
End Type
```

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

```
#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)
```

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_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
End Type
```

VB Data Type	Field	Description
As Integer	*Action*	Current process action (not used)

VB Data Type	Field	Description
As Integer	*State*	Current process state (either done or in progress). Values: ● ESB_STATE_DONE (0) ● ESB_STATE_INPROGRESS (1) ● ESB_STATE_FINALSTAGE (5)
As Integer	*Reserved1*	Reserved for future use
As Long	*Reserved2*	Reserved for future use
As Long	*Reserved3*	Reserved for future use

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
```

VB Data Type	Field	Description
As String * ESB_MBRNAMELEN	*MbrName*	The Member name (ESB_MBRNAMELEN)
As String * ESB_MBRNAMELEN	*RateMbr*	Array of rate member name (ESB_MBRNAME_T)

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
```

VB Data Type	Field	Description
As Long	*TimeValue*	Time value in seconds after 1/1/70

VB Data Type	Field	Description
As Integer	*Seconds*	Seconds after the minute. Values: 0-59.
As Integer	*Minutes*	Minutes after the hour. Values: 0-59.
As Integer	*Hours*	Hours since midnight. Values: 0-23.
As Integer	*Day*	Day of the month. Values: 1-31.
As Integer	*Month*	Months since January. Values: 0-11. January = 0.
As Integer	*Year*	Years since 1990.
As Integer	*Weekday*	Days since Sunday. Values: 0-6. Sunday = 0.

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:

```
Type ESB_USERAPP_T

   Access       As Integer
   MaxAccess    As Integer
   userName     As String * ESB_USERNAMELEN
   AppName      As String * ESB_APPNAMELEN
End Type
```

VB Data Type	Field	Description
As String * ESB_USERNAMELEN	*UserName*	The user or group name (ESB_USERNAMELEN)
As String* ESB_APPNAMELEN	*AppName*	The application name (ESB_APPNAMELEN)
As Integer	*Access*	The assigned access privilege to the application for the user or group. Values: ● ESB_PRIV_NONE ● ESB_PRIV_APPLOAD ● ESB_PRIV_APPDESIGN
As Integer	*MaxAccess*	The maximum access privilege to the application for the user or group from all sources

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:

```
Type ESB_USERDB_T

   Access       As Integer
```

```
   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
```

VB Data Type	Field	Description
As String * ESB_USERNAMELEN	*UserName*	User or group name (ESB_USERNAMELEN)
As String* ESB_APPNAMELEN	*AppName*	Application name (ESB_APPNAMELEN)
As String * ESB_DBNAMELEN	*DbName*	Database name (ESB_DBNAMELEN)
As Integer	*Access*	The assigned access privilege to the database for the user or group. Values: ● ESB_PRIV_NONE ● ESB_PRIV_READ ● ESB_PRIV_WRITE ● ESB_PRIV_CALC ● ESB_PRIV_DBLOAD ● ESB_PRIV_DBDESIGN These values are a subset of the Table 15.
As Integer	*MaxAccess*	Maximum access privilege to the database for the user or group from all sources
As String * ESB_FTRNAMELEN	*FilterName*	Name of the assigned database filter, if any. If none, this field is an empty string.

ESB_USERINFO_T, ESB_GROUPINFO_T

This structure stores information about users or groups.

```
Type ESB_USERINFO_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
   Description   As String * ESB_DESCLEN
   EMailID       As String * ESB_DESCLEN
   LockedOut     As Boolean
   PwdChgNow     As Boolean
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:

VB Data Type	Field	Description
As Long	*LastLogin*	Date of the user's last successful login stated as Greenwich Mean Time (users only).
As Long	*DbConnectTime*	Local (server) time of database connection. Read-only. Cannot be set by EsbSetUser.
As Long	*LoginId*	User login identification tag (users only).
As Integer	*Login*	Flag to indicate whether logged in (users only).
As Integer	*Type*	The type of the structure (user or group). Values: ● ESB_TYPE_USER ● ESB_TYPE_GROUP
As Integer	*Access*	User or group assigned default access privileges. This field can take any combination of the following bit values: ● ESB_ACCESS_SUPER /*Administrator, all bits set */ ● ESB_PRIV_APPCREATE ● ESB_PRIV_USERCREATE
As Integer	*MaxAccess*	User's maximum access privileges (users only). This combines individual access, and access levels conferred by group membership.
As Integer	*Expiration*	Reserved for future use.
As Integer	*FailCount*	Count of the failed login attempts since the last successful login (users only).
As String * ESB_ USERNAMELEN	*Name*	User or group name (ESB_USERNAMELEN).
As String* ESB_ APPNAMELEN	*AppName*	Name of the currently connected application (if applicable) (ESB_APPNAMELEN).
As String * ESB_ DBNAMELEN	*DbName*	Name of the currently connected database (if applicable) (ESB_DBNAMELEN).
As String	*Description*	User/group description (ESB_DESCLEN). For Future Use. Not settable by user.
As String	*EMailID*	User/group email address (ESB_DESCLEN). For Future Use. Not settable by user.
As Boolean	*LockedOut*	Flag that user is locked out.
As Boolean	*PwdChgNow*	Flag that user must change password.

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:

VB Data Type	Field	Description
As Long	*LastLogin*	Date of the user's last successful login stated as Greenwich Mean Time (users only).
As Long	*DbConnectTime*	Local (server) time of database connection. Read-only. Cannot be set by EsbSetUser.
As Long	*LoginId*	User login identification tag (users only).
As Integer	*Login*	Flag to indicate whether logged in (users only).
As Integer	*Type*	The type of the structure (user or group). Values: ● ESB_TYPE_USER ● ESB_TYPE_GROUP
As Integer	*Access*	User or group assigned default access privileges. This field can take any combination of the following bit values: ● ESB_ACCESS_SUPER /*Administrator, all bits set */ ● ESB_PRIV_APPCREATE ● ESB_PRIV_USERCREATE
As Integer	*MaxAccess*	User's maximum access privileges (users only). This combines individual access, and access levels conferred by group membership.
As Integer	*Expiration*	Reserved for future use.

VB Data Type	Field	Description
As Integer	*FailCount*	Count of the failed login attempts since the last successful login (users only).
As String * ESB_USERNAMELEN	*Name*	User or group name (ESB_USERNAMELEN).
As String* ESB_APPNAMELEN	*AppName*	Name of the currently connected application (if applicable) (ESB_APPNAMELEN).
As String * ESB_DBNAMELEN	*DbName*	Name of the currently connected database (if applicable) (ESB_DBNAMELEN).
As String * ESB_PASSWORDLEN	*Password*	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.
As String	*Description*	User/group description (ESB_DESCLEN). For Future Use. Not settable by user.
As String	*EMailID*	User/group email address (ESB_DESCLEN). For Future Use. Not settable by user.
As Integer	*LockedOut*	Flag that user is locked out.
As Integer	*PwdChgNow*	Flag that user must change password.
As String * ESB_ PROTOCOLNAMELEN	*protocol*	The External authentication protocol.
As String * ESB_CONNPARAMLEN	*connparam*	The External authentication connection.

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

    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
```

VB Data Type	Field	Description
ESB_SVRNAME_T	*Server*	Name of server where variable is defined (optional)

VB Data Type	Field	Description
ESB_APPNAME_T	*AppName*	Name of application to restrict variable to
ESB_DBNAME_T	*DbName*	Name of database to restrict variable to. If used, it requires that application be set.
ESB_MBRNAME_T	*VarName*	Name of substitution variable.
ESB_CHAR_T	*VarValue[256]*	Value of substitution variable.

20 Visual Basic Main API Functions

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.

Function	Description
EsbListAliases	Lists all the alias tables in the active database.
EsbLoadAlias	Loads an alias table for the active database from a structured text file
EsbGetAlias	Gets the active alias table name from the active database for a user.
EsbSetAlias	Sets the active alias table in the active database for a user.
EsbDisplayAlias	Dumps the contents of an alias table in the active database.
EsbRemoveAlias	Removes an alias table from the active database.
EsbClearAliases	Clears all alias tables for the active database.

VB Main API Application Functions

The application functions create new applications and modify, copy, get information about and otherwise manage existing applications.

Function	Description
EsbGetActive	Gets the names of the caller's current active application and database.
EsbSetActive	Sets the callers active application and database.
EsbClearActive	Clears the user's current active application and database.
EsbListApplications	Lists all applications which are accessible to the caller.
EsbCreateApplication	Creates a new application, either on the client or the server.
EsbCreateStorageTypedApplication	Creates a new application with the option of Multi-Dimensional or Aggregate Storage.
EsbDeleteApplication	Deletes an existing application, either on the client or the server.
EsbRenameApplication	Renames an existing application, either on the client or the server.
EsbGetApplicationInfoEx	Gets information from one or more applications.
EsbCopyApplication	Copies an existing application, either on the client or the server, to a new application, including all associated databases and objects.
EsbGetApplicationState	Gets an application state structure, which contains user-configurable parameters for the application.
EsbSetApplicationState	Sets user-configurable parameters for the application using the application's state structure.
EsbGetApplicationInfo	Gets an application's information structure, which contains non-user-configurable parameters for the application.
EsbLoadApplication	Starts an application on the server.

Function	Description
EsbUnloadApplication	Stops an application on the server.

VB Main API Attributes Functions

These Visual Basic Main functions are for attributes.

Function	Description
EsbCheckAttributes	Returns the attribute type for given attribute dimensions, base dimensions, attribute members, and base members
EsbGetAssociatedAttributesInfo	Returns the attribute members associated with a given base member
EsbGetAttributeInfo	Returns attribute information for a given attribute member or dimension
EsbGetAttributeSpecifications	Retrieves attribute specifications for the outline

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.

Function	Description
EsbClearDatabase	Clears all loaded data in the active database.
EsbCopyDatabase	Copies an existing database, either on the client or the server, to a new database, including all associated databases and objects.
EsbCreateDatabase	Creates a new database within an application, either on the client or the server.
EsbDeleteDatabase	Deletes an existing database from an application, either on the client or the server.
EsbGetCurrencyRateInfo	Gets a list of structures containing rate information for all members of the tagged currency partition dimension in the active database outline.
EsbGetDatabaseInfo	Gets a database's information structure, which contains non user-configurable parameters for the database.
EsbGetDatabaseNote	Gets a database's note-of-the-day message.
EsbGetDatabaseState	Gets a database's state structure, which contains user-configurable parameters for the database.
EsbGetDatabaseStats	Gets the active database's stats structure, which contains statistical information about the database.
EsbListCurrencyDatabases	Lists all currency databases within a specific application which are accessible to the caller.

Function	Description
EsbListDatabases	Lists all databases which are accessible to the caller, either within a specific application, or on an entire server.
EsbLoadDatabase	Starts a database within an application on the server.
EsbRenameDatabase	Renames an existing database within an application, either on the client or the server.
EsbSetDatabaseNote	Sets a database's note-of-the-day message.
EsbSetDatabaseState	Sets user-configurable parameters for the database using the database's state structure.
EsbUnloadDatabase	Stops a database within an application on the server.
EsbValidateDB	Checks the database for data integrity.

VB Main API Database Member Functions

These functions obtain information about database members and build database dimensions.

Function	Description
EsbQueryDatabaseMembers	Performs a report-style query to list a selection of database member information.
EsbCheckMemberName	Checks if a string is a valid member name within the active database outline.
EsbGetMemberInfo	Gets a structure containing information about a specific member in the active database outline.
EsbGetMemberCalc	Gets the calc equation for a specific member in the active database outline.
EsbGetDimensionInfo	Gets dimension information.
EsbBuildDimension	Allows the creation of a dimension in the active database from a data file and rules file.
EsbBuildDimFile	This function builds a data file to be used in the addition or removal of members from the outline in the active database.
EsbBuildDimStart	This function starts the process of the adding or removing members from the outline in the active database.

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.

Function	Description
EsbCreateDrillThruURL	Creates a drill-through URL, with the given link and the name, within the active database outline.
EsbDeleteDrillThruURL	Deletes a drill-through URL, with the given URL name, within the active database outline.

Function	Description
EsbGetCellDrillThruReports	Gets the drill-through reports associated with a data cell as a list of URL XMLs, given the cell's member combination.
EsbGetDrillThruURL	Gets a list of drill-through URL names within the active database outline.
EsbListDrillThruURLs	Lists the drill-through URLs within the active database outline.
EsbUpdateDrillThruURL	Updates a drill-through URL, with the given name, within the active database outline.

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.

Function	Description
EsbArchiveBegin	Prepares database for archive by setting READ-ONLY status.
EsbArchiveEnd	After archive, returns database status to READ-WRITE.
EsbCalcFile	Executes a calc script against the active database from a file.
EsbExport	Allows the exporting of data from the current database to a text file.
EsbImport	Allows the importing of data from text files and other sources to the current database.
EsbListDbFiles	Retrieves information on specified index and data files
EsbReportFile	Sends a report specification to the active database from a file.
EsbSetDefaultCalcFile	Sets the default calc script for the active database from a calc script file.
EsbUpdateFile	Sends an update specification to the active database from a file.

VB Main API Group Administration Functions

These functions create groups, set and modify group attributes, and obtain information about existing groups.

Function	Description
EsbListGroups	Lists all groups who have access to a particular Essbase Server.
EsbCreateGroup	Creates a new group.
EsbDeleteGroup	Deletes an existing group.
EsbRenameGroup	Renames an existing group.

Function	Description
EsbGetGroup	Gets a group information structure, which contains security information for the group.
EsbSetGroup	Sets a group information structure.
EsbGetGroupList	Gets the list of users who are members of a group (or the list of groups to which a user belongs).
EsbSetGroupList	Sets the list of users who are members of a group.
EsbAddToGroup	Adds a user to a list of group members.
EsbDeleteFromGroup	Removes a user from a list of group members

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.

Function	Description
EsbAutoLogin	Displays a dialog box which allows the user to log in to an Essbase Server, and optionally selects an active application and database.
EsbCreateLocalContext	Creates a local API context for use in local API operations
EsbDeleteLocalContext	Releases a local context previously created by EsbCreateLocalContext()
EsbGetAPIVersion	Gets version number of the API DLL in use
EsbGetVersion	Gets the full version number of the connected Essbase Server.
EsbInit	Initializes the API and message database.
EsbLogin	Logs a user in to the Essbase Server.
EsbLoginSetPassword	Logs in a user, and changes the password.
EsbLogout	Logs a user out from an Essbase Server.
EsbLogoutUser	Allows an Administrator or Application Manager to disconnect another user from an Essbase Server.
EsbShutdownServer	Allows an Administrator to remotely stop the Agent.
EsbTerm	Terminates the API and releases all system resources used by the API.
EsbValidateHCtx	Validates a specific API context handle (hCtx).

VB Main API LRO Functions

These functions create, retrieve and delete LROs and return information about them.

Function	Description
EsbLROAddObject	Links a reporting object to a data cell in an Essbase database.
EsbLRODeleteCellObjects	Deletes all objects linked to a given data cell in an Essbase database.
EsbLRODeleteObject	Deletes a specific object linked to a data cell in an Essbase database.
EsbLROGetCatalog	Retrieves a list of LRO catalog entries for a given data cell in an Essbase database.
EsbLROGetMemberCombo	Retrieves the n'th member from the member combination list of the current LRO.
EsbLROGetObject	Retrieves an object linked to a data cell in an Essbase database.
EsbLROListObjects	Retrieves a list of all objects linked to cells in the active database for a given user name and/or modification date.
EsbLROPurgeObjects	Deletes all objects linked to cells in the active database for a given user name and/or modification date.
EsbLROUpdateObject	Stores an updated version of an LRO on the server.

VB Main API Location Aliases Functions

These functions create, delete and list location aliases.

Function	Description
EsbCreateLocationAlias	Maps an alias name to the host name, application name, database name, user login name, and user password
EsbDeleteLocationAlias	Deletes an existing location alias
EsbGetLocationAliasList	Returns all location aliases and the names to which the location aliases are mapped

VB Main API Miscellaneous Functions

These functions manage asynchronous processes, obtain state information, handle log files, and retrieve messages.

Function	Description
EsbGetProcessState	Gets the current state of an asynchronous process, such as a calculate or a data import.
EsbCancelProcess	Cancels an asynchronous process which has not yet completed.
EsbGetLogFile	Copies all or part of an application log file from the server to the client.
EsbDeleteLogFile	Deletes an application log file on the server.
EsbGetGlobalState	Gets the server global state structure which contains parameters for system administration.

Function	Description
EsbSetGlobalState	Sets the server global state structure which contains parameters for system administration.
EsbGetNextItem	Gets the server global state structure which contains system administration parameters.
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 initialization.
EsbSetPath	Sets the ARBORPATH environment variable for the current process.

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.

Function	Description
EsbGetLocalPath	Gets the full local file for an object file on the client.
EsbListObjects	Lists all objects of types specified.
EsbGetObjectInfo	Gets information about a specified object.
EsbGetObject	Copies an object from the server to a local file, and optionally locks it.
EsbPutObject	Copies an object from a local file to the server, and optionally unlocks it.
EsbLockObject	Locks an object on the server to prevent other users from updating it.
EsbUnlockObject	Unlocks a locked object on the server.
EsbCreateObject	Creates a new object.
EsbDeleteObject	Deletes an existing object.
EsbRenameObject	Renames an existing object.
EsbCopyObject	Copies an object.

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.

Function	Description
EsbBeginCalc	Starts sending a calc script and optionally executes it against the active database.
EsbBeginReport	Starts sending a report specification to the active database.
EsbBeginUpdate	Starts sending an update specification to the active database.

Function	Description
EsbCalc	Sends and optionally executes a calc script against the active database as a single string.
EsbDefaultCalc	Executes the default calculation for the active database.
EsbEndCalc	Marks the end of a calc script being sent to the active database.
EsbEndReport	Marks the end of a report specification being sent to the active database.
EsbEndUpdate	Marks the end of an update specification being sent to the active database.
EsbGetDefaultCalc	Gets the default calc script for the active database.
EsbGetString	Gets a string of data from the active database.
EsbGetStringBuf	Gets data from the active database until it returns all available data or until the caller's buffer is full.
EsbReport	Sends a report specification to the active database as a single string.
EsbSendString	Sends a string of data to the active database.
EsbSetDefaultCalc	Sets the default calc script for a database.
EsbUpdate	Sends an update to the active database as a single string.

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.

Function	Description
EsbListFilters	Lists all filters for a database.
EsbGetFilter	Starts getting the contents of a filter.
EsbGetFilterRow	Gets the next row of a filter.
EsbSetFilter	Starts setting the contents of a filter.
EsbSetFilterRow	Gets the next row of a filter.
EsbGetFilterList	Gets the list of users who are assigned a filter.
EsbSetFilterList	Sets the list of users who are assigned a filter.
EsbDeleteFilter	Deletes an existing filter.
EsbRenameFilter	Renames an existing filter.
EsbCopyFilter	Copies an existing filter.
EsbVerifyFilter	Verifies the syntax of a series of filter row strings against a specified database.

Function	Description
EsbVerifyFilterRow	Verifies the syntax of a single filter row string against a specified database.

VB Main API Substitution Variables Functions

These functions create, retrieve and delete substitution variables and return information about them.

Function	Description
EsbCreateVariable	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.
EsbDeleteVariable	This function deletes a substitution variable.
EsbGetVariable	This function retrieves the value of a substitution variable.
EsbGetVariable	This function lists all substitution variables that conform to the input criteria.

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.

Function	Description
EsbListUsers	Lists all users who have access to a particular Essbase Server.
EsbCreateUser	Creates a new user.
EsbDeleteUser	Deletes an existing user.
EsbRenameUser	Renames an existing user.
EsbGetUser	Gets a user information structure, which contains security information for the user.
EsbSetUser	Sets a user information structure, which contains security information for the user.
EsbResetUser	Resets the user's security structure to its initial state.
EsbSetPassword	Sets a user's password, erasing the existing password.
EsbGetApplicationAccess	Gets a list of user application access structures, which contain information about user access to applications.
EsbSetApplicationAccess	Sets a list of user application access structures.
EsbGetDatabaseAccess	Gets a list of user database access structures.
EsbSetDatabaseAccess	Sets a list of user database access structures.

Function	Description
EsbGetCalcList	Gets the list of calc scripts objects accessible to the user.
EsbSetCalcList	Sets the list of calc script objects which are available to a user.
EsbListConnections	Lists all users who are connected to the current application and database.
EsbListLocks	Lists all users who are connected to a specific application and database.
EsbRemoveLocks	Removes all data block locks on a database which are currently held by a user.

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

```
EsbAddToGroup (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

```
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)
ByVal hCtx      As Long
ByVal AppName   As String
ByVal DbName    As String
ByVal FileName  As String
```

Parameter	Description
hCtx	API context handle.
AppName	Name of application to archive.

Parameter	Description
DbName	Name of database to archive.
FileName	Name of file to contain archive information.

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

```
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.
```

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
```

```
ByVal AppName As String
ByVal DbName  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, ByVal AppName As
String, ByVal DbName As String) As Long

Sub ESB_ArchiveEnd()
   Dim sts As Long
   Dim AppName 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?clustername=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?clustername=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 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

```
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
```

Parameter	Description
hCtx	API context handle.
Store	Controls storage of data. If TRUE, data is stored in the server; if FALSE, no data is stored.
Unlock	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.
abortOnError	If TRUE, data load stops on the first error. Otherwise, data load continues.
pRules	Pointer to the rules file object definition structure.

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

```
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 EsbImportfor 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 EsbBuild   Dimension 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
hCtx	API context handle.

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

```
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

```
EsbCalc (hCtx, isCalculate, cscQuery)
ByVal hCtx         As Long
ByVal isCalculate  As Integer
ByVal cscQuery     As String
```

Parameter	Description
hCtx	VB API context handle.

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

```
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

- EsbSetDefaultCalc

EsbCalcFile

Executes a calc script against the active database from a file.

Syntax

```
EsbCalcFile (hDestCtx, hSrcCtx, AppName, DbName, FileName, isCalculate)
ByVal hDestCtx    As Long
ByVal hSrcCtx     As Long
ByVal AppName     As String
ByVal DbName      As String
ByVal FileName    As String
ByVal isCalculate As Integer
```

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

```
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)
ByVal hCtx           As Long
ByVal Count          As Integer
      AttrNameArray() As String
      AttrTypeArray   As Variant
```

Parameter	Description
hCtx	Context handle
Count	Number of given dimensions and members
AttrNameArray()	An array of names of given dimensions and members
AttrTypeArray	One of the following constant identifiers for the attribute type array: • ESB_ATTRIBUTE_DIMENSION • ESB_ATTRIBUTE_MEMBER • ESB_STANDARD_DIMENSION • ESB_STANDARD_MEMBER • ESB_BASE_DIMENSION • ESB_BASE_MEMBER • ESB_ATTRIBUTED_MEMBER • ESB_INVALID_MEMBER

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 ESB_ATTRIBUTED_MEMBER
                         Out MbrNameArr(index) & " is of type  ESB_ATTRIBUTED_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

```
EsbCheckMemberName (hCtx, MemName, isOk)
ByVal hCtx    As Long
ByVal 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

```
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
       Print "Valid Member Name"
     End If
End Sub
```

See Also

- EsbGetMemberInfo
- EsbQueryDatabaseMembers
- EsbVerifyFilter
- EsbSetActive

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

```
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
hCtx	VB API context handle.
hSrcCtx	Not used; should be same as *hCtx*.
AppName	Name of an existing application to copy.
nAppName	Name of a new application. See "Application Name Limits" on page 1727.

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.

Parameter	Description
nAppName	Name of a destination application.
DbName	Name of an existing database to copy.
nDbName	Name of a new database. See "Database Name Limits" on page 1728.

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

```
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)
ByVal hCtx     As Long
ByVal hSrcCtx  As Long
ByVal AppName  As String
ByVal nAppName As String
ByVal DbName   As String
ByVal nDbName  As String
ByVal FltName  As String
ByVal nFltName As String
```

Parameter	Description
hCtx	VB API context handle.
hSrcCtx	Not used - should be same as *hCtx*.
AppName	Source application name.
nAppName	Destination application name.
DbName	Source database name.
nDbName	Destination database name.
FltName	Source name of an existing filter to be copied.
nFltName	Destination name of the copied filter. See "Filter Name Limits" on page 1728.

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 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

```
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

```
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

```
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
```

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

```
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

```
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
```

```
   '*****************
   sts = EsbCreateDatabase (hCtx, AppName, DbName,
   ESB_DBTYPE_NORMAL)
End Sub
```

See Also

- EsbCreateApplication
- EsbCreateObject

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"
```

```
   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
```

Parameter	Description
hCtx	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.
SecurityProvider	The name of the external authentication mechanism.
ProviderConnectionParameters	Parameters used by the external authentication mechanism, if any.

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,
ByVal AppName 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
```

Parameter	Description
hCtx	VB API context handle.
GrpName	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

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, 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

  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

```
EsbCreateObject (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 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

```
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
   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    '********************
   ' 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

```
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

```
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

```
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

- EsbRenameUser
- EsbSetPassword
- EsbSetUser

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" Susbtitution 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

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
   '***********************************
```

```
   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"
```

```
   '********************
   ' Delete Application
   '********************
   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"
   '*****************
```

```
   ' 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)
ByVal hCtx       As Long
ByVal AliasName As String
```

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
```

```
  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 *essbase.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
```

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

```
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" Susbtitution 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

```
EsbDisplayAlias (hCtx, AltName, pItems)
ByVal hCtx    As Long
ByVal AltName As String
      pItems  As Integer
```

Parameter	Description
hCtx	VB API context handle.
AltName	Name of alias table.
pItems	Address of variable to receive Items of aliases.

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)   For n = 1 To Items     '****************************
     ' 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

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

```
Declare 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*)
ByVal *hCtx* As Long

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

```
Declare Function EsbEndUpdate Lib "ESBAPIN" (ByVal hCtx As Long) As Long
```

See the example for EsbBeginUpdate.

See Also

- EsbBeginUpdate

- EsbSendString
- EsbUpdate

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
hCtx	VB API context handle.
AppName	Buffer to receive an application name string.
szApp	Size of an application name string buffer.
DbName	Buffer to receive a database name string.
szDb	Size of the database name string buffer.

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

```
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

```
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
ByVal AppName 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. If *AppName* 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

```
EsbGetApplicationInfo (hCtx, AppName, pAppInfo, pItems)
ByVal hCtx     As Long
ByVal AppName  As String
ByVal pAppInfo As ESB_APPINFO_T
      pItems   As Integer
```

Parameter	Description
hCtx	VB API context handle (logged in).
AppName	Application name. Required; cannot by NULL.
pAppInfo	Buffer to receive an application info structure.
pItems	Address of variable to receive Items of returned databases.

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

```
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)
```

```
   Next
End Sub
```

See Also

- EsbGetApplicationInfoEx
- EsbGetApplicationState
- EsbGetDatabaseInfo
- EsbGetNextItem

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

```
EsbGetApplicationState (hCtx, AppName, pAppState)
ByVal hCtx      As Long
ByVal AppName   As String
      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

```
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

- EsbOtlAssociateAttributeMember
- EsbOtlDisassociateAttributeDimension
- EsbOtlDisassociateAttributeMember
- EsbOtlFindAttributeMembers
- EsbOtlGetAssociatedAttributes
- EsbOtlGetAttributeInfo
- EsbOtlGetAttributeSpecifications
- EsbOtlQueryAttributes
- EsbOtlSetAttributeSpecifications

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 **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

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_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
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_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 progammer 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 fo **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_CALCSTRIPT"
    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
```

Parameter	Description
hCtx	Visual Basic API context handle
pMbrs	List of member names (or Aliases)
ppURLXMLLen	Returns length of URL XML generated
ppURLXML	Returns pointers to the URL XML byte stream

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

```
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

```
EsbGetDatabaseAccess (hCtx, User, AppName, DbName, pItems)
ByVal hCtx    As Long
ByVal User    As String
ByVal AppName As String
ByVal 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

```
EsbGetDatabaseInfo (hCtx, AppName, DbName, DbInfo, pItems)
ByVal hCtx    As Long
ByVal AppName As String
ByVal DbName  As String
      DbInfo As ESB_DBINFO_T
          pItems  As Integer
```

Parameter	Description
hCtx	VB API context handle
AppName	Application name
DbName	Database name
DbInfo	Buffer to receive a database info structure.
pItems	Number of "ESB_DBREQINFO_T" on page 1173 structures returned.

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

```
Declare Function EsbGetDatabaseInfo Lib "ESBAPIN" (ByVal hCtx As Long, ByVal AppName As
String, ByVal DbName As String, pDbInfo As ESB_DBINFO_T, Items As Integer) As Long

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

```
Declare Function EsbGetDatabaseInfoEx Lib "esbapin" (
ByVal 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)
ByVal hCtx     As Long
ByVal AppName  As String
ByVal DbName   As String
ByVal DbNote   As String
ByVal szDbNote As Integer
```

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 that 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.

Parameter	Description
DbName	Database name.
pDbState	Buffer to receive a database state structure.

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
```

Parameter	Description
hCtx	VB API context handle.
AppName	Application name.
DbName	Database name.
pDbStats	Buffer to receive a database stats structure.
pItems	Address of variable to receive Items of Dimension stats items.

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

```
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

- EsbGetNextItem
- EsbSetActive

EsbGetDefaultCalc

Gets the default calc script for the active database.

Syntax

```
EsbGetDefaultCalc (hCtx, cscString, szString)
ByVal hCtx      As Long
ByVal cscString As String
ByVal 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
```

Parameter	Description
hCtx	Visual Basic API context handle
URLName	Drill-through URL name
pUrl	URL definition
symRegions	List of symmetric regions

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

```
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
```

See also an extended example in "Drill-through Visual Basic API Example" on page 1132.

EsbGetFilter

Starts getting the contents of a filter.

Syntax

```
EsbGetFilter (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
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)
ByVal hCtx    As Long
ByVal FltRow  As String
ByVal szRow   As Integer
ByVal 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 that 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)
ByVal hCtx    As Long
      pGlobal As ESB_GLOBAL_T
```

Parameter	Description
hCtx	VB API context handle
pGlobal	Buffer to receive a global state 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

```
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

Gets a group information structure, which contains security information for the group.

Syntax

```
EsbGetGroup (hCtx, GrpName, pUserInfo)
ByVal hCtx      As Long
ByVal GrpName   As String
      pUserInfo As ESB_USERINFO_T
```

Parameter	Description
hCtx	VB API context handle.
GrpName	Group name.
pUserInfo	Buffer to receive a group info structure.

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)
ByVal hCtx    As Long
ByVal GrpName As String
      pItems  As Integer
```

Parameter	Description
hCtx	VB API context handle.
GrpName	Group or user name.
pItems	Address of variable to receive Items of user names.

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)
ByVal hCtx     As Long
ByVal ObjType  As Long
ByVal AppName  As String
ByVal DbName   As String,
ByVal ObjName  As String
ByVal isCreate As Integer
ByVal Path     As String
ByVal szPath   As Integer
```

Parameter	Description
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 = "Basic"
   ObjType = 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

```
EsbGetLocationAliasList (hCtx, ListCount, Aliases, Hosts, _
                         AppNames, DbNames, UserNames)
ByVal hCtx      As Long
ByRef ListCount As Integer
ByRef Aliases   As Variant
ByRef Hosts     As Variant
ByRef AppNames  As Variant
ByRef DbNames   As Variant
ByRef 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

```
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

  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 Long
ByVal LocalName  As String,
```

Parameter	Description
hCtx	VB API context handle
AppName	Application name. If *AppName* = " ", the Essbase.log is returned.

Parameter	Description
TimeStamp	Time stamp, indicating date & time of earliest log file entry required
LocalName	Full path name of local destination file on client

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

```
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

```
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 that 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

```
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 theStatus 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

```
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"
```

```
   '*************************
   ' Get Member Info structure
   '*************************
   sts = 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
```

```
   Next
End Sub
```

Global Constants for EsbGetNextItem

Constant	Context
ESB_USERINFO_TYPE = 1	ESB_USERINFO_T (EsbListUsers)
ESB_GROUPINFO_TYPE = 2	ESB_USERINFO_T (EsbListGroups)
ESB_USERAPP_TYPE = 3	ESB_USERAPP_T (EsbGetApplicationAccess)
ESB_USERDB_TYPE = 4	ESB_USERDB_T (EsbGetDatabaseAccess)
ESB_LOCKINFO_TYPE = 5	ESB_LOCKINFO_T (EsbListLocks)
ESB_OBJINFO_TYPE = 6	ESB_OBJINFO_T (EsbListObjects)
ESB_APPDB_TYPE = 7	ESB_APPDB_T (EsbListDatabases)
ESB_CAPPDB_TYPE = 8	ESB_APPDB_T (EsbListCurrencyDatabase)
ESB_APPNAME_TYPE = 9	ByVal *var* As String * ESB_APPNAMELEN (EsbListApplications)
ESB_DBNAME_TYPE = 10	ByVal *var* As String * ESB_DBNAMELEN (EsbGetApplicationInfo)
ESB_GROUPNAME_TYPE = 11	ByVal *var* As String * ESB_USERNAMELEN (EsbGetGroupList)
ESB_FTRNAME_TYPE = 12	ByVal *var* As String * ESB_FTRNAMELEN (EsbListFilters)
ESB_FUSERNAME_TYPE = 13	ByVal *var* As String * ESB_USERNAMELEN (EsbGetFilterList)
ESB_OBJNAME_TYPE = 14	ByVal *var* As String * ESB_OBJNAMELEN (EsbGetCalcList)
ESB_DIMSTATS_TYPE = 15	ESB_DIMSTATS_T (EsbGetDatabaseStats)
ESB_CUSERINFO_TYPE = 16	ESB_USERINFO_T (EsbListConnections)
ESB_LAPPDB_TYPE = 17	ESB_APPDB_T (EsbLogin)
ESB_ALIASNAME_TYPE = 18	ESB_ALIASNAME_T (EsbListAliases)
ESB_MBRALT_TYPE = 19	ESB_MBRALT_T (EsbDisplayAlias)
ESB_RATEINFO_TYPE = 20	ESB_RATEINFO_T (EsbGetCurrencyRateInfo)
ESB_OUTLINEINFO_TYPE = 21	ByVal *var* As String * ESB_ALIASNAMELEN (EsbOtlGetOutlineInfo)
ESB_OUTERROR_TYPE = 22	ESB_OUTERROR_T (EsbOtlVerifyOutline)
ESB_OTLUSERATTR_TYPE = 23	ByVal *var* As String * ESB_MBRNAMELEN (EsbOtlGetUserAttributes)
ESB_APPINFOEX_TYPE = 24	ESB_APPINFOEX_T (EsbGetApplicationInfoEx)
ESB_DBREQINFO_TYPE = 25	ESB_DBREQINFO_T (EsbGetDatabaseInfo)
ESB_DIMINFO_TYPE = 26	ESB_DIMENSIONINFO_T (EsbGetDimensionInfo)

Constant	Context
ESB_HMEMBER_TYPE = 27	ESB_HMEMBER_T (EsbOtlQueryMembers)
ESB_GENLEVELNAME_TYPE = 28	ESB_GENLEVELNAME_T (EsbOtlGetGenNames)
ESB_VARIABLE_TYPE = 29	ESB_VARIABLE_T (EsbListVariables)
ESB_LRO_TYPE = 30	ESB_LRODESC_T
ESB_PART_INFO_TYPE = 31	ESB_PART_INFO_T
ESB_DTS_TYPE = 32	ESB_DTSMBRINFO_T (EsbGetEnabledDTSMembers)
ESB_MBRNAME_TYPE = 33	ESB_MBRNAME_T (EsbOtlGetDimensionUserAttributes)

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

```
EsbGetObject (hCtx, ObjType, AppName, DbName, ObjName, LocalName, isLock)
ByVal hCtx      As Long
```

```
ByVal ObjType   As Long
ByVal AppName   As String
ByVal DbName    As String
ByVal objName   As String
ByVal LocalName As String
ByVal isLock    As Integer
```

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

```
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
```

```
   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 *pProcState*.
- 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 *pProcState*. Values for *pProcState*:

- 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

```
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

```
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 "esbapin" (
ByVal hCtx As Long,
ByVal userName As String,
      pUserInfo As ESB_USERINFOEX_T) As Long
```

Parameter	Description
hCtx	API context handle.
UserName	User name.
pUserInfoEx	Address of pointer to receive info structure of externally authenticated user.

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

```
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

```
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" Susbtitution 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)
ByVal hCtx     As Long
      Release  As Integer
      Version  As Integer
      Revision As Integer
```

Parameter Description

hCtx	VB API context handle.
Release	Address of variable to receive release number.
Version	Address of variable to receive version number.
Revision	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 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)
ByVal hCtx           As Long
      pRules         As ESB_OBJDEF_T
      pData          As ESB_OBJDEF_T
      User           As ESB_MBRUSER_T
ByVal ErrName        As String
ByVal isAbortOnError As Integer
```

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" (ByVal hCtx As Long, Rules As ESB_OBJDEF_T,
                                          Data As ESB_OBJDEF_T, User As ESB_MBRUSER_T,
                                         ByVal ErrName As String, ByVal AbortOnError As
Integer)
                                          As Long
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    '********************************
   ' 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

```
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.

EsbInit() returns an instance handle in *phInst*. The returned instance handle allows multiple applications to independently access the VB API (for DLLs only). The instance handle should be preserved and passed to the **EsbLogin**() and **EsbTerm**() functions.

Access

This function requires no special access.

Example

```
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

```
EsbKillRequest (hCtx, ReqInfo)
ByVal hCtx     As Long
ByVal 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
hCtx	VB API context handle.
pItems	Address of variable to receive Items of alias tables.

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

```
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)
ByVal hCtx   As Long
      pItems As Integer
```

Parameter	Description
hCtx	VB API context handle.

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
```

Parameter	Description
hCtx	API context handle.
CalcString	The string containing the available calculator functions. The string is in the form of XML.
szString	The size of the string containing the available calculator functions.

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><![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>
</group>
<group name="Custom">
</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

```
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

```
EsbListDatabases (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 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

```
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

```
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

```
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

```
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 Items 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 Items of data blocks which they currently have locked.

Syntax

```
EsbListLocks (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.

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

```
Declare Function EsbListLogins Lib "esbapin" (
ByVal hCtx As Long,
      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

```
EsbListObjects (hCtx, ObjType, AppName, DbName, pItems)
ByVal hCtx    As Long
ByVal ObjType As Long
ByVal AppName As String
ByVal DbName  As String
      pItems  As Integer
```

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
      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

```
EsbListUsers (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 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
```

Parameter	Description
hCtx	The API context handle.

Parameter	Description
AppName	The Application name. If NULL, lists all users.
DbName	The database name. If NULL, lists users for all databases within application.
Protocol	The name of the external authentication security protocol mechanism.
pItems	The count of users.

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

```
EsbListVariables (hCtx, pVariable, pItems)
ByVal hCtx      As Long
      pVariable As ESB_PVARIABLE_T
      pItems    As Integer
```

Parameter	Description
hCtx	Context handle to the API.

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

```
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

```
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)
ByVal hCtx     As Long
ByVal AppName As String
```

Parameter	Description
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

```
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)
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 load.

Return Value

None.

Access

This function requires the caller to have Database Load/Unload privilege (ESB_PRIV_APPLOAD).

Example

```
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

```
EsbLockObject (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. May 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	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

```
EsbLogin (hInst, Server, User, Password, pItems, hCtx)
ByVal hInst    As Long
ByVal Server   As String
ByVal User     As String
ByVal Password As String
      pItems   As Integer
      hCtx     As Long
```

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,
	`https://myhost:13080/aps/Essbase?clustername=Essbase-Cluster1&SecureMODE=Yes`

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
hInstance	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?clustername=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?clustername=Essbase-Cluster1&SecureMODE=Yes`

UserName	User name.
Password	Old password.
NewPassword	New password.
Items	Number of accessible databases.
hCtx	Essbase Server context handle.

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
hCtx	VB API context handle of user forcing the log out.
LoginId	Login ID of user to be logged out.

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

```
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

```
Declare Function EsbLogSize Lib "esbapin" (
ByVal hCtx As Long,
ByVal isAgentLog As Integer,
ByVal AppName As String,
      pulLogSize As Long) As Long
```

Parameter	Description
hCtx	API context handle.
isAgentLog	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.
AppName	Application name.

Parameter	Description
pulLogSize	Size of log file returned.

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

```
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 *pMemComb*.
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

```
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)
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	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
```

```
        '******************************
        '* Get the next LRO description
        '* item from the list
        '******************************         sts = EsbGetNextItem(hCtx,
ESB_LRO_TYPE, Desc)      Next i
   End IfEnd Sub
```

See Also

- "Constant and Structure Definitions for Linked Objects" on page 1155
- EsbGetNextItem
- EsbLROAddObject
- EsbLRODeleteObject
- EsbLROPurgeObjects

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

```
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

```
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 *n*th 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
```

Parameter	Description
hCtx	VB API context handle.
memberIndex	Position in the member list of the member to be returned.
memberName	Returned member's name.

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
```

```
            '************************************************************
            '* Get the jth member from the member list of the current LRO
            '************************************************************
            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

```
EsbLROGetObject (hCtx, pLinkId, targetFile, usOption, pLRODesc)
ByVal hCtx       As Long
      pLinkId    As ESB_LROHANDLE_API_T
ByVal targetFile As String
ByVal usOption   As Integer
      pLRODesc   As ESB_LRODESC_API_T
```

Parameter	Description
hCtx	API context handle.
pLinkId	Pointer to object identification structure. The link ID is returned by EsbLROAddObject through the "ESB_LRODESC_API_T" on page 1156 structure.
targetFile	The name of the target file into which the object is retrieved.
usOption	Option specifying whether to retrieve the object, its catalog entry, or both. Use one of the following: • ESB_LRO_OBJ_API retrieves only the object. • ESB_LRO_CATALOG_API retrieves only the catalog entry. • ESB_LRO_BOTH_API retrieves object and catalog entry.
pLRODesc	Object's description structure, "ESB_LRODESC_API_T" on page 1156.

Notes

Cell notes are part of the catalog entry for an object. To determine if the object is a cell note, check the *ObjType* 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 EsbLROListObjects(), 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 EsbLROListObjects Lib "esbapin" _
(ByVal hCtx As Long, ByVal userName As String, _
ByVal listDate As Long, PulCount As Integer) As Long

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 CutOffDate As Date

   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, 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
- 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 u*serName* 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
hCtx	API context handle.
pLinkId	Object identification structure.
usOption	Option specifying which part of the object to update. Use one of the following: • ESB_LRO_BOTH_API to update both the object's file and catalog entry • ESB_LRO_OBJ_API to update only the object's file • ESB_LRO_CATALOG_API to update only the object's catalog entry
pLRODesc	Object's description structure, "ESB_LRODESC_API_T" on page 1156.

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

```
Declare Function EsbLROUpdateObject Lib "esbapin" _
(ByVal hCtx As Long, 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

```
EsbPartitionApplyOtlChangeFile (hCtx, usFileNum, fileList)
ByVal hCtx      As Long
ByVal usfilenum As Integer
ByVal filelist  As String
```

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

```
EsbPartitionGetAreaCellCount (hCtx, pszSlice, pdCount)
ByVal hCtx     As Long
ByVal pszSlice As String
      pdCount  As Double
```

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

```
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
```

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)
ByVal hCtx          As Long
      MetaQuery     As ESB_PARTOTL_QUERY_T
ByVal ChangeFile    As String
ByVal szChangeFile As Integer
```

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

```
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 =
"Sampeast"
   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

```
EsbPartitionGetReplCells (hCtx, ReplicatedRegion, HostAppDbList)
ByVal hCtx             As Long
      ReplicatedRegion As ESB_PART_REPL_T
ByVal HostAppDbList    As String
```

Parameter	Description
hCtx	API context handle.
ReplicatePartition	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
- EsbPartitionPurgeOtlChangeFile
- 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

```
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
ReplicatedPartition	Partition information.
HostAppDbList	List of database/applications on the host server.

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

```
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

```
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
```

Parameter	Description
hCtx	API context handle.
pszChgFileName	The name of the metadata change file.
QueryFilter	The query filter expression.
MetaChangeReadHandle	Handle for the metadata change file.
SourceTime	The time of the latest change to the metadata file.

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`
- `EsbPartitionGetReplCells`
- `EsbPartitionPurgeOtlChangeFile`
- `EsbPartitionPutReplCells`
- `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

```
EsbPartitionResetOtlChangeTime (hCtx, pSourceRegion, pDestRegion)
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

```
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

```
EsbPutObject (hCtx, ObjType, AppName, DbName, ObjName, LocalName, isUnlock)
ByVal hCtx       As Long
ByVal ObjType    As Long
ByVal AppName    As String
ByVal DbName     As String
ByVal ObjName    As String
ByVal LocalName  As String
ByVal isUnlock   As Integer
```

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 put.
LocalName	Full path name of local source file on client.
isUnlock	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 (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, ByVal AppName 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
   Dim AppName As String
   Dim DbName As String
   Dim ObjName As String
   Dim ObjType As Integer
   Dim LocalName As String
   Dim Unlock As Integer   AppName = "Sample"
   DbName = "Basic"
   ObjName = "Basic"
   ObjType = ESB_OBJTYPE_TEXT
   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 |
```

```
                    <NEWLINESEPARATED |
                    <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:

  ```
  <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:

```
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

```
EsbRemoveLocks (hCtx, AppName, DbName, LoginId)
ByVal hCtx    As Long
ByVal AppName As String
ByVal DbName  As String
ByVal LoginId As Long
```

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.
- **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 (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

```
EsbRenameApplication (hCtx, AppName, nAppName)
ByVal hCtx     As Long
ByVal AppName  As String
ByVal nAppName As String
```

Parameter	Description
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.

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 (ESB_PRIV_APPCREATE).

Example

```
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)
ByVal hCtx    As Long
ByVal AppName As String
ByVal DbName  As String
ByVal nDbName As String
```

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 OldName As String, ByVal NewName 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
   '*****************
   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

```
EsbRenameGroup (hCtx, GrpName, nGrpName)
ByVal hCtx     As Long
ByVal GrpName  As String
ByVal nGrpName As String
```

Parameter	Description
hCtx	VB API context handle.
GrpName	Old name of an existing group to rename.
nGrpName	New name for the 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 (ESB_PRIV_USERCREATE) for the logged in server.

Example

```
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

```
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

```
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

```
EsbReportFile (hDestCtx, hSrcCtx, AppName, DbName, FileName, isOutput, isLock)
ByVal hDestCtx As Long
ByVal hSrcCtx  As Long
ByVal AppName  As String
ByVal DbName   As String
ByVal FileName As String
ByVal isOutput As Integer
ByVal 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
hCtx	API context handle
UserName	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 (ESB_PRIV_USERCREATE) for the logged in server.

Example

```
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 ESB_SetActive ()
   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

```
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

```
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

```
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

EsbSetApplicationState (*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	Pointer to application state structure.

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:

```
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

```
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
   Dim AppName 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
```

```
   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

```
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

```
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
```

```
ByVal DbName  As String
ByVal DbNote  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

EsbSetDefaultCalc

Sets the default calc script for the active database.

Syntax

```
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 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

```
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

```
EsbSetDefaultCalcFile (hDestCtx, hSrcCtx, AppName, DbName, FileName)
ByVal hDestCtx As Long
ByVal hSrcCtx  As Long
```

```
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

```
EsbSetFilter (hCtx, AppName, DbName, FltName, isActive, pAccess)
ByVal hCtx     As Long
ByVal AppName  As String
ByVal DbName   As String
ByVal FltName  As String
ByVal isActive As Integer
ByVal pAccess  As Integer
```

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

```
Declare Function EsbSetFilter Lib "ESBAPIN" (ByVal hCtx As Long, ByVal AppName As
String, ByVal DbName As String, ByVal FltName As String, ByVal is Active As Integer,
ByVal pAccess As Integer) As Long

Sub ESB_SetFilter()
   Dim sts As Long
   Dim AppName As String
   Dim DbName As String
   Dim FilterName As String
   Dim Active As Integer
   Dim pAccess As Integer
   Dim Row As String AppName = "Demo"
   DbName = "Basic"
```

```
   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
hCtx	VB API context handle.
AppName	Application name.
DbName	Database name.
FltName	Filter name.
UserList	String of user names (CR and EOL delimited), must be less than 64 KB in length.
Items	Number of user names in the string.

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

```
EsbSetPassword (hCtx, userName, Password)
ByVal hCtx      As Long
ByVal userName  As String
ByVal Password  As String
```

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

```
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

```
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
hCtx	VB API context handle.
pUserInfo	Pointer to user info structure.

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 User 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

Rerminates 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
```

See Also

- EsbLoadApplication
- EsbUnloadDatabase

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

- EsbLoadDatabase

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

- EsbGetObjectInfo
- EsbListObjects
- EsbLockObject
- EsbPutObject

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

```
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
AppName	Application name for the update file location.
DbName	Database name for the update file location.
FileName	Name of the update specification file.
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.

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

```
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
```

See Also

- EsbBeginUpdate
- EsbReportFile
- EsbUpdate

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

`EssValidateHCtx` (*hCtx*)

Parameter	Data Type	Description
hCtx	ESB_HCTX_T	The API context handle to validate

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

```
        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

```
EsbVerifyFilter (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	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
hCtx	API context handle.
isAgentLog	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`.
Message	Message to be logged to the Essbase Server log file (`essbase.log`), or to the application log file (*`appname`*`.log`).

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)

21 Using the Visual Basic Outline API

In This Chapter

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

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 relevent 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 **EssOtlFreeMembers**() 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**().

22 Visual Basic Outline API Declarations

In This Chapter

VB Outline Error Return Values 1451
VB Outline Symbolic Constant Definitions 1454
ESB_ATTRIBUTEQUERY_T 1458
ESB_GENLEVELNAME_T 1459
ESB_MBRCOUNTS_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 21 Error Return Values

Value	Description
OTLAPI_BAD_ALIASTABLE	Illegal alias table
OTLAPI_BAD_CONSOL	Invalid consolidation type (+,-,etc)
OTLAPI_BAD_GENLEVELNAME	Invalid generation or level name
OTLAPI_BAD_HOUTLINE	Invalid outline handle passed to EsbOtl... function
OTLAPI_BAD_MBRNAME	Invalid member name
OTLAPI_BAD_MEMBER	Invalid member handle
OTLAPI_BAD_MOVE	Illegal move of member; for example, can't move a member to its descendant
OTLAPI_BAD_OBJTYPE	Illegal object type
OTLAPI_BAD_OUTLINETYPE	Invalid outline type
OTLAPI_BAD_RENAMESHARE	A shared member cannot be renamed

Value	Description
OTLAPI_BAD_RESTRUCTTYPE	Invalid restructure type
OTLAPI_BAD_SORTCOMPAREFUNC	Invalid sorting compare function
OTLAPI_BAD_SORTTYPE	Invalid sort type
OTLAPI_BAD_TRANSTYPE	Unknown transaction type when creating a transaction (internal error)
OTLAPI_BAD_USERATTR	Invalid user attribute
OTLAPI_CUR_NOACCOUNTS	There is no Accounts dimension. You need an Accounts dimension to create a currency database
OTLAPI_CUR_NOCOUNTRY	There is no Country dimension. You need a Country dimension to create a currency database
OTLAPI_CUR_NOTIME	There is no Time dimension. You need a Time dimension to create a currency database.
OTLAPI_ERR_ADDNAMEUSED	Member name already used (add operation)
OTLAPI_ERR_ALIASSHARED	A shared member cannot have an alias
OTLAPI_ERR_ALIASTABLEEXISTS	Alias table already exists
OTLAPI_ERR_ALIASTABLENAME	Illegal alias table name
OTLAPI_ERR_ALREADYCURRENCY	The outline is a currency outline. You are trying to create a currency outline, and the initial outline is already a currency outline.
OTLAPI_ERR_BADSHARE	Illegal share value
OTLAPI_ERR_BADSKIP	Illegal time balance skip value
OTLAPI_ERR_BADSTORAGE	Illegal dimension storage value
OTLAPI_ERR_BADSTORAGECATEGORY	Illegal storage category
OTLAPI_ERR_BADTIMEBAL	Illegal time balance value
OTLAPI_ERR_CONFIGTOOMANYDIMS	Too many dimensions to configure automatically
OTLAPI_ERR_COPYALIASTABLE	Source and destination tables are the same
OTLAPI_ERR_CREATETEMP	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.
OTLAPI_ERR_CURTOOMANYDIMS	Too many dimensions in a currency outline. A currency outline is limited to four dimensions.
OTLAPI_ERR_DELETEDEFALIAS	Cannot delete the default alias table
OTLAPI_ERR_DUP_LANGCODE	The language code is assigned to another alias table within the same database
OTLAPI_ERR_DUPGENLEVNAME	Cannot add, rename, or set a member name or alias that duplicates a generation or level name
OTLAPI_ERR_DUPLICATEALIAS	Duplicate alias

Value	Description
OTLAPI_ERR_DUPLICATENAME	Duplicate member name
OTLAPI_ERR_FILEIO	Could not read from or write to file
OTLAPI_ERR_FILEOPEN	Could not open file
OTLAPI_ERR_GENLEVELEXISTS	Generation or level already has a name
OTLAPI_ERR_GENLEVELNAMEEXISTS	Generation or level name already exists
OTLAPI_ERR_GENLEVNAMEMBR	Cannot add a generation or level name that duplicates a member name or alias
OTLAPI_ERR_GENLEVELVALUE	Illegal generation or level value
OTLAPI_ERR_ILLEGALALIASSTRING	Illegal member combinational for alias
OTLAPI_ERR_ILLEGALCOMBOALIAS	Illegal combinational alias name
OTLAPI_ERR_ILLEGALCURRENCY	Illegal currency member
OTLAPI_ERR_ILLEGALDEFALIAS	Illegal default alias name
OTLAPI_ERR_ILLEGALNAME	Illegal member name
OTLAPI_ERR_ILLEGALTAG	Illegal dimension tag (category)
OTLAPI_ERR_INVALIDOPTION	Occurs when the user passes in an invalid option to **EssOtlGetGenNames()** or **EssOtlGetLevelNames()**
OTLAPI_ERR_LEAFLABEL	Leaf member defined as a label member
OTLAPI_ERR_MAXALIASTABLES	Maximum number of alias tables has been reached
OTLAPI_ERR_MEMBERCALC	Illegal member formula
OTLAPI_ERR_NOALIAS	No alias for this member
OTLAPI_ERR_NOALIASCOMBO	No alias combination
OTLAPI_ERR_NODTSMBRANDGENMATCH	DTS member is not enabled for this generation
OTLAPI_ERR_NOFORMULA	No formula for this member
OTLAPI_ERR_NOSHAREPROTO	Shared member with no actual member
OTLAPI_ERR_NOTADIM	Dimension name expected
OTLAPI_ERR_NOTIMEDIM	No time dimension defined (can't do time balance operations without a time dimension)
OTLAPI_ERR_NOTVERIFIED	Outline has errors (when saving to the server)
OTLAPI_ERR_OPENMODE	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.
OTLAPI_ERR_RENAMEDEFALIAS	Cannot rename the default alias table

Value	Description
OTLAPI_ERR_RENAMENAMEUSED	Member name already used (rename operation)
OTLAPI_ERR_SHAREDMEMBERFORMULA	Shared member cannot have a formula
OTLAPI_ERR_SHARENOTLEVEL0	Shared member not at level 0 (a shared member cannot be a parent of another member)
OTLAPI_ERR_SHAREUDA	Cannot set a user attribute for a shared member
OTLAPI_ERR_TIMESPARSE	Accounts dimension is dense and time dimension sparse-is not used
OTLAPI_NULL_ARG	NULL argument passed to EsbOtl... function
OTLAPI_NO_GENLEVELNAME	Cannot find generation or level name
OTLAPI_NO_USERATTR	Cannot find user attribute
OTLAPI_SORT_TOOMANY	Too many members to sort (64K / 4 members is the maximum sorting capacity)

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

Value	Description
ESB_DOR_ALLDATA	Keep all data
ESB_DOR_NODATA	Discard all data
ESB_DOR_LOWDATA	Keep only level 0 data

Value	Description
ESB_DOR_INDATA	Keep only input data

Table 23 Account Member Currency Conversion Category Values

Value	Description
ESB_CONV_NONE	Default conversion category. Member inherits category from parent.
ESB_CONV_CATEGORY	Define a currency conversion category for this member
ESB_CONV_NOCONV	No conversion for this member

Table 24 Account Member Time Balance Values

Value	Description
ESB_TIMEBAL_NONE	No time balance
ESB_TIMEBAL_FIRST	First time balance member
ESB_TIMEBAL_LAST	Last time balance member
ESB_TIMEBAL_AVG	Average time balance member

Table 25 Account Member Time Balance Skip Values

Value	Description
ESB_SKIP_NONE	Don't skip anything
ESB_SKIP_MISSING	Skip the value if the data is #missing
ESB_SKIP_ZEROS	Skip the value if the data is 0
ESB_SKIP_BOTH	Skip the value if the data is #missing or 0

Note: Account Member Time Balance Skip values are only valid if time balance does not equal ESB_TIMEBAL_NONE.

Table 26 Share Constants

Value	Description
ESB_SHARE_DATA	Normal member (default value)
ESB_SHARE_NEVER	Never share this member, even if it would normally be an implicit share.
ESB_SHARE_LABEL	Label member. Do not store data for this member.
ESB_SHARE_SHARE	Shared member. This member cannot have children and must have the actual member with the same name in the same dimension.

Table 27 Dimension Categories (Tags)

Value	Description
ESB_CAT_NONE	No category
ESB_CAT_ACCOUNTS	Accounts dimension
ESB_CAT_TIME	Time dimension
ESB_CAT_COUNTRY	Country dimension
ESB_CAT_TYPE	Type dimension. This dimension is valid only in currency databases
ESB_CAT_CURPARTITION	Currency partition dimension. Valid only in non-currency databases.

Table 28 Dimension Categories Used For Optimizing Auto-Configure Storage

Value	Description
ESB_STORECAT_OTHER	None or don't know storage category
ESB_STORECAT_TIME	Time storage category
ESB_STORECAT_UNITS	Units storage category
ESB_STORECAT_SCENARIO	Scenario storage category
ESB_STORECAT_ACCOUNTS	Accounts storage category
ESB_STORECAT_PRODUCT	Product storage category
ESB_STORECAT_ORGAN	Organization storage category
ESB_STORECAT_MARKET	Market storage category
ESB_STORECAT_CUSTOMER	Customer storage category
ESB_STORECAT_DIST	Distribution Channel storage category
ESB_STORECAT_BUSUNIT	Business Unit storage category
ESB_STORECAT_GEOG	Geographical Location storage category

Note: Used for optimizing storage when using storage auto-configure

Table 29 Sorting Options

Value	Description
ESB_SORT_ASCENDING	Sort in ascending order
ESB_SORT_DESCENDING	Sort in descending order

Query Types

Used for defining the operation to perform in **ESB_PREDICATE_T**.

- 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

Table 30 Query Options

Value	Description
ESB_MEMBERSONLY	Valid for ESB_SEARCH, ESB_WILDSEARCH
ESB_ALIASESONLY	Valid for ESB_SEARCH, ESB_WILDSEARCH
ESB_MEMBERSANDALIASES	Valid for ESB_SEARCH, ESB_WILDSEARCH
ESB_COUNTONLY	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.

Note: You can specify for certain query types in **ESB_PREDICATE_T**.

Table 31 Generation and Level Options

Value	Description
ESB_GENLEV_ALL	Returns default and user-defined names
ESB_GENLEV_ACTUAL	Returns only names that are user-defined
ESB_GENLEV_DEFAULT	Returns all default names, including the default names for generations and levels that also have user-defined names
ESB_GENLEV_NOACTUAL	Returns all default names, excluding the default names for generations and levels that also have user-defined names

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:

```
Type ESB_ATTRIBUTEQUERY_T
  InputMember      As Variant
  InputMemberType  As Integer
  OutputMemberType As Integer
  Operation        As Integer
  Attribute        As Variant
End Type
```

VB Data Type	Field	Description
As Variant	*InputMember*	Attribute member or dimension
As Integer	*InputMemberType*	One of the following member types: ● ESB_ATTRIBUTE_DIMENSION ● ESB_ATTRIBUTE_MEMBER ● ESB_STANDARD_DIMENSION ● ESB_STANDARD_MEMBER ● ESB_BASE_DIMENSION ● ESB_BASE_MEMBER ● ESB_ATTRIBUTED_MEMBER See Table 20, "VB API Attributes Terminology," on page 1164.
As Integer	*OutputMemberType*	One of the following member types: ● ESB_ATTRIBUTE_DIMENSION ● ESB_ATTRIBUTE_MEMBER ● ESB_STANDARD_DIMENSION ● ESB_STANDARD_MEMBER ● ESB_BASE_DIMENSION ● ESB_BASE_MEMBER ● ESB_ATTRIBUTED_MEMBER ● ESB_INVALID_MEMBER

VB Data Type	Field	Description
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: greather 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
End Type
```

Data Type	Field	Description
Integer	*usNumber*	Generation or level number.
String * ESB_MBRNAMELEN	*szName*	Generation or level name.

ESB_MBRCOUNTS_T

```
Type ESB_MBRCOUNTS_T

   ulStart        As Long
   ulMaxCount     As Long
   ulTotalCount   As Long
   ulReturnCount  As Long
End Type
```

Data Type	Field	Description
Long	*ulStart*	Starting member for retrieval of information.
Long	*ulMaxCount*	Maximum number of members to retrieve.
Long	*ulTotalCount*	Return of the total count of members that exist in the results of the query.
Long	*ulReturnCount*	Return count of returned member handles.

ESB_MBRINFO_T

Contains information about an outline member.

```
Type ESB_MBRINFO_T

   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
```

Data Type	Field	Description
String * ESB_ MBRNAMELEN	*szMember*	Member name. This field can be set only by the caller when creating the member.
Integer	*usLevel*	Level of the member in the outline. This field cannot be modified.
Integer	*usGen*	Generation of the member in the outline. This field cannot be modified.
Integer	*usConsolidation*	Unary consolidation type. It can be one of the following: ● ESB_UCALC_ADD ● ESB_UCALC_SUB ● ESB_UCALC_MULT ● ESB_UCALC_DIV ● ESB_UCALC_PERCENT ● ESB_UCALC_NOOP
Integer	*fTwoPass*	ESB_TRUE if two-pass calculation member
Integer	*fExpense*	ESB_TRUE if expense member
Integer	*usConversion*	Currency Conversion type. This is valid only for members of the Accounts dimension. It can be one of the following: ● ESB_CONV_NONE ● ESB_CONV_CATEGORY ● ESB_CONV_NOCONV

Data Type	Field	Description
String * ESB_ MBRNAMELEN	*szCurMember*	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.
Integer	*usTimeBalance*	Time balance option. This is valid only for members of the Accounts dimension. It can be one of the following: ● ESB_TIMEBAL_NONE ● ESB_TIMEBAL_FIRST ● ESB_TIMEBAL_LAST ● ESB_TIMEBAL_AVG
Integer	*usSkip*	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: ● ESB_SKIP_NONE ● ESB_SKIP_MISSING ● ESB_SKIP_ZEROS ● ESB_SKIP_BOTH
Integer	*usShare*	Share option. It can be one of the following: ● ESB_SHARE_DATA (default value) ● ESB_SHARE_DYNCALCSTORE ● ESB_SHARE_DYNCALCNOSTORE ● ESB_SHARE_NEVER ● ESB_SHARE_LABEL ● ESB_SHARE_SHARE (Valid for level 0 members only)
Integer	*usStorage*	Dimension storage type. This field is valid only for dimension members and can be one of the following: ● ESB_DIMTYPE_DENSE ● ESB_DIMTYPE_SPARSE
Integer	*usCategory*	Dimension category. This field is valid only for dimensions and attribute members. It can be one of the following: ● ESB_CAT_ACCOUNTS ● ESB_CAT_ATTRCALC (for internal use only) ● ESB_CAT_ATTRIBUTE ● ESB_CAT_COUNTRY ● ESB_CAT_CURPARTITION (for non-currency databases only) ● ESB_CAT_NONE ● ESB_CAT_TIME (for currency databases only) ● ESB_CAT_TYPE

Data Type	Field	Description
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
```

Data Type	Field	Description
Long	*hMember*	Handle to member with errors.
Long	*ulErrors*	Bitmask of errors for the member. It can be any combination of the following values: • ESB_OUTERROR_ALIASSHARED • ESB_OUTERROR_BADCATEGORY • ESB_OUTERROR_BADSHARE • ESB_OUTERROR_BADSKIP • ESB_OUTERROR_BADSTORAGE • ESB_OUTERROR_BADSTORAGECATEGORY • ESB_OUTERROR_BADTIMEBAL • ESB_OUTERROR_CURTOOMANYDIMS • ESB_OUTERROR_DUPGENLEVNAME • ESB_OUTERROR_DUPLICATEALIAS • ESB_OUTERROR_DUPLICATENAME • ESB_OUTERROR_ILLEGALALIASSTRING • ESB_OUTERROR_ILLEGALCOMBOALIAS • ESB_OUTERROR_ILLEGALCURRENCY • ESB_OUTERROR_ILLEGALDEFALIAS • ESB_OUTERROR_ILLEGALNAME • ESB_OUTERROR_ILLEGALTAG • ESB_OUTERROR_LEAFLABEL • ESB_OUTERROR_MEMBERCALC • ESB_OUTERROR_NOSHAREPROTO • ESB_OUTERROR_NOTIMEDIM • ESB_OUTERROR_SHAREDMEMBERFORMULA • ESB_OUTERROR_SHARENOTLEVEL0 • ESB_OUTERROR_SHAREUDA • ESB_OUTERROR_TIMESPARSE

ESB_OUTLINEINFO_T

```
Type ESB_OUTLINEINFO_T

   fCaseSensitive As String * 1
   usOutlineType  As Integer
   fAutoConfigure As String * 1
End Type
```

Data Type	Field	Description
String * 1	*fCaseSensitive*	ESB_TRUE if member names are case-sensitive.
Integer	*usOutlineType*	Type of the outline. It can be one of the following: ● ESB_DBTYPE_NORMAL ● ESB_DBTYPE_CURRENCY
String * 1	*fAutoConfigure*	ESB_TRUE to automatically configure the dimension storage (dense/sparse) when the outline is saved.

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	Field	Description
Long	*ulQuery*	Type of query. See `EsbOtlQueryMembers` for more information.
Long	*ulOptions*	Options dependent on the query type. See `EsbOtlQueryMembers` for more information.
String * ESB_ MBRNAMELEN	*pszDimension*	Dimension name. See `EsbOtlQueryMembers` for more information.
String * 256	*pszString1*	Input string value. See `EsbOtlQueryMembers` for more information.
String * 256	*pszString2*	Input string value. See `EsbOtlQueryMembers` for more information.

23 Visual Basic Outline API Functions

In This Chapter

Visual Basic Outline API Function Categories 1465
Visual Basic Outline API Function Reference 1470

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.

Function	Description
EsbOtlCreateAliasTable	Creates an empty alias table in the outline
EsbOtlCopyAliasTable	Copies an alias table to another alias table
EsbOtlRenameAliasTable	Renames an existing alias table

Function	Description
EsbOtlClearAliasTable	Clears all entries from an existing alias table without deleting the alias table
EsbOtlDeleteAliasTable	Deletes an alias table from the outline and clears all of its entries
EsbOtlSetAliasTableLanguage	Sets a language code for an alias table. An alias table can have multiple language codes.
EsbOtlGetAliasTableLanguages	Gets the set of language codes associated with an alias table.
EsbOtlClearAliasTableLanguages	Clears the set of language codes associated with an alias table.

VB Outline API Attributes Functions

These Visual Basic Outline functions are for attributes.

Function	Description
EsbOtlAssociateAttributeDimension	Associates an attribute dimension with a base dimension
EsbOtlAssociateAttributeMember	Associates an attribute member with a base member
EsbOtlDisassociateAttributeDimension	Disassociates an attribute dimension from a base dimension
EsbOtlDisassociateAttributeMember	Disassociates an attribute member from a base member
EsbOtlFindAttributeMembers	Returns all base members that are associated with an attribute member
EsbOtlGetAssociatedAttributes	Returns all attribute members that are associated with a base member or dimension
EsbOtlGetAttributeInfo	Returns attribute information for a given attribute member or dimension
EsbOtlGetAttributeSpecifications	Retrieves attribute specifications for the outline
EsbOtlQueryAttributes	Does complex queries concerning attributes
EsbOtlSetAttributeSpecifications	Sets attribute specifications for the outline

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.

Function	Description
EsbOtlDeleteDTSMemberAlias	Deletes an alias name for a Dynamic Time Series member.
EsbOtlEnableDTSMember	Enables a new Dynamic Time Series members for the outline.

Function	Description
EsbOtlGetEnabledDTSMembers	Gets the defined Dynamic Time Series members for the outline.
EsbOtlGetDTSMemberAlias	Gets an alias name for a Dynamic Time Series member.
EsbOtlSetDTSMemberAlias	Sets an alias name for a Dynamic Time Series member.

VB Outline API Generation Name Functions

These functions perform operations on generation names.

Function	Description
EsbOtlGetGenName	Gets the generation name for the specified dimension and generation number
EsbOtlGetGenNames	Retrieves all generation names specified for a particular dimension
EsbOtlSetGenName	Sets the generation name for the specified dimension and generation number
EsbOtlDeleteGenName	Deletes the generation name of the specified dimension and level number

VB Outline API Level Name Functions

These functions perform operations on level names.

Function	Description
EsbOtlGetLevelName	Gets the level name of the specified dimension
EsbOtlGetLevelNames	Retrieves all level names specified for a particular dimension
EsbOtlSetLevelName	Sets the level name of the specified dimension
EsbOtlDeleteLevelName	Deletes the level name of the specified dimension

VB Outline API Member Administration Functions

These functions assist in managing the members of an outline.

Function	Description
EsbOtlAddMember	Adds a member
EsbOtlDeleteMember	Deletes a member
EsbOtlAddDimension	Adds a dimension
EsbOtlDeleteDimension	Deletes a dimension

Function	Description
EsbOtlRenameMember	Renames a member
EsbOtlMoveMember	Moves a member
EsbOtlFindMember	Finds a member
EsbOtlGetMemberInfo	Gets member information
EsbOtlSetMemberInfo	Sets member information

VB Outline API Member Alias Functions

These functions perform operations on member aliases.

Function	Description
EsbOtlFindAlias	Finds a member with the specified alias name
EsbOtlGetMemberAlias	Gets the default member alias for a specific member in a specific alias table
EsbOtlSetMemberAlias	Sets the default member alias for a specific member in a specific alias table
EsbOtlDeleteMemberAlias	Deletes the default member alias for a specific member in a specific alias table
EsbOtlAddAliasCombination	Adds an alias combination to a member for a specific alias table
EsbOtlDeleteAliasCombination	Deletes an alias combination from a member for a specific alias table
EsbOtlGetNextAliasCombination	Returns the alias combinations for the specified member in the specified alias table

VB Outline API Member Formula Functions

These functions perform operations on member formulas.

Function	Description
EsbOtlGetMemberFormula	Gets the formula of the specified member
EsbOtlGetMemberLastFormula	Returns the last formula used to calculate the member
EsbOtlSetMemberFormula	Sets the formula for the specified member
EsbOtlDeleteMemberFormula	Deletes the formula of the specified member

VB Outline API Member Traversal Functions

These functions are used in traversing the outline tree.

Function	Description
EsbOtlGetFirstMember	Returns a member handle to the first member in the outline; the first dimension defined in the outline
EsbOtlGetChild	Returns a member handle to the child of a member
EsbOtlGetParent	Returns a member handle to the parent of a member
EsbOtlGetNextSibling	Returns a member handle to the next sibling of a member
EsbOtlGetPrevSibling	Returns a member handle to the previous sibling of a member
EsbOtlGetNextSharedMember	Returns a member handle to the next shared member of a real member

VB Outline API Outline Administration Functions

These functions assist in managing outlines.

Function	Description
EsbOtlGetOutlineInfo	Returns information about the outline file
EsbOtlGetUpdateTime	Returns the timestamp for the specified outline
EsbOtlSetOutlineInfo	Sets outline information
EsbOtlVerifyOutline	Verifies that an outline is correct
EsbOtlSortChildren	Sorts the children of an outline member
EsbOtlGenerateCurrencyOutline	Generates a currency outline based on the existing outline

VB Outline API Outline Query Functions

These functions assist in making outline queries.

Function	Description
EsbOtlOpenOutlineQuery	Opens an existing outline
EsbOtlQueryMembers	Queries the outline, using a member handle
EsbOtlQueryMembersByName	Queries the outline, using a member name string
EsbOtlFreeMember	Frees the member array returned from **EsbOtlQueryMembers()**

VB Outline API Setup and Cleanup Functions

These functions start and finish editing operations on an outline.

Function	Description
EsbOtlNewOutline	Creates a new outline
EsbOtlOpenOutline	Opens an existing outline
EsbOtlWriteOutline	Writes the outline to the server
EsbOtlRestructure	Restructures the database based on the newly saved outline
EsbOtlCloseOutline	Frees resources associated with the outline

VB Outline API User Attribute Functions

These functions perform operations on user attributes.

Function	Description
EsbOtlGetDimensionUserAttributes	Gets the user attributes of the specified dimension
EsbOtlGetUserAttributes	Gets the user attributes of the specified member
EsbOtlSetUserAttribute	Sets a user attribute for the specified member
EsbOtlDeleteUserAttribute	Deletes a user attribute of the specified member

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

```
EsbOtlAddAliasCombination (hOutline, hMember, pszAliasTable, pszAlias, pszCombination)
ByVal hOutline       As Long
ByVal hMember        As Long
ByVal pszAliasTable  As String
ByVal pszAlias       As String
ByVal pszCombination As String
```

Parameter	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.

Parameter	Description
pszAlias	Alias.
PszCombination	Member combination to associate with the alias. This can be a cross-dimensional member list.

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_ALIASSHARED
- OTLAPI_ERR_ILLEGALCOMBOALIAS
- OTLAPI_ERR_ILLEGALALIASSTRING
- OTLAPI_ERR_DUPLICATEALIAS

Example

```
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)
ByVal hOutline     As Long
      pMemberInfo  As ESB_MBRINFO_T
ByVal hPrevSibling As Long
ByVal pszDataMbr   As String
      phMember     As Long
```

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:

Field	Setting
usConsolidation	ESB_UCALC_NOOP

Field	Setting
fTwoPass	ESB_FALSE
fExpense	ESB_FALSE
usConversion	ESB_CONV_NONE
usTimeBalance	ESB_TIMEBAL_NONE
usSkip	ESB_SKIP_NONE
usShare	ESB_SHARE_DYNCALCNOSTORE
usStorage	ESB_DIMTYPE_SPARSE
usCategory	ESB_CAT_ATTRIBUTE
usStorageCategory	ESB_STORECAT_ATTRIBUTE
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

- OTLAPI_ERR_ILLEGALTAG
- OTLAPI_ERR_LEAFLABEL
- OTLAPI_ERR_NONATTRDIMFOLLOWED
- OTLAPI_ERR_NOSHAREPROTO
- OTLAPI_ERR_NOTIMEDIM

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
pMemberInfo	Member information structure defining the member and its attributes.
hparent	Handle of parent. This field is used only if the *hPrevSibling* field is ESB_NULL.
hPrevSibling	Handle of previous sibling.
phMember	Handle of new member returned from the API.

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:

Field	Setting
usConsolidation	ESB_UCALC_NOOP
fTwoPass	ESB_FALSE
fExpense	ESB_FALSE
usConversion	ESB_CONV_NONE
usTimeBalance	ESB_TIMEBAL_NONE
usSkip	ESB_SKIP_NONE
usShare	ESB_SHARE_DYNCALCNOSTORE

Field	Setting
usStorage	ESB_DIMTYPE_SPARSE
usCategory	ESB_CAT_ATTRIBUTE
usStorageCategory	ESB_STORECAT_ATTRIBUTE
Attribute	Attribute value. For an attribute dimension or zero-level (leaf node) attribute member, one of the following data types: ❍ Boolean (True|False) ❍ Date ("09/19/2006") ❍ Double (3.14) ❍ String ("Hello") For any attribute member, but not an attribute dimension: ESB_ATTRMBRDT_NONE = Everything else including empty.

- **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

```
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

```
EsbOtlAssociateAttributeDimension (hOutline, BaseDimension, AttributeDimension)
ByVal hOutline           As Long
ByVal BaseDimension      As Long
ByVal AttributeDimension As Long
```

Parameter	Description
hOutline	Handle to the outline
BaseDimension	Handle to the standard or base dimension
AttributeDimension	Handle to the base 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.

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(), ESBOTLNriteOutline(),
EsbOtlRestructure(),EsbUnlockObject() and
' EsbOtlCloseOutline() as neededà
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."
Out "ESB_OtlAssociateAttributeDimension() failed."
Exit Sub
End If

        AttrMbr = ESB_OtlFindMember("Enter attribute dimension: ")
```

```
        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(), ESBOTLNriteOutline(),
EsbOtlRestructure(),EsbUnlockObject() and
' EsbOtlCloseOutline() as neededà
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 EsbOtlClearAliasTable Lib "ESBOTLN"
(ByVal hOutline As Long, ByVal pszAliasTable As String) As Long

Sub ESB_OtlClearAliasTable()
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 = EsbOtlClearAliasTable(hOutline, "Default")
End If
End Sub
```

See Also

- EsbOtlCreateAliasTable
- EsbOtlCopyAliasTable
- EsbOtlRenameAliasTable
- EsbOtlDeleteAliasTable
- EsbOtlSetAliasTableLanguage

EsbOtlClearAliasTableLanguages

Clears the set of language codes associated with the specified alias table.

Syntax

```
ESB_FUNC_M EsbOtlClearAliasTableLanguages (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

```
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

```
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)
ByVal   hOutline              As Long
ByVal   pszSourceAliasTable   As String
ByVal   pszDestAliasTable     As String
ByVal   fMerge                As Integer
```

Parameter	Description
hOutline	Outline context handle.
pszSourceAliasTable	Name of alias table to copy from. If this parameter is "", the default alias table is used.
pszDestAliasTable	Name of alias table to copy to.
fMerge	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.

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_ALIASTABLE
- OTLAPI_ERR_MAXALIASTABLES
- OTLAPI_ERR_ALIASTABLENAME

Example

```
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"
```

```
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	Description
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

```
EsbOtlDeleteAliasCombination (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 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

- OTLAPI_ERR_NOALIASCOMBO

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_ALIASTABLE
- OTLAPI_ERR_DELETEDEFALIAS

Example

```
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

EsbOtlDeleteDimension (*hOutline, hMember, pszDataMbr*)

```
ByVal hOutline   As Long
ByVal hMember    As Long
ByVal pszDataMbr As String
```

Parameter	Description
hOutline	Outline context handle.
hMember	Handle of member to delete.

Parameter	Description
pszDataMbr	Member name in the dimension to be deleted from which data will be saved when the outline is restructured. If this field is "", the dimension is used.

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

```
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

- EsbOtlAddMember
- EsbOtlFindMember
- EsbOtlGetMemberInfo

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.
pzsAliasTable	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

```
EsbOtlDeleteGenName (hOutline, pszDimension, usGen)
ByVal hOutline     As Long
ByVal pszDimension As String
ByVal usGen        As Integer
```

Parameter	Description
hOutline	Outline context handle.
pszDimension	Name of the dimension that contains the generation.
usGen	Number of generation 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 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
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 Generation Name ***
'*********************************
Dimension = "Year"
GenNum = 2
GenName = "Qtr1 Qtr2 Qtr3 Qtr4"
If sts = 0 Then
   sts = EsbOtlDeleteGenName(hOutline,
    Dimension, GenNum)
End If
End Sub
```

See Also

- EsbOtlGetGenName
- EsbOtlGetGenNames
- EsbOtlSetGenName

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

```
EsbOtlDeleteMemberAlias (hOutline, hMember, pszAliasTable)
ByVal hOutline      As Long
ByVal hMember       As Long
ByVal pszAliasTable As String
```

Parameter	Description
hOutline	Outline context handle.
hMember	Handle of member to delete the alias from.
pszAliasTable	Alias table to delete the alias from. If this parameter is "", the default table is used.

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 = EsbOtlDeleteMemberAlias(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,
```

```
ByVal hMember As Long) 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)
ByVal hOutline  As Long
ByVal hMember   As Long
ByVal pszString As String
```

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)
ByVal hOutline           As Long
ByVal BaseDimension      As Long
ByVal AttributeDimension As Long
```

Parameter	Description
hOutline	Handle to the outline
BaseDimension	Handle to the base dimension
AttributeDimension	Handle to the attribute dimension

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

```
        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: " & sts: 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(ghOutline, BaseMbr, AttrMbr)
                sts = EsbOtlDeleteDimension(ghOutline, AttrMbr, "")
                If sts <> 0 Then
                                Out "EsbOtlDeleteDimension failed" & sts: Exit Sub
                Else
                                Out "EsbOtlDeleteDimension succeeded: " & sts
                End If
' abstract sub to call EsbOtlVerifyOutline(), ESBOTLNriteOutline(),
EsbOtlRestructure(),EsbUnlockObject() and
' EsbOtlCloseOutline() as neededà
tuckinoutline
                If sts <> 0 Then Out "EsbOtlDisassociateAttributeDimension failed: " &
sts: Exit Sub
End Sub
```

See Also

- EsbCheckAttributes
- EsbGetAssociatedAttributesInfo
- EsbGetAttributeInfo
- EsbGetAttributeSpecifications
- EsbOtlAssociateAttributeDimension
- EsbOtlAssociateAttributeMember

- EsbOtlDisassociateAttributeMember
- EsbOtlFindAttributeMembers
- EsbOtlGetAssociatedAttributes
- EsbOtlGetAttributeInfo
- EsbOtlGetAttributeSpecifications
- EsbOtlQueryAttributes
- EsbOtlSetAttributeSpecifications

EsbOtlDisassociateAttributeMember

Disassociates an attribute member from a base member.

Syntax

```
EsbOtlDisassociateAttributeMember (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 base member
AttributeMember	Handle to the attribute member

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

```
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. "
```

```
& 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(), ESBOTLNriteOutline(),
EsbOtlRestructure(),EsbUnlockObject() and
' EsbOtlCloseOutline() as neededà
tuckinoutline
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)
ByVal hOutline       As Long
ByVal MemberName     As String
ByVal DimensionName  As String
      Count          As Integer
      MemberArray    As Variant
```

Parameter	Description
hOutline	Handle to the outline
MemberName	Attribute short name
DimenisonName	Attribute dimension name (optional)
Count	Number of members returned
MemberArray	Array of base member handles

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

- EsbOtlGetAttributeInfo
- EsbOtlGetAttributeSpecifications
- EsbOtlQueryAttributes
- EsbOtlSetAttributeSpecifications

EsbOtlFindMember

Finds a member with the specified name and returns a handle to the member.

Syntax

```
EsbOtlFindMember (hOutline, pszMember, phMember)
ByVal hOutline  As Long
ByVal pszMember As String
      phMember  As Long
```

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
```

```
   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
          sts = EsbGetNextItem(hCtx, ESB_HMEMBER_TYPE, ihMember)
       If sts = 0 And ihMember <> 0 Then
                sts = EsbOtlFreeMember(hOutline, ihMember)
       End If
      Next
     End If
   End If
End If
End Sub
```

See Also

- EsbOtlOpenOutlineQuery
- EsbOtlQueryMembers
- EsbOtlQueryMembersByName

EsbOtlGenerateCurrencyOutline

Generates a currency outline based on the existing outline.

Syntax

```
EsbOtlGenerateCurrencyOutline (hOutline, phCurOutline)
ByVal hOutline     As Long
      phCurOutline As Long
```

Parameter	Description
hOutline	Outline contexthandle.
phCurOutline	Return variable for the currency outline handle.

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 **ESBOTLNriteOutline**() 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

```
ESB_FUNC_M EsbOtlGetAliasTableLanguages (hOutline, pszAliasTable, pItems)
ByVal   hOutline        As Long
ByVal   pszAliasTable   As String
        pItems          As Long
```

Parameter	Description
hOutline	Handle to the outline.
pszAliasTable	Name of the alias table for which to get an associated language code.
pItems	Address of a variable in which to return the number of language codes associated with the alias 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_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

- EsbOtlClearAliasTableLanguages
- EsbOtlSetAliasTableLanguage

EsbOtlGetAssociatedAttributes

Returns all attribute members that are associated with a base member or dimension.

Syntax

```
EsbOtlGetAssociatedAttributes (hOutline, Member, Count, MemberArray)
ByVal hOutline    As Long
ByVal Member      As Long
      Count       As Integer
      MemberArray As Variant
```

Parameter	Description
hOutline	Handle to the outline
Member	Handle to the base member or base dimension
Count	Number of attribute members returned
MemberArray	Array of attribute member handles

Return Value

Returns 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
```

Parameter	Description
hOutline	Handle to the outline
Member	Handle to the attribute member or dimension
AttrInfo	Attribute information

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

```
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 = EsbOtlGetAttributeSpecifications(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
```

```
        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_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

```
EsbOtlGetChild (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 child of.
phMember	Return variable for the handle of the child of the *hMember* parameter.

Notes

- If there is no child, **phMember* is set to ESB_NULL and the call returns 0.

Return Value

Returns 0 if successful.

Example

```
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.

Syntax

```
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.pszDimension = "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 If
End Sub
```

See Also

- EsbGetNextItem
- EsbOtlOpenOutlineQuery
- EsbOtlQueryMembers
- EsbOtlQueryMembersByName

EsbOtlGetDTSMemberAlias

Gets an alias name for a DTS member.

Syntax

```
EsbOtlGetDTSMemberAlias (hOutline, pszDTSMember, pszAliasTable, ppszAlias)
ByVal hOutline      As Long
ByVal pszDTSMember  As String
ByVal pszAliasTable As String
ByVal pszAlias      As String
```

Parameter	Description
hOutline	Essbase outline handle returned from the **EsbOtlOpenOutlineQuery** call.
pszDTSMember	Name of the DTS member which provides the alias.
pzsAliasTable	Name of the alias table which provides the alias. If NULL, the default alias table is used.
ppszAlias	Pointer to a pointer to a C string containing the alias name for the DTS member.

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_DTSMBRNOTDEFINED
- OTLAPI_BAD_ALIASTABLE

Example

```
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)
```

```
         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

```
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
Object.AppName = "Sample"
Object.DbName = "Basic"
Object.FileName = "Basic"
sts = EsbOtlOpenOutline(hCtx, Object,
ESB_YES, ESB_YES, hOutline)
If sts = 0 Then
   sts = EsbOtlGetFirstMember
    (hOutline, hMemberFirst)
```

```
End If
End Sub
```

See Also

- EsbOtlGetParent
- EsbOtlGetNextSibling
- EsbOtlGetPrevSibling
- EsbOtlGetChild

EsbOtlGetGenName

Retrieves the name for a specific generation within a dimension. Generation names are explicitly added to the outline with EsbOtlSetGenName.

Syntax

```
EsbOtlGetGenName (hOutline, pszDimension, usGen, pszName)
ByVal hOutline     As Long
ByVal pszDimension As String
ByVal usGen        As Integer
ByVal pszName      As String
```

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.
ppszName	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

```
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
```

Parameter	Description
hOutline	Essbase 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

```
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"
     GenOtp = ESB_GENLEV_DEFAULT
```

```
     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

```
EsbOtlGetLevelName (hOutline, pszDimension, usLevel, pszName)
ByVal hOutline     As Long
ByVal pszDimension As String
ByVal usLevel      As Integer
ByVal 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.

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

```
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"
   LevOtp = 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
```

Parameter	Description
hOutline	Outline context handle.
hMember	Handle of member to get the alias for.
pszAliasTable	Alias table to get the alias from. If this parameter is "", the default alias table is used.
pszAlias	Buffer for the return of the alias. The buffer is allocated by the caller.

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:

OTLAPI_BAD_ALIASTABLE

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

```
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)
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.
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.

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

```
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

```
EsbOtlGetNextAliasCombination (hOutline, hMember, pszAliasTable, pszAlias,
pszCombination,
usBufSize)
ByVal hOutline       As Long
ByVal hMember        As Long
ByVal pszAliasTable  As String
ByVal pszAlias       As String
ByVal pszCombination As String
ByVal usBufSize      As Integer
```

Parameter	Description
hOutline	Outline context handle.
hMember	Handle of member to retrieve the alias combination from.
pszAliasTable	Alias table to retrieve the alias combination from. If this parameter is "", the default alias table is used.
pszAlias	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.
pszCombination	Member combination of the returned alias. This buffer is allocated by the caller.
usBufSize	Size of the *pszCombination* buffer.

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

```
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

```
EsbOtlGetNextSharedMember (hOutline, hMember, phMember)
ByVal hOutline As Long
ByVal hMember  As Long
      phMember As Long
```

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

```
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
```

See Also

- EsbOtlFindMember

EsbOtlGetNextSibling

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
```

```
   sts = EsbOtlGetNextSibling(hOutline, hChild, hNextSibling)
End If
End Sub
```

See Also

- EsbOtlGetPrevSibling
- EsbOtlGetParent
- EsbOtlGetChild
- EsbOtlGetFirstMember

EsbOtlGetOutlineInfo

Returns information about the outline file.

Syntax

```
EsbOtlGetOutlineInfo (hOutline, pInfo, pusCount)
ByVal hOutline As Long
      pInfo    As ESB_OUTLINEINFO_T
      pusCount As Integer
```

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

```
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"
```

```
Object.FileName = "Basic"
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

Returns 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

```
EsbOtlGetPrevSibling (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 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

```
Declare Function EsbOtlGetPrevSibling Lib
"ESBOTLN" (ByVal hOutline As Long, ByVal hMember As Long,
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
```

```
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)
ByVal hOutline        As Long
ByVal hMember         As Long
ByVal hNewParent      As Long
ByVal hNewPrevSibling As Long
```

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 hFQ 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", hFQ)
End If
If sts = 0 Then
   sts = EsbOtlFindMember(hOutline, "Profit", hMemberProfit)
End If
If sts = 0 And hFQ And hMemberProfit Then
   sts = EsbOtlMoveMember(hOutline, hFQ,
    hMemberProfit, ESB_NULL)
End If
End Sub
```

See Also

- EsbOtlFindMember
- EsbOtlRenameMember
- EsbOtlAddMember
- EsbOtlDeleteMember

EsbOtlNewOutline

Creates an outline without creating a file. Used as an alternative to **EsbOtlOpenOutline**().

Syntax

```
EsbOtlNewOutline (hCtx, pNewInfo, phOutline)
ByVal hCtx       As Long
      pNewInfo   As ESB_OUTLINEINFO_T
      phOutline  As Long
```

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

```
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,
```

```
phOutline As Long) As Long

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

- EsbOtlGetMemberAlias
- EsbOtlGetMemberFormula
- EsbOtlGetMemberInfo
- EsbOtlGetNextAliasCombination
- EsbOtlGetOutlineInfo
- EsbOtlGetUserAttributes
- EsbOtlGetGenName
- EsbOtlGetGenNames
- EsbOtlGetLevelName
- EsbOtlGetLevelNames

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
   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)
   End If
End Sub
```

See Also

- EsbOtlCloseOutline
- EsbOtlOpenOutline
- EsbOtlQueryMembers
- EsbOtlQueryMembersByName
- EsbSetActive

EsbOtlQueryAttributes

Queries member information for a given attribute member or dimension.

Syntax

```
EsbOtlQueryAttributes (hOutline, AttrQuery, Count, MemberArray)
ByVal hOutline    As Long
      AttrQuery   As ESB_ATTRIBUTEQUERY_T
      Count       As Long
      MemberArray As Variant
```

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

```
        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
```

```
                        Else
                        Out "EsbOtlGetMemberInfo Failed: " & sts
                        End If
                Next index
        Else
                Out "EsbOtlQueryAttributes failed: " & sts
        End If
End Sub
```

See Also

- EsbCheckAttributes
- EsbGetAssociatedAttributesInfo
- EsbGetAttributeInfo
- EsbGetAttributeSpecifications
- EsbOtlAssociateAttributeDimension
- EsbOtlAssociateAttributeMember
- EsbOtlDisassociateAttributeDimension
- EsbOtlDisassociateAttributeMember
- EsbOtlFindAttributeMembers
- EsbOtlGetAssociatedAttributes
- EsbOtlGetAttributeInfo
- EsbOtlGetAttributeSpecifications
- EsbOtlSetAttributeSpecifications

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
```

Parameter	Description
hOutline	Essbase outline handle. This must have been returned from **EsbOtlOpenOutlineQuery**().
hMember	The member 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: • ESB_NAMEDGENERATION • ESB_NAMEDLEVEL • ESB_USERATTRIBUTE • ESB_SEARCH • ESB_WILDSEARCH
pPredicate	Structure defining the query. The fields of this structure are described in Notes.

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 leve.
  - 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
```

```
   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

   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)
       End If
      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)
ByVal hOutline   As Long
ByVal pszMember  As String
      pPredicate As ESB_PREDICATE_T
      pCounts    As ESB_MBRCOUNTS_T
```

Parameter	Description
hOutline	Essbase outline handle. This must have been returned from **EsbOtlOpenOutlineQuery**().

Parameter	Description
pszMember	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: ● 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

- 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:
  - SB_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
  - 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
                                                           'else message with error
      Predicate.ulQuery = ESB_CHILDREN
      Predicate.pszDimension = "Year"
      Counts.ulStart = 0
      Counts.ulMaxCount = 10
      If sts = 0 Then
         sts = EsbOtlQueryMembersByName(hOutline, pszMember, 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)
          End If
          Next
        End If
      End If
   Else
     msgbox "Outline open failed with error: " & sts
   Endif
End Sub
```

See Also

- EsbGetNextItem
- EsbOtlFreeMember
- EsbOtlGetDimensionUserAttributes
- EsbOtlOpenOutlineQuery
- EsbOtlQueryMembers

EsbOtlRenameAliasTable

Renames an existing alias table.

Syntax

```
EsbOtlRenameAliasTable (hOutline, pszAliasTable, pszNewAliasTable)
ByVal hOutline         As Long
ByVal pszAliasTable    As String
ByVal pszNewAliasTable As String
```

Parameter	Description
hOutline	Outline context handle.
pszAliasTable	Name of alias table to rename.
pszNewAliasTable	Name of new 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

```
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
```

See Also

- EsbOtlCreateAliasTable
- EsbOtlCopyAliasTable
- EsbOtlClearAliasTable
- EsbOtlDeleteAliasTable
- EsbOtlSetAliasTableLanguage

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_ATTRMBRDT_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 hMemberProfit <> 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
'ESBOTLNriteOutline()
'***
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

- 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

```
ESB_FUNC_M EsbOtlSetAliasTableLanguage (hOutline, pszAliasTable, pszLanguageCode)
ByVal   hOutline          As Long
ByVal   pszAliasTable     As String
ByVal   pszLanguageCode   As String
```

Parameter	Description
hOutline	Outline context handle.
pszAliasTable	Name of the alias table for which to set a language code.
pszLanguageCode	A language code to assign to the alias table specified in *pszAliasTable*. The language code should be a middle-tier language tag from an ApplCore session. Language codes are not case-sensitive.

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_SUFFIX")
     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" & vbCrLf & _
```

```
          "1. ESB_ESB_LOWERBOUNDINCLUSIVE" & vbCrLf & _
          "2. ESB_UPPERBOUNDNONINCLUSIVE" & vbCrLf & _
          "3. ESB_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
          Exit Sub
     End If
End Sub
```

See Also

- EsbCheckAttributes
- EsbGetAssociatedAttributesInfo
- EsbGetAttributeInfo
- EsbGetAttributeSpecifications
- EsbOtlAssociateAttributeDimension
- EsbOtlAssociateAttributeMember
- EsbOtlDisassociateAttributeDimension
- EsbOtlDisassociateAttributeMember
- EsbOtlFindAttributeMembers
- EsbOtlGetAssociatedAttributes
- EsbOtlGetAttributeInfo
- EsbOtlGetAttributeSpecifications

- EsbOtlQueryAttributes

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.
pzsAliasTable	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)
ByVal hOutline     As Long
ByVal pszDimension As String
ByVal usGen        As Integer
ByVal pszName      As String
```

Parameter	Description
hOutline	Outline context handle.
pszDimension	Name of dimension that contains the generation.
usGen	Number of generation for which to set a name. The dimension itself is generation 1.
pszName	Name to give the generation.

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_GENLEVELEXISTS
- OTLAPI_ERR_GENLEVELVALUE
- LAPI_ERR_NOTADIM
- OTLAPI_ ERR_GENLEVELNAMEMBR

Example

```
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
```

```
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

```
EsbOtlSetLevelName (hOutline, pszDimension, usLevel, pszName)
ByVal hOutline     As Long
ByVal pszDimension As String
ByVal usLevel      As Integer
ByVal 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_GENLEVELEXISTS
- 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_ILLEGALDEFALIAS
- 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",
```

```
    hMemberYear)
End If
If sts = 0 And hMemberYear <> 0 Then
   szAlias = "Year Dimension"
   sts = EsbOtlSetMemberAlias(hOutline,
    hMemberYear, "", szAlias)
End If
End Sub
```

See Also

- EsbOtlGetMemberAlias
- 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
hOutline	Outline context handle.
hMember	Member handle.
pszFormula	Buffer containing the member formula.

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	*IsAttributed*	Indicates whether the member has attributes associated with it.

- Values for two fields of the ESB_MBRINFO_T structure are for attributes only:

Data Type	Field	Description
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

```
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

```
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

EsbOtlSetUserAttribute (*hOutline, hMember, pszString*)

```
ByVal hOutline  As Long
ByVal hMember   As Long
ByVal pszString As String
```

Parameter	Description
hOutline	Outline context handle.
hMember	Handle of member for which to set the user-defined attribute.
pszString	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 **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

```
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
```

```
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

```
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 (**pulErrors* == 0 and **pulCount* == 0).

➤ To retrieve error values:

1 **Allocate an ESB_OUTERROR_T structure.**

2 **Call EsbGetNextItem() once for each member error (returned in the *pulCount* variable).**

Each call to **EsbGetNextItem**() returns the error information for a member in an ESB_OUTERROR_T structure.

Return Value

Returns 0 if successful; otherwise one of the following:

- ESB_OUTERROR_SHAREUDA
- ESB_OUTERROR_DUPGENLEVNAME

Example

```
Declare Function EsbOtlVerifyOutline Lib
"ESBOTLN" (ByVal hOutline As Long, pulErrors As Long,
pulCount As Long) As Long

Sub ESB_OtlVerifyOutline()
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 = EsbOtlOpenOutline(hCtx, Object,
ESB_YES, ESB_YES, hOutline)
'body of code
If sts = 0 Then
   sts = EsbOtlVerifyOutline(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

- EsbOtlNewOutline
- EsbOtlOpenOutline
- EsbOtlWriteOutline

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

24 Example of Traversing an Outline (VB)

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.

```
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);
}
```

Part VII

Other APIs

In Other APIs:

- Java API Reference
- MDX Provider API
- Welcome to XMLA Reference
- Working with XMLA

25 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.

26 MDX Provider API

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:

```
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]),
```

```
    ([Additions], [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:

```
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:

```
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:

```
/* 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);
```

Parameter	Type	Description
hQry	input	Query handle

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);
```

Parameter	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

```
ESS_FUNC_M EssMdxGetAxes(
   ESS_MDX_QRYHDL_T      hQry,
   ESS_PULONG_T          pulNAxes,
   ESS_MDX_PPAXISHDL_T   pphAxes,
   ESS_MDX_PAXISHDL_T    phSlicer);
```

Parameter	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

```
ESS_FUNC_M EssMdxGetAxisInfo(
   ESS_MDX_AXISHDL_T    hAxis,
   ESS_PULONG_T         pulSize,
   ESS_PULONG_T         pulNDims,
   ESS_MDX_PPDIMHDL_T   pphDims);
```

Parameter	Type	Description
hAxis	input	Axis handle
pulSize	output	Number of tuples in axis
pulNDims	output	Number of dimensions in axis
pphDims	output	Array of dimension handles

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

```
ESS_FUNC_M EssMdxGetAxisMembers(
   ESS_MDX_AXISHDL_T    hAxis,
   ESS_ULONG_T          ulIndex,
   ESS_MDX_PPMBRHDL_T   pphMbrs);
```

Parameter	Type	Description
hAxis	input	Axis handle
ulIndex	input	Tuple position within axis
pphMbrs	output	Array of member handles for tuple

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" on page 1590
- "EssMdxGetValue" on page 1601

Syntax

```
ESS_FUNC_M EssMdxGetCellAtIndices(
   ESS_MDX_QRYHDL_T     hQry,
   ESS_PULONG_T         pulIndices,
   ESS_MDX_PCELLHDL_T   phCell);
```

Parameter	Type	Description
hQry	input	Query handle
pulIndices	input	Tuple indices, one for each axis
phCell	output	Cell handle

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

```
ESS_FUNC_M EssMdxGetCellAtOffset(
   ESS_MDX_QRYHDL_T     hQry,
   ESS_ULONG_T          ulOffset,
   ESS_MDX_PCELLHDL_T   phCell);
```

Parameter	Type	Description
hQry	input	Query handle
ulOffset	input	Cell offset (first axis changes fastest)
phCell	input	Cell handle

EssMdxGetCellInfo

Returns the type of the cell corresponding to the input cell handle.

Syntax

```
ESS_FUNC_M EssMdxGetCellInfo (
    ESS_MDX_CELLHDL_T       hCell,
```

```
    ESS_PULONG_T             pUlType,
    ESS_MDX_PCELLINFO_T      pulCellInfo,
    ESS_MDX_PCELLSTATUS_T    pulStatus);
```

Parameter	Type	Description
hCell	input	Cell handle
pulType	output	Cell data type. Values: ● ESS_MDX_VALTYPE_DOUBLE Numeric type ● ESS_MDX_VALTYPE_SMARTLIST Smartlist type ● ESS_MDX_VALTYPE_DATE Date type
pulCellInfo	output	Cell status bit map specified using the following bitmasks: ● ESS_MDX_CELLINFO_MISSING The cell value is missing ● ESS_MDX_CELLINFO_NOACCESS The cell value is not accessible to the current user. ● ESS_MDX_CELLINFO_MEANINGLESS The cell value is meaningless in the context of attribute members ● ESS_MDX_CELLINFO_OUTOFRANGE The cell value is out of range in the context of a smartlist
pulStatus	output	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.

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

```
ESS_FUNC_M EssMdxGetCellStatus(
   ESS_MDX_QRYHDL_T         hQry,
   ESS_MDX_CELLHDL_T        hCell,
   ESS_MDX_PCELLSTATUS_T    pulStatus);
```

Parameter	Type	Description
hQry	input	Query handle
hCell	input	Cell handle

Parameter	Type	Description
pulStatus	output	Cell status: Bitmap with the following masks: • ESS_MDX_CELLSTATUS_LINKEDOBJS • ESS_MDX_CELLSTATUS_DYNCALC • ESS_MDX_CELLSTATUS_CALCED • ESS_MDX_CELLSTATUS_READONLYMBR

EssMdxGetClusterDimMembers

Returns the member handles for the specified dimension within the given cluster.

Syntax

```
ESS_FUNC_M EssMdxGetClusterDimMembers(
   ESS_MDX_CLUSTERHDL_T   hCluster,
   ESS_ULONG_T            ulIndex,
   ESS_MDX_PPMBRHDL_T     pphMbrs);
```

Parameter	Type	Description
hCluster	input	Cluster handle
ulIndex	input	Dimension index within axis containing cluster
pphMbrs	output	Array of member handles for the specified dimension

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

```
ESS_FUNC_M EssMdxGetClusterInfo(
   ESS_MDX_CLUSTERHDL_T   hCluster,
   ESS_PULONG_T           pulSize,
   ESS_PULONG_T           pulNDims,
   ESS_PPULONG_T          ppulDimSizes);
```

Parameter	Type	Description
hCluster	input	Cluster handle
pulSize	output	Number of tuples in cluster
pulNDims	output	Number of dimensions in cluster (same as that in the axis that this cluster belongs to)
ppulDimSizes	output	Array of dimension sizes (number of members)

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

```
ESS_FUNC_M EssMdxGetClusterMembers(
   ESS_MDX_CLUSTERHDL_T   hCluster,
   ESS_ULONG_T            ulIndex,
   ESS_MDX_PPMBRHDL_T     pphMbrs);
```

Parameter	Type	Description
hCluster	input	Cluster handle
ulIndex	input	Tuple position within cluster (first dimension changes slowest)
pphMbrs	output	Array of member handles for the tuple

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" on page 1593
- "EssMdxGetClusterMembers" on page 1594
- "EssMdxGetAxisMembers" on page 1590
- "EssMdxGetMbrIdentifier" on page 1597
- "EssMdxGetMbrProperty" on page 1597

Syntax

```
ESS_FUNC_M EssMdxGetClusters(
   ESS_MDX_AXISHDL_T        hAxis,
   ESS_PULONG_T             pulNClusters,
   ESS_MDX_PPCLUSTERHDL_T   pphClusters);
```

Parameter	Type	Description
hAxis	input	Axis handle
pulNClusters	output	Number of clusters
pphClusters	output	Array of cluster handles

EssMdxGetDimInfo

Returns information about the specified dimension, including the properties available for members in this dimension.

Syntax

```
ESS_FUNC_M EssMdxGetDimInfo(
   ESS_MDX_DIMHDL_T      hDim,
   ESS_PSTR_T            ppszName,
   ESS_PULONG_T          pulNProps,
   ESS_MDX_PPPROPHDL_T   pphProps);
```

Parameter	Type	Description
hDim	input	Dimension handle
ppszDimName	output	Dimension name
pulNProps	output	Number of properties returned
pphProps	output	Array of property handles

Notes

- Before calling this query, you should call "EssMdxGetAxisInfo" on page 1589to 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 "EssMdxExecuteQuery" on page 1588 to execute the query.
3. Call "EssMdxGetAxes" on page 1589 to get the number of axes and the individual axis handles from the result of the query.
4. Call "EssMdxGetAxisInfo" on page 1589 to get information (dimensions/tuples) for an individual axis from an axis handle.
5. Call "EssMdxGetDimInfo" on page 1595 to get information for a dimension (dimension name, number of properties for this dimension, and property handles).
6. Call "EssMdxGetPropertyInfo" on page 1599 to get the dimension properties. To get properties, the MDX query in EssMdxQuery must use the DIMENSION PROPERTIES option.

EssMdxGetFormatString

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 EssMdxSetQueryCellProperties.

Syntax

```
ESS_FUNC_M EssMdxGetFormatString(
   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

"EssMdxGetFormattedValue" on page 1596

EssMdxGetFormattedValue

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 EssMdxSetQueryCellProperties.

Syntax

```
ESS_FUNC_M EssMdxGetFormattedValue(
   ESS_MDX_QRYHDL_T     hQry,
   ESS_MDX_CELLHDL_T    hCell,
   ESS_PSTR_T           pFmtVal);
```

Parameter	Type	Description
hQry	input	Input query handle
hCell	input	Input cell handle
pFmtVal	output	Formatted value of the cell

See Also

"EssMdxGetFormatString" on page 1596

EssMdxGetMbrIdentifier

Returns the identifier for the specified member.

Syntax

```
ESS_FUNC_M EssMdxGetMbrIdentifier(
   ESS_MDX_MBRHDL_T    hMbr,
   ESS_PSTR_T          ppszIdentifier);
```

Parameter	Type	Description
hMbr	input	Member handle
ppszIdentifier	output	Member identifier (name or alias)

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

```
ESS_FUNC_M EssMdxGetMbrProperty(
   ESS_MDX_MBRHDL_T        hMbr,
   ESS_MDX_PROPHDL_T       hProp,
   ESS_MDX_PPROPVALUE_T    pPropValue);
```

Parameter	Type	Description
hMbr	input	Member handle
hProp	input	Property handle

Parameter	Type	Description
pPropValue	output	Property value

EssMdxGetNamedSets

Returns the named sets in the query.

Syntax

```
ESS_FUNC_M EssMdxGetNamedSets(
   ESS_HCTX_T     hCtx,
   ESS_PULONG_T   pulCount,
   ESS_PPSTR_T    ppNames,
   ESS_PLONG_T    *ppTypes);
```

Parameter	Type	Description
hCtx	input	Context handle.
pulCount	output	Count of the named sets returned in the query.
ppNames	output	An array of named sets. The memory allocated for *ppNames* should be freed using **EssFree()**.
*ppTypes	output	Pointer to the named set type: ESS_MDX_NAMEDSET_TYPE_SESSION.

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

```
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++)
```

```
    {
        fileHandle = 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);
```

Parameter	Type	Description
hProp	input	Property handle
ppszName	output	Property name

Parameter	Type	Description
pPropType	output	Property type: • ESS_MDX_PROPTYPE_BOOL • ESS_MDX_PROPTYPE_DOUBLE • ESS_MDX_PROPTYPE_STRING • ESS_MDX_PROPTYPE_DATETIME • ESS_MDX_PROPTYPE_ULONG

EssMdxGetQueryCellProperties

Returns the cell properties in effect for this query.

Syntax

```
ESS_FUNC_M    EssMdxGetQueryCellProperties(
   ESS_MDX_QRYHDL_T      hQry,
   ESS_MDX_CELLPROPS_T   pulProp);
```

Parameter	Type	Description
hQry	input	Query handle
pulProp	output	Pointer to bitmask specifying what cell properties are returned

See Also

"EssMdxSetQueryCellProperties" on page 1604

EssMdxGetQueryOptions

Returns the query options in effect for the current query.

Syntax

```
ESS_FUNC_M    EssMdxGetQueryOptions(
   ESS_MDX_QRYHDL_T    hQry,
   ESS_MDX_PQRYOPT_T   pulOpt);
```

Parameter	Type	Description
hQry	input	Query handle
pulOpt	output	Pointer to bitmask specifying current query options in effect

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

```
ESS_FUNC_M EssMdxGetSmartlistforCell(
   ESS_MDX_QRYHDL_T    hQry,
   ESS_MDX_CELLHDL_T   hCell,
   ESS_PSTR_T          pSmartlist);
```

Parameter	Type	Description
hQry	input	Query handle
hCell	input	Cell handle
pSmartlist	output	Name of the Smartlist object a cell is associated with

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

```
ESS_FUNC_M EssMdxGetValue(
   ESS_MDX_CELLHDL_T   hCell,
   ESS_PBOOL_T         pbIsMissing,
   ESS_PBOOL_T         pbNoAccess,
   ESS_PDOUBLE_T       pdValue);
```

Parameter	Type	Description
hCell	input	Cell handle
pbIsMissing	output	Whether cell value is #Missing
pbNoAccess	output	Whether cell value is #NoAccess

Parameter	Type	Description
pdValue	output	The cell's value, if not #Missing

EssMDXIsCellGLDrillable

Checks whether the cell is associated with a drill-through URL.

Syntax

```
ESS_FUNC_M EssMdxIsCellGLDrillable (hQry, hCell, pIsDrillable);
```

Parameter	Data Type	Description
hQry	ESS_MDX_QRYHDL_T	Query handle
hCell	ESS_MDX_CELLHDL_T	Cell handle
pIsDrillable	ESS_PBOOL_T	True, if the cell is associated with a drill-through URL; False, otherwise

Return Value

- If successful, sets *pIsDrillable* based on the cell's status.
- If unsuccessful, returns an error message.

Example

```
#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))
```

```
      != ESS_STS_NOERR)
   {
      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

```
ESS_FUNC_M EssMdxNewQuery(
   ESS_HCTX_T           hCtx,
   ESS_STR_T            pszQry,
   ESS_MDX_PQRYHDL_T    phQry);
```

Parameter	Type	Description
hCtx	input	API context handle
pszQry	input	Query text
phQry	output	Query handle

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

```
ESS_FUNC_M EssMdxSetDataLess(
   ESS_MDX_QRYHDL_T    hQry);
```

Parameter	Type	Description
hQry	input	Query handle

EssMDXSetHideData

Converts #NOACCESS cells to #MISSING.

Syntax

```
ESS_FUNC_M EssMDXSetHideData(
   ESS_MDX_QRYHDL_T   hQry);
```

Parameter	Type	Description
hQry	input	Query handle

EssMdxSetMbrIdType

Sets the type of member identifier desired in the result. Defaults to ESS_MDX_MEMBERIDTYPE_NAME.

Syntax

```
ESS_FUNC_M EssMdxSetMbrIdType(
   ESS_MDX_QRYHDL_T          hQry,
   ESS_MDX_MEMBERIDTYPE_T    mbrIdType);
```

Parameter	Type	Description
hQry	input	Query handle
mbrIdType	input	Member identifier desired (name/alias): • ESS_MDX_MEMBERIDTYPE_NAME • ESS_MDX_MEMBERIDTYPE_ALIAS

EssMdxSetNeedCellStatus

Turns on retrieval of cell status information. By default the cell status information is not retrieved.

Syntax

```
ESS_FUNC_M EssMdxSetNeedCellStatus(
   ESS_MDX_QRYHDL_T   hQry);
```

Parameter	Type	Description
hQry	input	Query handle

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

```
ESS_FUNC_M    EssMdxSetQueryCellProperties(
   ESS_MDX_QRYHDL_T      hQry,
   ESS_MDX_CELLPROPS_T   ulProp);
```

Parameter	Type	Description
hQry	input	Query handle
ulProp	input	Bitmask specifying what cell properties should be sent. Values: • ESS_MDX_CELLPROP_FORMATTED_VALUE • ESS_MDX_CELLPROP_FORMAT_STRING • ESS_MDX_CELLPROP_SMLIST_NAME

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

```
ESS_FUNC_M    EssMdxSetQueryOptions(
   ESS_MDX_QRYHDL_T   hQry,
   ESS_MDX_QRYOPT_T   ulOpt);
```

Parameter	Type	Description
hQry	input	Query handle
ulOpt	input	Query options. Bitmask values: • ESS_MDX_QRYOPT_GET_MI_CELLS This option indicates that formatted values should be generated for #Missing cells also. By default #Missing cells are not formatted by server. • ESS_MDX_QRYOPT_GET_ME_CELLS 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.

See Also

"EssMdxGetQueryOptions" on page 1600

MDX Sample Client Program

```
#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,
                             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);
}

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_PULONG_T ulaNProps = NULL;
   ESS_MDX_PPPROPHDL_T haaProps = NULL;
   ESS_MDX_PPROPHDL_T haProps;
   ESS_MDX_PROPHDL_T hProp;
   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;
```

```
}
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);
   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;
               }
               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);
   }
   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);
         }
         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:
```

```
            {
               printf("Ulong Value: %ld\n",
                      propval.value.ulData);
               break;
            }
            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);
```

```
      exit ((int) sts);
   }
   printf("EssFree sts: %ld\n", sts);
}

void ESS_MdxQry()
{
   ESS_STS_T    sts;
   ESS_MDX_QRYHDL_T hQry;
   ESS_ULONG_T ulNAxes, ulNAxisDims, ulAxisSize, ulResultSize;
   ESS_ULONG_T ulNClusters, ulClusterSize, ulNClusterDims;
   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_PMBRHDL_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\n", 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);
```

```
            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;
}
```

27 Welcome to XMLA Reference

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.

28 Working with XMLA

In This Chapter

Key Features 1621
Methods 1621
XMLA Rowsets 1627
Flattened Rowset Examples 1657

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

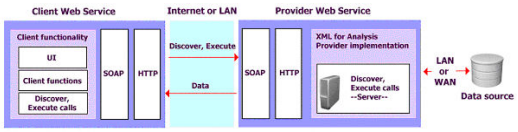
The following methods provide a standard way for XML applications to access basic information from the server. Because these methods are invoked using SOAP, they accept input and deliver output in XML. By default, these methods are stateless, so the server context ends at the completion of any command.

The simplified interface model has two methods.

- Discover
- Execute

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.

Enumeration value	Description
DISCOVER_ DATASOURCES	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.)
DISCOVER_ PROPERTIES	Returns a list of information and values about the requested properties that are supported by the specified data source (provider).
DISCOVER_SCHEMA_ ROWSETS	Returns the names, values, and other information of all supported RequestType enumeration values (including those listed here), and any additional provider-specific enumeration values.
DISCOVER_ ENUMERATORS	Returns a list of names, data types, and enumeration values of enumerators supported by the provider of a specific data source.
DISCOVER_ KEYWORDS	Returns a rowset containing a list of keywords reserved by the provider.
DISCOVER_LITERALS	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 "DISCOVER_SCHEMA_ ROWSETS Rowset."

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.

Result [out]

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:

```
<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_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 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">
     <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: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="CUBE_TYPE" type="xsd:string"
          sql:field="CUBE_TYPE"/>
         <xsd:element name="LAST_SCHEMA_UPDATE" type="xsd:dateTime"
          sql:field="LAST_SCHEMA_UPDATE" minOccurs="0"/>
         <xsd:element name="DESCRIPTION" type="xsd:string"
          sql:field="DESCRIPTION" minOccurs="0"/>
        </xsd:sequence>
       </xsd:complexType>
      </xsd:schema>
      <row>
       <CATALOG_NAME>Demo</CATALOG_NAME>
       <CUBE_NAME>Demo.Basic</CUBE_NAME>
       <CUBE_TYPE>CUBE</CUBE_TYPE>
      </row>
     </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:

```
<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>
 <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>
   <PropertyList>
    <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>
</SOAP-ENV:Envelope>
```

The abbreviated response for the preceding method call:

```
<?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: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:mddataset">
    <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xars="urn:schemas-microsoft-com:xars">
     ...<!-The schema for the data goes here. -- >
    </xsd:schema>
    ... <!-The data in MDDataSet format goes here. -- >
   </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 32 CATALOGS Rowset Structure

Column Name	Essbase Mapping
CATALOG_NAME	Application name
DESCRIPTION	Always null

Request Example

```
<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>DBSCHEMA_CATALOGS</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 (truncated)

```
<?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">
     <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: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>
```

```
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

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 33 MDSCHEMA_CUBES Rowset structure

Column Name	Essbase Mapping
CATALOG_NAME	Application name
CUBE_NAME	Database name
CUBE_TYPE	“CUBE”
LAST_SCHEMA_UPDATE	Time stamp of last outline update
DESCRIPTION	Database description

Request Example

```
<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_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 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">
     <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: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="CUBE_TYPE" type="xsd:string"
         sql:field="CUBE_TYPE"/>
        <xsd:element name="LAST_SCHEMA_UPDATE" type="xsd:dateTime"
         sql:field="LAST_SCHEMA_UPDATE" minOccurs="0"/>
        <xsd:element name="DESCRIPTION" type="xsd:string"
         sql:field="DESCRIPTION" minOccurs="0"/>
       </xsd:sequence>
      </xsd:complexType>
     </xsd:schema>
     <row>
      <CATALOG_NAME>Demo</CATALOG_NAME>
      <CUBE_NAME>Demo.Basic</CUBE_NAME>
      <CUBE_TYPE>CUBE</CUBE_TYPE>
     </row>
    </root>
   </m:return>
  </m:DiscoverResponse>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

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 34 MDSCHEMA_DIMENSIONS Rowset structure

Column Name	Essbase Mapping
CATALOG_NAME	Application name
CUBE_NAME	Database name
DIMENSION_NAME	Dimension name
DIMENSION_UNIQUE_NAME	Dimension name
DIMENSION_CAPTION	Dimension name
DIMENSION_ORDINAL	Dimension number. First dimension is 1, second is 2, and so on
DIMENSION_TYPE	If Essbase dimension type is: ● TIME: MD_DIMTYPE_TIME ● ACCOUNTS: MD_DIMTYPE_MEASURE ● ALL OTHER: MD_DIMTYPE_OTHER
DIMENSION_CARDINALITY	Number of members in the dimension
DEFAULT_HIERARCHY	Dimension name
DESCRIPTION	Comment added for the dimension
DIMENSION_UNIQUE_SETTINGS	2
DIMENSION_IS_VISIBLE	True always

Request Example

```
<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_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>
```

Response Example(truncated)

```
<?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">
     <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: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_NAME" type="xsd:string"
        sql:field="DIMENSION_NAME"/>
       <xsd:element name="DIMENSION_UNIQUE_NAME" type="xsd:string"
        sql:field="DIMENSION_UNIQUE_NAME"/>
       <xsd:element name="DIMENSION_CAPTION" type="xsd:string"
        sql:field="DIMENSION_CAPTION"/>
       <xsd:element name="DIMENSION_ORDINAL" type="xsd:unsignedInt"
        sql:field="DIMENSION_ORDINAL"/>
       <xsd:element name="DIMENSION_TYPE" type="xsd:short"
        sql:field="DIMENSION_TYPE"/>
       <xsd:element name="DIMENSION_CARDINALITY" type="xsd:unsignedInt"
        sql:field="DIMENSION_CARDINALITY"/>
       <xsd:element name="DEFAULT_HIERARCHY" type="xsd:string"
        sql:field="DEFAULT_HIERARCHY"/>
       <xsd:element name="DESCRIPTION" type="xsd:string"
        sql:field="DESCRIPTION" minOccurs="0"/>
       <xsd:element name="DIMENSION_UNIQUE_SETTINGS" type="xsd:int"
        sql:field="DIMENSION_UNIQUE_SETTINGS"/>
       <xsd:element name="DIMENSION_IS_VISIBLE" type="xsd:boolean"
        sql:field="DIMENSION_IS_VISIBLE"/>
      </xsd:sequence>
     </xsd:complexType>
    </xsd:schema>
    <row>
     <CATALOG_NAME>Sample</CATALOG_NAME>
     <CUBE_NAME>Sample.Basic</CUBE_NAME>
```

```
      <DIMENSION_NAME>Year</DIMENSION_NAME>
      <DIMENSION_UNIQUE_NAME>[Year]</DIMENSION_UNIQUE_NAME>
      <DIMENSION_CAPTION>Year</DIMENSION_CAPTION>
      <DIMENSION_ORDINAL>1</DIMENSION_ORDINAL>
      <DIMENSION_TYPE>1</DIMENSION_TYPE>
      <DIMENSION_CARDINALITY>19</DIMENSION_CARDINALITY>
      <DEFAULT_HIERARCHY>[Year]</DEFAULT_HIERARCHY>
      <DIMENSION_UNIQUE_SETTINGS>2</DIMENSION_UNIQUE_SETTINGS>
      <DIMENSION_IS_VISIBLE>true</DIMENSION_IS_VISIBLE>
     </row>
     < ..............More Rows..... >
    </root>
   </m:return>
  </m:DiscoverResponse>
  </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 35describes the rowset structure.

Table 35 MDSCHEMA_FUNCTIONS Rowset structure

Column Name	Essbase Mapping
FUNCTION_NAME	Name of the function
DESCRIPTION	Description of the function
PARAM_LIST	A comma delimited list of parameters
RETURN_TYPE	Always 12
ORIGIN	1 (always:MDX functions)
INTERFACE_NAME	One of the following: Member, Set, Tuple, Numeric, Dimension, Level, Boolean
OBJECT	One of the following values: Set, Member, Tuple, Level, Hierarchy, Dimension
HELP_CONTEXT	Help context ID for the function
CAPTION	Display caption of the function

Request Example

```
<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_FUNCTIONS</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 (truncated)

```
<?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">
     <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:complexType name="row">
       <xsd:sequence maxOccurs="unbounded" minOccurs="0">
        <xsd:element name="FUNCTION_NAME" type="xsd:string"
         sql:field="FUNCTION_NAME"/>
        <xsd:element name="DESCRIPTION" type="xsd:string"
         sql:field="DESCRIPTION"/>
        <xsd:element name="PARAMETER_LIST" type="xsd:string"
         sql:field="PARAMETER_LIST"/>
        <xsd:element name="RETURN_TYPE" type="xsd:int"
         sql:field="RETURN_TYPE"/>
        <xsd:element name="ORIGIN" type="xsd:int"
         sql:field="ORIGIN"/>
        <xsd:element name="INTERFACE_NAME" type="xsd:string"
         sql:field="INTERFACE_NAME"/>
        <xsd:element name="OBJECT" type="xsd:string"
         sql:field="OBJECT" minOccurs="0"/>
        <xsd:element name="HELP_CONTEXT" type="xsd:int"
         sql:field="HELP_CONTEXT" minOccurs="0"/>
        <xsd:element name="CAPTION" type="xsd:string"
```

```
          sql:field="CAPTION"/>
        </xsd:sequence>
       </xsd:complexType>
      </xsd:schema>
      <!-- Begin: All MDX functions that return a Member
       (INTERFACE_NAME=Member) -->
      <row>
       <FUNCTION_NAME>Ancestor</FUNCTION_NAME>
       <DESCRIPTION>Given the input member, returns the ancestor
        at the specified level.</DESCRIPTION>
       <PARAMETER_LIST>Member, Level | Numeric Expression</PARAMETER_LIST>
       <RETURN_TYPE>12</RETURN_TYPE>
       <ORIGIN>1</ORIGIN>
       <INTERFACE_NAME>Member</INTERFACE_NAME>
       <HELP_CONTEXT>9142</HELP_CONTEXT>
       <CAPTION>Ancestor</CAPTION>
      </row>
      < ..............More Rows..............  >
     </root>
    </m:return>
   </m:DiscoverResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

MDSCHEMA_HIERARCHIES Rowset

The HIERARCHIES rowset contains information about the hierarchies available in a dimension.

GUID: MDSCHEMA_HIERARCHIES

Table 36describes the rowset structure.

Table 36 MDSCHEMA_HIERARCHIES Rowset Structure

Column Name	Essbase Mapping
CATALOG_NAME	Application name
CUBE_NAME	Database name
DIMENSION_UNIQUE_NAME	Dimension name
HIERARCHY_NAME	Dimension name
HIERARCHY_UNIQUE_NAME	Dimension name
HIERARCHY_CAPTION	Dimension name
DIMENSION_TYPE	If Essbase dimension type is: ● TIME: MD_DIMTYPE_TIME ● ACCOUNTS: MD_DIMTYPE_MEASSURE ● ALL OTHER: MD_DIMTYPE_OTHER
HIERARCHY_CARDINALITY	Number of members in the dimension

Column Name	Essbase Mapping
DEFAULT_MEMBER	Dimension name
ALL_MEMBER	Dimension name
DESCRIPTION	Dimension comment
STRUCTURE	MD_STRUCTURE_UNBALANCED(2)
HIERARCHY_UNIQUE_SETTINGS	2
HIERARCHY_IS_VISIBLE	True

Request Example

```
<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 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">
     <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: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_NAME" type="xsd:string"
    sql:field="HIERARCHY_NAME"/>
   <xsd:element name="HIERARCHY_UNIQUE_NAME" type="xsd:string"
    sql:field="HIERARCHY_UNIQUE_NAME"/>
   <xsd:element name="HIERARCHY_CAPTION" type="xsd:string"
    sql:field="HIERARCHY_CAPTION"/>
   <xsd:element name="DIMENSION_TYPE" type="xsd:short"
    sql:field="DIMENSION_TYPE"/>
   <xsd:element name="HIERARCHY_CARDINALITY" type="xsd:unsignedInt"
    sql:field="HIERARCHY_CARDINALITY"/>
   <xsd:element name="DEFAULT_MEMBER" type="xsd:string"
    sql:field="DEFAULT_MEMBER"/>
   <xsd:element name="ALL_MEMBER" type="xsd:string"
    sql:field="ALL_MEMBER"/>
   <xsd:element name="DESCRIPTION" type="xsd:string"
    sql:field="DESCRIPTION" minOccurs="0"/>
   <xsd:element name="STRUCTURE" type="xsd:int"
    sql:field="STRUCTURE"/>
   <xsd:element name="HIERARCHY_UNIQUE_SETTINGS" type="xsd:int"
    sql:field="HIERARCHY_UNIQUE_SETTINGS"/>
   <xsd:element name="HIERARCHY_IS_VISIBLE" type="xsd:boolean"
    sql:field="HIERARCHY_IS_VISIBLE"/>
  </xsd:sequence>
 </xsd:complexType>
</xsd:schema>
<row>
 <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>
```

```
     </row>
    </root>
   </m:return>
  </m:DiscoverResponse>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

MDSCHEMA_MEASURES Rowset

The MEASURES rowset contains information about the available measures.

GUID: MDSCHEMA_MEASURES

Table 37describes the rowset structure.

Table 37 MDSCHEMA_MEASURES Rowset Structure

Column Name	Essbase Mapping
CATALOG_NAME	Application name
CUBE_NAME	Database name
MEASURE_NAME	Member names in the Accounts dimension
MEASURE_UNIQUE_NAME	Above member name
MEASURE_CAPTION	Above member name
MEASURE_AGGREGATOR	Essbase ADDITION: 1 Essbase SUBSTRACTION: 17 Essbase MULTIPLICATION:18 Essbase DIVISION:19 Essbase PERCENT:20 Essbase NOOP: 21
DESCRIPTION	Member comment
DATA_TYPE	5
EXPRESSION	Member formula
MEASURE_IS_VISIBLE	True

Request Example

```
<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>
```

```
     <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"?>
<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">
     <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: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="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:element name="DATA_TYPE" type="xsd:unsignedShort"
         sql:field="DATA_TYPE"/>
        <xsd:element name="NUMERIC_PRECISION" type="xsd:unsignedShort"
         sql:field="NUMERIC_PRECISION"/>
        <xsd:element name="NUMERIC_SCALE" type="xsd:short"
         sql:field="NUMERIC_SCALE"/>
        <xsd:element name="EXPRESSION" type="xsd:string"
         sql:field="EXPRESSION" minOccurs="0"/>
        <xsd:element name="MEASURE_IS_VISIBLE" type="xsd:boolean"
         sql:field="MEASURE_IS_VISIBLE"/>
       </xsd:sequence>
      </xsd:complexType>
     </xsd:schema>
     <row>
      <CATALOG_NAME>Sample</CATALOG_NAME>
      <CUBE_NAME>Sample.Basic</CUBE_NAME>
      <MEASURE_NAME>Measures</MEASURE_NAME>
      <MEASURE_UNIQUE_NAME>[Measures]</MEASURE_UNIQUE_NAME>
      <MEASURE_CAPTION>Measures</MEASURE_CAPTION>
      <MEASURE_AGGREGATOR>0</MEASURE_AGGREGATOR>
      <DATA_TYPE>5</DATA_TYPE>
      <NUMERIC_PRECISION>0</NUMERIC_PRECISION>
      <NUMERIC_SCALE>0</NUMERIC_SCALE>
      <MEASURE_IS_VISIBLE>true</MEASURE_IS_VISIBLE>
     </row>
     < ..............More Rows..............  >
    </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 38 MDSCHEMA_MEMBERS Rowset Structure

Column Name	Essbase Mapping
CATALOG_NAME	Application name
CUBE_NAME	Database name
DIMENSION_UNIQUE_NAME	Dimension name
HIERARCHY_UNIQUE_NAME	Dimension name
LEVEL_UNIQUE_NAME	Level name
LEVEL_NUMBER	Level number

Column Name	Essbase Mapping
GENERATION_NUMBER	Generation number
MEMBER_ORDINAL	Member number
MEMBER_NAME	Member name
MEMBER_UNIQUE_NAME	Unique member name
MEMBER_TYPE	1 (REGULAR)
MEMBER_CAPTION	Member name
MEMBER_ALIAS	Default alias
CHILDREN_CARDINALITY	Child count
PARENT_LEVEL	Level number of the parent. For dimension, same level number as the dimension level number
PARENT_UNIQUE_NAME	Name of the parent. For dimension, same name as the dimension name
PARENT_COUNT	Always 1
DESCRIPTION	Member comment

Request Example

```
<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>
```

```
     <Format>Tabular</Format>
    </PropertyList>
   </Properties>
  </Discover>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Response Example(truncated)

```
<?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">
       <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: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_UNIQUE_NAME" type="xsd:string"
            sql:field="LEVEL_UNIQUE_NAME"/>
           <xsd:element name="LEVEL_NUMBER" type="xsd:unsignedInt"
            sql:field="LEVEL_NUMBER"/>
           <xsd:element name="GENERATION_NUMBER" type="xsd:unsignedInt"
            sql:field="GENERATION_NUMBER"/>
           <xsd:element name="MEMBER_ORDINAL" type="xsd:unsignedInt"
            sql:field="MEMBER_ORDINAL"/>
           <xsd:element name="MEMBER_NAME" type="xsd:string"
            sql:field="MEMBER_NAME"/>
           <xsd:element name="MEMBER_UNIQUE_NAME" type="xsd:string"
            sql:field="MEMBER_UNIQUE_NAME"/>
           <xsd:element name="MEMBER_TYPE" type="xsd:int"
            sql:field="MEMBER_TYPE"/>
           <xsd:element name="MEMBER_CAPTION" type="xsd:string"
```

```
            sql:field="MEMBER_CAPTION"/>
           <xsd:element name="MEMBER_ALIAS" type="xsd:string"
            sql:field="MEMBER_ALIAS" minOccurs="0"/>
           <xsd:element name="CHILDREN_CARDINALITY" type="xsd:unsignedInt"
            sql:field="CHILDREN_CARDINALITY"/>
           <xsd:element name="PARENT_LEVEL" type="xsd:unsignedInt"
            sql:field="PARENT_LEVEL"/>
           <xsd:element name="PARENT_UNIQUE_NAME" type="xsd:string"
            sql:field="PARENT_UNIQUE_NAME"/>
           <xsd:element name="PARENT_COUNT" type="xsd:unsignedInt"
            sql:field="PARENT_COUNT"/>
           <xsd:element name="DESCRIPTION" type="xsd:string"
            sql:field="DESCRIPTION" minOccurs="0" />
          </xsd:sequence>
         </xsd:complexType>
       </xsd:schema>
       <row>
        <CATALOG_NAME>Sample</CATALOG_NAME>
        <CUBE_NAME>Sample.Basic</CUBE_NAME>
        <DIMENSION_UNIQUE_NAME>[Year]</DIMENSION_UNIQUE_NAME>
        <HIERARCHY_UNIQUE_NAME>[Year]</HIERARCHY_UNIQUE_NAME>
        <LEVEL_UNIQUE_NAME>[Year].Levels(2)</LEVEL_UNIQUE_NAME>
        <LEVEL_NUMBER>2</LEVEL_NUMBER>
        <GENERATION_NUMBER>1</GENERATION_NUMBER>
        <MEMBER_ORDINAL>1</MEMBER_ORDINAL>
        <MEMBER_NAME>Jan</MEMBER_NAME>
        <MEMBER_UNIQUE_NAME>[Jan]</MEMBER_UNIQUE_NAME>
        <MEMBER_TYPE>1</MEMBER_TYPE>
        <MEMBER_CAPTION>Jan</MEMBER_CAPTION>
        <CHILDREN_CARDINALITY>0</CHILDREN_CARDINALITY>
        <PARENT_LEVEL>1</PARENT_LEVEL>
        <PARENT_UNIQUE_NAME>[Qtr1]</PARENT_UNIQUE_NAME>
        <PARENT_COUNT>1</PARENT_COUNT>
       </row>
       < ..............More Rows..... >
      </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 39 MDSCHEMA_PROPERTIES Rowset Structure

Column Name	Essbase Mapping
CATALOG_NAME	Application name
CUBE_NAME	Database name
HIERARCHY_UNIQUE_NAME	Dimension name
LEVEL_UNIQUE_NAME	Dimension name
PROPERTY_TYPE	1 (MDPROP_MEMBER)
PROPERTY_NAME	One of the following: ● For attribute dimension, the name of the dimension is the name of the property ● For UDA, the UDA name ● For aliases, the alias name
PROPERTY_CAPTION	One of the following: ● For attribute dimensions, the attribute dimension name ● For UDA, the UDA name ● For aliases, the alias name
DATA_TYPE	1 (double) – attribute dimension 2 (boolean) – attribute dimension 3 (string) – attribute dimension, UDA or alias 4 (integer) – attribute dimension
CHARACTER_MAXIMUM_LENGTH	80 (for UDA or an attribute dimension) 30 (for alias)
CHARACTER_OCTET_LENGTH	320 (for UDA or an attribute dimension) 120 (for alias)
PROPERTY_CONTENT_TYPE	0 (MD_PROPTYPE_REGULAR)
SQL_COLUMN_NAME	One of the following: ● For attribute dimensions, the attribute dimension name ● For UDA, the UDA name ● For aliases, the alias name
PROPERTY_ORIGIN	1 (MD_USER_DEFINED)
PROPERTY_ATTRIBUTE_HIERARCHY_NAME	For attribute dimensions, the attribute dimension name
PROPERTY_CARDINALITY	ONE (for UDA and aliases) MANY (for attribute dimension)

Column Name	Essbase Mapping
PROPERTY_IS_VISIBLE	True

Request Example

```
<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 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">
      <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: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_UNIQUE_NAME" type="xsd:string"
    sql:field="LEVEL_UNIQUE_NAME" minOccurs ="0"/>
   <xsd:element name="MEMBER_UNIQUE_NAME" type="xsd:string"
    sql:field="MEMBER_UNIQUE_NAME" minOccurs ="0"/>
   <xsd:element name="PROPERTY_TYPE" type="xsd:short"
    sql:field="PROPERTY_TYPE" minOccurs ="0"/>
   <xsd:element name="PROPERTY_NAME" type="xsd:string"
    sql:field="PROPERTY_NAME" minOccurs ="0"/>
   <xsd:element name="PROPERTY_CAPTION" type="xsd:string"
    sql:field="PROPERTY_CAPTION" minOccurs ="0"/>
   <xsd:element name="DATA_TYPE" type="xsd:unsignedShort"
    sql:field="DATA_TYPE" minOccurs ="0"/>
   <xsd:element name="CHARACTER_MAXIMUM_LENGTH"
    type="xsd:unsignedInt"
    sql:field="CHARACTER_MAXIMUM_LENGTH" minOccurs ="0"/>
   <xsd:element name="CHARACTER_OCTET_LENGTH" type="xsd:unsignedInt"
    sql:field="CHARACTER_OCTET_LENGTH" minOccurs ="0"/>
   <xsd:element name="NUMERIC_PRECISION" type="xsd:unsignedShort"
    sql:field="NUMERIC_PRECISION" minOccurs ="0"/>
   <xsd:element name="NUMERIC_SCALE" type="xsd:short"
    sql:field="NUMERIC_SCALE" minOccurs ="0"/>
   <xsd:element name="DESCRIPTION" type="xsd:string"
    sql:field="DESCRIPTION" minOccurs ="0"/>
   <xsd:element name="PROPERTY_CONTENT_TYPE" type="xsd:short"
    sql:field="PROPERTY_CONTENT_TYPE" minOccurs ="0"/>
   <xsd:element name="SQL_COLUMN_NAME" type="xsd:string"
    sql:field="SQL_COLUMN_NAME" minOccurs ="0"/>
   <xsd:element name="LANGUAGE" type="xsd:unsignedShort"
    sql:field="LANGUAGE" minOccurs ="0"/>
   <xsd:element name="PROPERTY_ORIGIN" type="xsd:unsignedShort"
    sql:field="PROPERTY_ORIGIN" minOccurs ="0"/>
   <xsd:element name="PROPERTY_ATTRIBUTE_HIERARCHY_NAME"
    type="xsd:string"
    sql:field="PROPERTY_ATTRIBUTE_HIERARCHY_NAME" minOccurs ="0"/>
   <xsd:element name="PROPERTY_CARDINALITY" type="xsd:string"
    sql:field="PROPERTY_CARDINALITY" minOccurs ="0"/>
   <xsd:element name="MIME_TYPE" type="xsd:string"
    sql:field="MIME_TYPE" minOccurs ="0"/>
   <xsd:element name="PROPERTY_IS_VISIBLE" type="xsd:boolean"
    sql:field="PROPERTY_IS_VISIBLE" minOccurs ="0"/>
  </xsd:sequence>
 </xsd:complexType>
</xsd:schema>
<row>
 <CATALOG_NAME>Sample</CATALOG_NAME>
```

```
        <CUBE_NAME>Sample.Basic</CUBE_NAME>
        <DIMENSION_UNIQUE_NAME>[Product]</DIMENSION_UNIQUE_NAME>
        <HIERARCHY_UNIQUE_NAME>[Product]</HIERARCHY_UNIQUE_NAME>
        <LEVEL_UNIQUE_NAME>[Product]</LEVEL_UNIQUE_NAME>
        <PROPERTY_TYPE>1</PROPERTY_TYPE>
        <PROPERTY_NAME>Caffeinated</PROPERTY_NAME>
        <PROPERTY_CAPTION>Caffeinated</PROPERTY_CAPTION>
        <DATA_TYPE>2</DATA_TYPE>
        <PROPERTY_CONTENT_TYPE>0</PROPERTY_CONTENT_TYPE>
        <SQL_COLUMN_NAME>Caffeinated</SQL_COLUMN_NAME>
        <PROPERTY_ORIGIN>1</PROPERTY_ORIGIN>
        <PROPERTY_ATTRIBUTE_HIERARCHY_NAME>Caffeinated
        </PROPERTY_ATTRIBUTE_HIERARCHY_NAME>
        <PROPERTY_CARDINALITY>MANY</PROPERTY_CARDINALITY>
        <PROPERTY_IS_VISIBLE>true</PROPERTY_IS_VISIBLE>
       </row>
       < ..............More Rows..... >
      </root>
     </m:return>
    </m:DiscoverResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

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 40describes the rowset structure.

Table 40 MDSCHEMA_SETS Rowset Structure

Column Name	Essbase Mapping
CATALOG_NAME	Application name
CUBE_NAME	Database name
SET_NAME	Name of the set
SCOPE	Session

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 41 MDSCHEMA_LEVELS Rowset Structure

Column Name	Essbase Mapping
CATALOG_NAME	Application name
CUBE_NAME	Database name
DIMENSION_UNIQUE_NAME	Name of the dimension to which the level belongs
HIERARCHY_UNIQUE_NAME	Name of the dimension to which the level belongs
LEVEL_NAME	Unique level name
LEVEL_UNIQUE_NAME	Unique level name
LEVEL_CAPTION	Level name
LEVEL_NUMBER	Level number
LEVEL_CARDINALITY	Number of members in the level
LEVEL_TYPE	MDLEVEL_TYPE_ALL (for dimension level) MDLEVEL_TYPE_TIME (for dimension type TIME) MDLEVEL_TYPE_REGULAR (for all others)
LEVEL_UNIQUE_SETTINGS	2 (MDDIMENSIONS_MEMBER_NAME_UNIQUE)
LEVEL_IS_VISIBLE	True
ESSBASE_GEN_UNIQUE_NAME	Generation unique name
ESSBASE_GEN_CAPTION	Generation caption

Request Example

```
<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>
```

```
  </Discover>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

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"
    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">
     <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: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:element name="ESSBASE_GEN_UNIQUE_NAME" type="xsd:string"
                sql:field="ESSBASE_GEN_UNIQUE_NAME"/>
            <xsd:element name="ESSBASE_GEN_CAPTION" type="xsd:string"
                sql:field="ESSBASE_GEN_CAPTION"/>
          </xsd:sequence>
         </xsd:complexType>
        </xsd:schema>
        <row>
         <CATALOG_NAME>Sample</CATALOG_NAME>
         <CUBE_NAME>Sample.Basic</CUBE_NAME>
         <DIMENSION_UNIQUE_NAME>[Year]</DIMENSION_UNIQUE_NAME>
         <HIERARCHY_UNIQUE_NAME>[Year]</HIERARCHY_UNIQUE_NAME>
         <LEVEL_NAME>[Year].Levels(2)</LEVEL_NAME>
         <LEVEL_UNIQUE_NAME>[Year].Levels(2)</LEVEL_UNIQUE_NAME>
         <LEVEL_CAPTION>[Year].Level 2</LEVEL_CAPTION>
         <LEVEL_NUMBER>2</LEVEL_NUMBER>
         <LEVEL_CARDINALITY>12</LEVEL_CARDINALITY>
         <LEVEL_TYPE>4</LEVEL_TYPE>
         <LEVEL_UNIQUE_SETTINGS>2</LEVEL_UNIQUE_SETTINGS>
         <LEVEL_IS_VISIBLE>true</LEVEL_IS_VISIBLE>
               <ESSBASE_GEN_UNIQUE_NAME>[Year].[Months]</ESSBASE_GEN_UNIQUE_NAME>
               <ESSBASE_GEN_CAPTION>[Year].Months</ESSBASE_GEN_CAPTION>
        </row>
        < ..............More Rows.............. >
       </root>
      </m:return>
     </m:DiscoverResponse>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

DISCOVER_SCHEMA_ROWSETS Rowset

GUID: DISCOVER_SCHEMA_ROWSETS

Table 42describes the rowset structure.

Table 42 DISCOVER_SCHEMA Rowset Structure

Column Name	Essbase Mapping
SchemaName	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.
Restrictions	List of restrictions allowed
Description	Description of the schema

DISCOVER_DATASOURCES Rowset

GUID: DISCOVER_DATASOURCES

Table 43 describes the rowset structure.

Table 43 DISCOVER_DATASOURCES Rowset Structure

Column Name	Essbase Mapping
DataSourceName	Name of the data source
DataSourceDescription	Description of the data source
DataSourceInfo	Provider=Essbase Data Source= name of the Analytic Server
ProviderName	XMLA for Essbase
ProviderType	MDP
AuthenticationMode	Authenticated

DISCOVER_PROPERTIES Rowset

GUID: DISCOVER_PROPERTIES

Table 44 describes the rowset structure.

Table 44 DISCOVER_PROPERTIES Rowset Structure

Column Name	Essbase Mapping
PropertyName	Name of the property
PropertyDescription	Description of the property
PropertyType	XML data type of the property.
PropertyAccessType	Access for the property. The value can be Read, Write, or ReadWrite
IsRequired	True if a property is required, false if it is not required
Value	Current value of the property

Request Example

```
<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>
```

```
   </Properties>
  </Discover>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

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"
    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">
     <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: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>
```

```
     < ..............More Rows............. >
    </root>
   </m:return>
  </m:DiscoverResponse>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

DISCOVER_ENUMERATORS Rowset

GUID: DISCOVER_ENUMERATORS

Table 45 describes the rowset structure.

Table 45 DISCOVER_ENUMERATORS Rowset Structure

Column Name	Essbase Mapping
EnumName	Name of the enumerator that contains a set of values
EnumDescription	Description of the enumerator
ElementName	Name of one of the value elements in the enumerator set Example: TDP
ElementDescription	Description of the element
EnumType	Data type of the Enum values
ElementValue	Value of the element Example: 01

Request Example

```
<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 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">
     <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:complexType  name="row">
       <xsd:sequence maxOccurs="unbounded" minOccurs="0">
        <xsd:element name="EnumName" type="xsd:string"
         sql:field="EnumName"/>
        <xsd:element name="EnumDescription" type="xsd:string"
         sql:field="EnumDescription" minOccurs="0"/>
        <xsd:element name="ElementName" type="xsd:string"
         sql:field="ElementName"/>
        <xsd:element name="ElementDescription" type="xsd:string"
         sql:field="ElementDescription" minOccurs="0"/>
        <xsd:element name="ElementValue" type="xsd:string"
         sql:field="ElementValue" minOccurs="0"/>
        <xsd:element name="EnumType" type="xsd:string"
         sql:field="EnumType"/>
       </xsd:sequence>
      </xsd:complexType>
     </xsd:schema>
     <row>
      <EnumName>ProviderType</EnumName>
      <ElementName>TDP</ElementName>
      <EnumType>string</EnumType>
     </row>
     < ..............More Rows..... >
    </root>
   </m:return>
  </m:DiscoverResponse>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

DISCOVER_KEYWORDS Rowset

GUID: DISCOVER_KEYWORDS

Table 46 describes the rowset structure.

Table 46 DISCOVER_KEYWORDS Rowset Structure

Column Name	Essbase Mapping
Keyword	A list of keywords reserved by a provider Example: AND

Request Example

```
<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 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">
     <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:complexType name="row">
       <xsd:sequence maxOccurs="unbounded" minOccurs="0">
        <xsd:element name="Keyword" type="xsd:string"
         sql:field="Keyword"/>
       </xsd:sequence>
      </xsd:complexType>
     </xsd:schema>
     <row><Keyword>aggregate</Keyword></row>
     <row><Keyword>ancestors</Keyword></row>
     < ..............More Rows..... >
    </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

Column Name	Essbase Mapping
LiteralName	Name of the literal described in the row Example: DBLITERAL_LIKE_PERCENT
LiteralValue	Contains the literal value 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 "%."
LiteralInvalidChars	Characters, in the literal, that are not valid Example: If table names can contain anything other than a numeric character, this string would be "0123456789"
LiteralInvalidStartingChars	Characters that are not valid as the first character of the literal. If the literal can start with any valid character, this is null.
LiteralMaxLength	Maximum number of characters in the literal. If there is no maximum or the maximum is unknown, the value is -1.

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.

```
SELECT NON EMPTY {[Profit]} ON COLUMNS,
NON EMPTY [Product].Levels(1).ALLMEMBERS ON ROWS
FROM Sample.Basic
```

This query has the following result:

[Product].[Family].[MEMBER_CAPTION]	[Profit]
100	30468
200	27954
300	25799
400	21301
Diet	28826

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.

```
SELECT NON EMPTY {[Profit] } ON COLUMNS,
NON EMPTY [Product].Levels(2).ALLMEMBERS ON ROWS
FROM Sample.Basic
```

This query has the following result (truncated):

[Product].[Family].[MEMBER_CAPTION]	[Product].[SKU].[MEMBER_CAPTION]	[Profit]
100	**100-10**	22777
100	**100-20**	5708
100	**100-30**	1983
200	**200-10**	7201
200	**200-20**	12025
200	**200-30**	4636
200	**200-40**	4092
...	**...**	...

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.

```
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):

[Product]. [Family]. [MEMBER_ UNIQUE_ NAME]	[Product]. [Family]. LEVEL_ NUMBER	[Product]. [SKU]. [MEMBER_ UNIQUE_ NAME]	[Product]. [SKU]. LEVEL_ NUMBER	[Profit]
[100]	1	[100-10]	2	22777
[100]	1	[100-20]	2	5708
[100]	1	[100-30]	2	1983
[200]	1	[200-10]	2	7201
[200]	1	[200-20]	2	12025
[200]	1	[200-30]	2	4636
[200]	1	[200-40]	2	4092
[300]	1	[300-10]	2	12195
[300]	1	[300-20]	2	2511
[300]	1	[300-30]	2	2511

[Product]. [Family]. [MEMBER_ UNIQUE_ NAME]	[Product]. [Family]. LEVEL_ NUMBER	[Product]. [SKU]. [MEMBER_ UNIQUE_ NAME]	[Product]. [SKU]. LEVEL_ NUMBER	[Profit]
...	...	...	...	...

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).

```
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):

[Market]. Levels(1). [MEMBER_CAPTION]	[Product]. [Family]. [MEMBER_CAPTION]	[Profit]
East	**Colas**	12656
East	**Root Beer**	2534
East	**Cream Soda**	2627
East	**Fruit Soda**	6344
East	**Diet Drinks**	2408
West	**Colas**	3549
West	**Root Beer**	9727
West	**Cream Soda**	10731
West	**Fruit Soda**	5854
West	**Diet Drinks**	8087
...	...	...

Example 5

In this example, CrossJoin is used to request levels 1–2 for Market and Product.

```
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):

[Market]. Levels(1). [MEMBER_ CAPTION]	[Market]. Levels(2). [MEMBER_ CAPTION]	[Product]. [Family]. [MEMBER_ CAPTION]	[Product]. [SKU]. [MEMBER_ CAPTION]	[Profit]
East	**New York**	**Colas**	**Cola**	3498
East	**New York**	**Root Beer**	**Old Fashioned**	-2594
East	**New York**	**Root Beer**	**Birch Beer**	3086
East	**New York**	**Cream Soda**	**Dark Cream**	2496
East	**New York**	**Cream Drinks**	**Vanilla Cream**	-1952
East	**New York**	**Fruit Soda**	**Grape**	1329
East	**New York**	**Fruit Soda**	**Orange**	1388
East	**New York**	**Fruit Soda**	**Strawberry**	951
...	**...**	**...**	**...**	...

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):

[Market]. Levels(1). [MEMBER_ CAPTION]	[Market]. Levels(1). [LEVEL_ NUMBER]	[Product]. [Family]. [MEMBER_ CAPTION]	[Market]. Levels(1). [LEVEL_ NUMBER]	[Product]. [SKU]. [MEMBER_ CAPTION]	[Market]. Levels(1). [LEVEL_ NUMBER]	[Profit]
East	1	**Colas**	1	**Cola**	2	11129
East	1	**Colas**	1	**Diet Cola**	2	1114
East	1	**Colas**	1	**Caffeine Free Cola**	2	413
East	1	**Root Beer**	1	**Old Fashioned**	2	-2540
East	1	**Root Beer**	1	**Diet Root Beer**	2	982
East	1	**Root Beer**	1	**Birch Beer**	2	4092
East	1	**Cream Soda**	1	**Dark Cream**	2	3233
East	1	**Cream Soda**	1	**Vanilla Cream**	2	-918

[Market]. Levels(1). [MEMBER_ CAPTION]	[Market]. Levels(1). [LEVEL_ NUMBER]	[Product]. [Family]. [MEMBER_ CAPTION]	[Market]. Levels(1). [LEVEL_ NUMBER]	[Product]. [SKU]. [MEMBER_ CAPTION]	[Market]. Levels(1). [LEVEL_ NUMBER]	[Profit]
...	...	...	...	...	...	...

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):

[Market]. Levels(1). [MEMBER_ CAPTION]	[Product]. [Family]. [MEMBER_ CAPTION]	[Year]. Levels(1). [MEMBER_ CAPTION]	[Profit]
East	Colas	Qtr1	2747
East	Colas	Qtr2	3352
East	Colas	Qtr3	3740
East	Colas	Qtr4	2817
East	Root Beer	Qtr1	562
East	Root Beer	Qtr2	610
East	Root Beer	Qtr3	372
East	Root Beer	Qtr4	990
...	...	...	...

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.

```
<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>
  <Execute xmlns="urn:schemas-microsoft-com:xml-analysis"
   SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <Command>
    <Statement>
     WITH MEMBER [Year].[calctest] AS '4'
     SELECT NON EMPTY { [Profit] } ON COLUMNS,
     NON EMPTY {[Year].ALLMEMBERS } ON ROWS
     FROM Sample.Basic
    </Statement>
   </Command>
   <Properties>
    <PropertyList>
     <DataSourceInfo>Provider=Essbase;Data Source=localhost
     </DataSourceInfo>
     <Catalog>Sample</Catalog>
     <Format>Tabular</Format>
     <AxisFormat>TupleFormat</AxisFormat>
    </PropertyList>
   </Properties>
  </Execute>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

An example of a flattened rowset response:

```
<?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: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="row" />
          </xsd:sequence>
         </xsd:complexType>
        </xsd:element>
        <xsd:complexType name="row">
         <xsd:sequence minOccurs="0" maxOccurs="unbounded">
          <xsd:element name="column1" type="xsd:string"
           sql:field="[Year].Levels(1).[MEMBER_CAPTION]" minOccurs="0"/>
          <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:schema>

<row>
 <column3>105522.000000</column3>
</row>
<row>
 <column1>Qtr1</column1>
 <column3>24703.000000</column3>
</row>
<row>
 <column1>Qtr1</column1>
 <column2>Jan</column2>
 <column3>8024.000000</column3>
</row>
<row>
 <column1>Qtr1</column1>
 <column2>Feb</column2>
 <column3>8346.000000</column3>
</row>
<row>
 <column1>Qtr1</column1>
 <column2>Mar</column2>
 <column3>8333.000000</column3>
</row>
<row>
 <column1>Qtr2</column1>
 <column3>27107.000000</column3>
</row>
<row>
 <column1>Qtr2</column1>
 <column2>Apr</column2>
 <column3>8644.000000</column3>
</row>
<row>
 <column1>Qtr2</column1>
 <column2>May</column2>
 <column3>8929.000000</column3>
</row>
<row>
 <column1>Qtr2</column1>
 <column2>Jun</column2>
 <column3>9534.000000</column3>
</row>
<row>
 <column1>Qtr3</column1>
 <column3>27912.000000</column3>
</row>
<row>
 <column1>Qtr3</column1>
 <column2>Jul</column2>
 <column3>9878.000000</column3>
</row>
<row>
 <column1>Qtr3</column1>
 <column2>Aug</column2>
 <column3>9545.000000</column3>
```

```
        </row>
        <row>
         <column1>Qtr3</column1>
         <column2>Sep</column2>
         <column3>8489.000000</column3>
        </row>
        <row>
         <column1>Qtr4</column1>
         <column3>25800.000000</column3>
        </row>
        <row>
         <column1>Qtr4</column1>
         <column2>Oct</column2>
         <column3>8653.000000</column3>
        </row>
        <row>
         <column1>Qtr4</column1>
         <column2>Nov</column2>
         <column3>8367.000000</column3>
        </row>
        <row>
         <column1>Qtr4</column1>
         <column2>Dec</column2>
         <column3>8780.000000</column3>
        </row>
        <row>
         <column1>calctest</column1>
         <column3>4.000000</column3>
        </row>
       </root>
      </m:return>
     </m:ExecuteResponse>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

A API Sample Programs

In This Appendix

Sample C API Program 1 (cs1.c) 1665
Sample C API Program 2 (cs2.c) 1672
Sample C API Program 3 (cs3.c) 1681
Sample Visual Basic API Program 1 (initialize.vbp) 1692
Sample Visual Basic API Program 2 (appdb.vbp) 1695
Sample Visual Basic API Program 3 (reports.vbp) 1703

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 loginvoid
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  ***************/
/**********************************************************/

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\tUsage: ");
     printf("MAINAPI servername username password\n");
     printf("\tDefault: \n\tserver name: local\n\t");
     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_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,"");
   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\r\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\r\n", sts);

   if(!sts)
   {
     printf("\r\nEssbase Application Server - ");
     printf("Version %d.%d.%d\r\n", Release, Version, Revision);

   }
}
```

```
/*************************************************/
/*************************************************/

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");
               }
            }
         }
         EssFree(hInst, pAppsDbs);
      }
      else
         printf("\r\nDatabase List is Empty\r\n\r\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)
      {
         printf ("\r\n-----User List from EssListUsers()-----\r\n\r\n");
         for (ind = 0; ind < Count; ind++)
         {
            printf ("Name->%s\tApplication->%s\tdatabase->%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\r\", count);
         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");
   }
}

/*************************************************/
/*************** MAIN FUNCTION ***********************/

/*
This is the actual program. It initializes and logs with EssAutoLogin,
then gets the Essbase Server version and the version of the API. It
sets the active application and lists the users connected to the
application. The output consists of simple printf statements.
*/
main()
{
   ESS_Init();
   ESS_AutoLogin();

/*
Every Essbase API program must issue EssInit to get the context
```

```
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,
                             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\tUsage:  MAINAPI servername username password\n");
      printf("\tDefault: \n\tserver name: local\n\t");
      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_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\r\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\r\n", sts);

   if(!sts)
   {
     printf("\r\nEssbase Application Server -  ");
     printf("Version %d.%d.%d\r\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("\r\nApplication 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\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");
               }
            }
         }
         EssFree(hInst, pAppsDbs);
      }
      else
         printf("\r\nDatabase List is Empty\r\n\r\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("\r\n----- Results of EssGetDatabaseInfo -----\r\n");
     printf("AppName: %s\r\n",DbInfo->AppName);
      printf("DbName: %s\r\n",DbInfo->Name);
      printf("DbType: %d\r\n",DbInfo->DbType);
      printf("Status: %d\r\n",DbInfo->Status);
      printf("nConnects: %d\r\n",DbInfo->nConnects);
      printf("nLocks: %d\r\n",DbInfo->nLocks);
      printf("nDims: %d\r\n",DbInfo->Data);
      printf("Country: %s\r\n",DbInfo->Country);
      printf("Time: %s\r\n",DbInfo->Time);
      printf("Category: %s\r\n",DbInfo->Category);
      printf("Type: %s\r\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\tApplication->%s\tdatabase->%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);
```

```
    printf("User name:  %s\n",userName);
    printf("Password:  %s\n",pswd);
}

/*****************************************************/
/*************** MAIN FUNCTION ***********************/
/*****************************************************/

void main(int argc, char *argv[])
{

   getCmdLineArgs(argc,argv);

   ESS_Init();
   ESS_AutoLogin();
   ESS_GetVersion();
   ESS_GetAPIVersion();

   ESS_SetActive();
   ESS_ListApplications();
   ESS_ListDatabases();
   ESS_GetDatabaseInfo();

   ESS_ListUsers();
   //ESS_CreateUser();
   //ESS_RenameUser();
   //ESS_DeleteUser();
   //ESS_GetUserInfo ();

   ESS_Logout();
   ESS_Term();
}
/*
End of program
*/
```

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  **********************/
/*********************************************************************/

#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_GetActive();

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\tUsage:  MAINAPI servername username password\n");
      printf("\tDefault: \n\tserver name: local\n\t");
```

```
      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_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\r\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\r\n", sts);

   if(!sts)
   {
     printf("\r\nEssbase Application Server -  ");
     printf("Version %d.%d.%d\r\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");
```

```
               }
            }
         }
         EssFree(hInst, pAppsDbs);
      }
      else
         printf("\r\nDatabase List is Empty\r\n\r\n");
   }
}

/*********************************************************************/
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 \
<QUOTEMBRNAMES \
   {SUPMISSING} \
<BOTTOM ( 4, @DATACOL(1) ) \
<SYM \
<PAGE( 'Measures') \
'Measures' \
<COL( 'Market','Scenario') \
   { OUTALTNAMES } \
<ICHILDREN 'Market' \
'Actual' \
'Budget' \
<ROW( 'Year','Product') \
<ICHILDREN 'Year' \
<DIMBOTTOM '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 = "cdupdtdb.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\r\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 #" & (intMsgLev) _
           & "/message #" & (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
    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>
\Client is the default
  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
  cmdGetActive.Enabled = False
  cmdSetActive.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
  cmdGetActive.Enabled = True
  cmdSetActive.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 the error stack has messages and drop messages in a ListBox
'********************************************************************
  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

'********************************************************************
'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)

'*********************************
'Error Checking and Message display
'*********************************
  If lngStatus = 0 Then
    MsgBox "EsbGetActive() succeeded"

    If Mid$(strAppName, 1, 1) = Chr$(0) Then
      lstMessages "No active application/database is set"
```

```
    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
```

```
      ' 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)

'*********************************
'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
```

```
  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
```

```
'*****************************************************************
'Sends a report specification to the active database from a file
'*****************************************************************
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"

  '*************************************************************************
  'Calls EsbGetString to read the returned data until an empty string is returned
  '*************************************************************************
  Call ESB_GetString

  Else
    MsgBox "EsbReportFile() failed: " & (lngStatus)
  End If
End Sub

'*********************************************************************
'Sends a report specification to the active database as a single string
'*********************************************************************
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

'************************************************************
'Starts sending a report specification to the active database
'************************************************************
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

'************************************************************************
'EsbEndReport marks the end of the report specification sent to the
'active database.
'************************************************************************

Sub ESB_EndReport()

  lngStatus = EsbEndReport(lngCtxHndl)
```

```
'***************
'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

  Dim strErrorsOnLoadFile    As String
  Dim intWhetherAbortOnError As Integer

  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)

    '*******************************
    'Calls error checking sub routine
    '*******************************
    Call ESB_ListErrorStackMsgs
  End If
End Sub

'****************************************************************
'Get the current state of an asynchronous process until it finishes
'****************************************************************
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

'**********************************************************
'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 the error stack has messages and drops messages in a ListBox
'****************************************************************
  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

Sub lstMessages(strItem As String)
    frmRprts.lstMessages.AddItem (strItem)
End Sub

Sub lstMessagesClear()
    frmRprts.lstMessages.Clear
End Sub
```

B 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 ()

/
*****************************************************************************/
/*-------------------------- Example Usage Starts Here -------------------------------*/
/
*****************************************************************************/

/*
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 */

   sts = EssSetSSSecurityMode(hCtx, option, newpassword);

   if(sts)
      printf("Failed to migrate Analytic Services Server to Shared Services mode.
```

```
\n");

   return (sts);
}

/*
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 EssListSSMigrFailedUsers(ESS_HCTX_T,
                                    ESS_PUSHORT_T,
                                    ESS_PPUSERNAME_T);
*/
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\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");

   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");
   }
   else
      printf("Failed to get Shared Services migration failed Groups list.\n");

   return (sts);
}

/*
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_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;

   /* 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_FUNC_M EssSetGroupsToSS(ESS_HCTX_T    hCtx);
*/
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);
}

/*
ESS_FUNC_M EssSetEasLocation (ESS_HCTX_T hCtx,
                              ESS_STR_T EasLocation);
*/
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");

   sts = EssSetEasLocation(hCtx, easLoc);

   if (sts)
      printf("Failed to set EAS Location.\n");

   if (easLoc)
      EssFree(hInst, easLoc);

   return (sts);
}

/*
ESS_FUNC_M EssReRegisterApplication (ESS_HCTX_T hCtx,
                                     ESS_STR_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_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);
}

/
*****************************************************************************/
/*-------------------------- Example Usage Starts Here -------------------------------*/
/
*****************************************************************************/
```

```
/
*****************************************************************************/
/*-------------------------------- Testing API ------------------------------------*/
/
*****************************************************************************/

/*
Declaration of handles and connection information variables
*/


/*****************************************************/
/*************** MAIN FUNCTION ***********************/
/*****************************************************/

main()
{
   ESS_HINST_T      hInst;
   ESS_HCTX_T       hCtx;
   ESS_SVRNAME_T    srvrName   =   "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,
                            0L
                           };

   sts = EssInit(&InitStruct, &hlnst);
   if (sts)
   {
      printf("EssInit failure: %ld\n", sts);
      exit ((int) sts);
   }

   sts = EssLogin(hInst, srvrName, 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);
   sts = EssTerm(hInst);
}
```

C

Limits

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
context handles
  C, 82
  VB, 1121
conventions
  document, 26
coordinate 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

D

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

E

EESS_TRANSACTION_REQSPECIFIC_T, 188
empty strings in Visual Basic, 1119
error codes
  C Grid API, 1115
error handling, 679

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_APPSTATE_T, 1167
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, 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_PARTSLCT_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
EsbClrSpanRelationalSource, 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
EsbLROListObjects, 1371
EsbLROPurgeObjects, 1373
EsbLROUpdateObject, 1374
EsbOtlAddAliasCombination, 1470
EsbOtlAddDimension, 1472
EsbOtlAddMember, 1474
EsbOtlAssociateAttributeDimension, 1478

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
EsbPartitionGetReplCells, 1381
EsbPartitionPurgeOtlChangeFile, 1382
EsbPartitionPutReplCells, 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_BLDDL_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
ESS_FUNC_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

ESS_PARTOTL_MBR_RSRVD_API_T, 164
ESS_PARTOTL_MBRASSOCCHG_API_T, 164
ESS_PARTOTL_MBRATTR_API_T, 164
ESS_PARTOTL_MBRCHG_API_T, 165
ESS_PARTOTL_NAMECHG_API_T, 167
ESS_PARTOTL_NAMED_GENLEV_API_T, 167
ESS_PARTOTL_NAMEMAP_API_T, 167
ESS_PARTOTL_OSN_RELATIVES_API_T, 168
ESS_PARTOTL_QRY_FILTER_T, 170
ESS_PARTOTL_QUERY_T, 169
ESS_PARTOTL_READ_T, 170
ESS_PARTOTL_SELECT_APPLY_T, 171
ESS_PARTOTL_SELECT_CHG_T, 171
ESS_PARTSLCT_T, 171
ESS_PARTSLCT_VALIDATE_T, 172
ESS_PERF_ALLOC_ARG_T, 172
ESS_PERF_ALLOC_ERROR_T, 173
ESS_PERF_ALLOC_T, 174
ESS_PERF_CUSTCALC_T, 177
ESS_PREDICATE_T, 709
ESS_PROCSTATE_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_USERAPPEX_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
ESSBASE.MDB, 39
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_DTREPORT_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

EssGDTRequestDrillThrough, 1064
EssGDTSetInfo, 1065
EssGEndOperation, 1065
EssGetActive, 339
EssGetAlias, 341
EssGetAPIVersion, 342
EssGetApplicationAccess, 342
EssGetApplicationAccessEx, 344
EssGetApplicationInfo, 349
EssGetApplicationInfoEx, 350
EssGetApplicationState, 352
EssGetAssociatedAttributesInfo, 353
EssGetAsyncProcLog, 355
EssGetAsyncProcState, 356
EssGetAttributeInfo, 357
EssGetAttributeSpecifications, 359
EssGetCalcList, 362
EssGetCurrencyRateInfo, 365
EssGetDatabaseAccess, 367
EssGetDatabaseAccessEx, 369
EssGetDatabaseInfo, 374
EssGetDatabaseInfoEx, 375
EssGetDatabaseNote, 377
EssGetDatabaseState, 377
EssGetDatabaseStats, 379
EssGetDefaultCalc, 380
EssGetDimensionInfo, 381
EssGetFilter, 385
EssGetFilterList, 387
EssGetFilterRow, 388
EssGetGlobalState, 389
EssGetGroup, 390
EssGetGroupInfoEx, 391
EssGetGroupList, 395
EssGetGroupListEx, 396
EssGetIBH, 398
EssGetLocalPath, 399
EssGetLogFile, 401
EssGetMemberCalc, 402
EssGetMemberInfo, 404
EssGetObject, 405
EssGetObjectInfo, 407
EssGetProcessState, 407
EssGetServerLocaleString, 409
EssGetServerMode, 410
EssGetSpoolFile, 410
EssGetStatBufSize, 412
EssGetString, 413
EssGetUser, 414
EssGetUserEx, 415
EssGetUserInfoEx, 417
EssGetUserType, 421
EssGetVariable, 421
EssGetVersion, 424
EssGFreeCellLinkResults, 1066
EssGFreeMemberInfo, 1067
EssGFreeRows, 1070
EssGGetAPIContext, 1071
EssGGetAPIInstance, 1072
EssGGetCellLinkResults, 1073
EssGGetDataPointResults, 1074
EssGGetFormattedValue, 1075
EssGGetGridOption, 1079
EssGGetGridPerspective, 1080
EssGGetLinkedPartitionDesc, 1081
EssGGetLRO, 1082
EssGGetLRODesc, 1083
EssGGetMemberInfo, 1084
EssGGetResults, 1085
EssGGetRows, 1086
EssGGetSmartlistforCell, 1087
EssGInit, 1088
EssGLoginSetPass, 1089
EssGNewGrid, 1090
EssGPerformOperation, 1092
EssGSendRows, 1092
EssGSetGridOption, 1093
EssGSetGridPerspective, 1095
EssGSetPath, 1096
EssGTerm, 1097
EssGUnlock, 1098
EssGUpdateLRO, 1099
EssGVersion, 1099
ESSHELP.H, 41
EssImport, 425
EssIncrementalBuildDim, 429
EssInit, 430
EssInit function, 81, 84, 89, 90
EssKillRequest, 432
EssKillRequestEx, 434
EssListAliases, 436
EssListApplications, 437
EssListCalcFunctions, 438
EssListConnections, 442

EssListConnectionsEx, 443
EssListCurrencyDatabases, 447
EssListDatabases, 449
EssListFilters, 457
EssListFilterUsers, 458
EssListGroups, 459
EssListGroupsInfoEx, 460
EssListLocks, 464
EssListLocksEx, 466
EssListLogins, 467
EssListLoginsEx, 468
EssListObjects, 469
EssListRequests, 471
EssListRequestsEx, 473
EssListSpoolFiles, 476
EssListUsers, 481
EssListUsersEx, 483
EssListUsersInfoEx, 485
EssListVariables, 489
EssLoadAlias, 492
EssLoadApplication, 493
EssLoadDatabase, 502
EssLocateIBH, 502
EssLockObject, 503
EssLogin, 504
EssLoginEx, 508
EssLoginSetPassword, 510
EssLogout, 512
EssLogoutUser, 513
EssLogSize, 514
EssLROAddObject, 515
EssLRODeleteCellObjects, 517
EssLRODeleteObject, 519
EssLROGetCatalog, 520
EssLROGetObject, 525
EssLROListObjects, 526
EssLROPurgeObjects, 527
EssLROUpdateObject, 529
EssMdxTrig, 531
EssOtlAddDimension, 721
EssOtlAddMember, 726
EssOtlAssociateAttributeDimension, 729
EssOtlAssociateAttributeMember, 731
EssOtlClearAliasTable, 733
EssOtlClearAliasTableLanguages, 734
EssOtlCloseOutline, 736
EssOtlCompactOutline, 737
EssOtlCopyAliasTable, 739
EssOtlCreateAliasTable, 741
EssOtlDeleteAliasTable, 744
EssOtlDeleteDimension, 745
EssOtlDeleteDTSMemberAlias, 747
EssOtlDeleteGenName, 749
EssOtlDeleteLevelName, 750
EssOtlDeleteMember, 753
EssOtlDeleteMemberAlias, 754
EssOtlDeleteMemberFormula, 755
EssOtlDeleteUserAttribute, 758
EssOtlDisassociateAttributeDimension, 763
EssOtlDisassociateAttributeMember, 764
EssOtlEnableDTSMember, 766
EssOtlFindAlias, 768
EssOtlFindMember, 771
EssOtlFreeMembers, 774
EssOtlFreeStructure, 778
EssOtlGenerateCurrencyOutline, 779
EssOtlGetAliasTableLanguages, 783
EssOtlGetASOCompressionDimension, 785
EssOtlGetAssociatedAttributes, 786
EssOtlGetAttributeAssocLevel, 787
EssOtlGetAttributeInfo, 789
EssOtlGetAttributeSpecifications, 791
EssOtlGetChild, 795
EssOtlGetCountOfDupMemberNameInDim, 796
EssOtlGetDimensionNameUniqueness, 799
EssOtlGetDimensionSolveOrder, 800
EssOtlGetDimensionUserAttributes, 801
EssOtlGetDTSMemberAlias, 803
EssOtlGetEnabledDTSMembers, 805
EssOtlGetFirstMember, 806
EssOtlGetGenName, 807
EssOtlGetGenNameEx, 809
EssOtlGetGenNames, 811
EssOtlGetLevelName, 814
EssOtlGetLevelNameEx, 815
EssOtlGetLevelNames, 817
EssOtlGetLinkedAttributeAttachLevel, 819
EssOtlGetMemberAlias, 821
EssOtlGetMemberCommentEx, 822
EssOtlGetMemberField, 823
EssOtlGetMemberFormula, 826
EssOtlGetMemberInfo, 827
EssOtlGetMemberInfoArray, 828
EssOtlGetMemberLastFormula, 830

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, 863
EssOtlMoveMember, 867
EssOtlNewOutline, 869
EssOtlOpenOutline, 870
EssOtlOpenOutlineEx, 872
EssOtlOpenOutlineQuery, 873
EssOtlQueryAttributes, 877
EssOtlQueryAttributesEx, 879
EssOtlQueryGenerationInfo, 879
EssOtlQueryMembers, 884
EssOtlQueryMembersByName, 888
EssOtlQueryMembersEx, 891
EssOtlQueryMembersExArray, 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
EssPartitionGetAreaLev0CellCount, 543
EssPartitionGetList, 544
EssPartitionGetOtlChanges, 546
EssPartitionGetReplCells, 548
EssPartitionNewDefFile, 549
EssPartitionOpenDefFile, 551
EssPartitionPurgeOtlChangeFile, 552
EssPartitionPutReplCells, 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 include 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

building, 33

I

identity functions
C Main API, 212
Implied Share Setting
C constants, 103
import libraries provided with API, 42
include files, 32
including API files in your program, 32
information flag
C constants, 104
VB constants, 1154
initialization and setup
Grid API, 993
initialization functions
C Main API, 206
VB Main API, 1204
initialization structure, 39, 40
C, 90
initialize.vbp sample program, 1692
initializing the Essbase API
C, 90
installing programs, 43
instance handles
C, 81
VB, 1120
Integration Server
attributes functions
C Main API, 201
VB Main API, 1201
Integration Server drill-through functions, 203
introduction to the API, 25

J

Java API reference, 1583

L

LD_LIBRARY_PATH, 35
level name functions
C Outline API, 715
VB Outline API, 1467
level options
C constants, 695
VB constants, 1457
LIBDIR, 33
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

M

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
optimizing storage dimension categories
C constants, 693
VB constants, 1456
other data types
C, 96
outline administration functions
C Outline API, 717
VB Outline API, 1469
Outline API introduction, 679
outline queries, 680
outline query functions
C Outline API, 718
VB Outline API, 1469
outline verification, 680
VB Outline API, 1446
overview to the API, 25

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

R

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

S

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
VB Outline API, 1447
selecting an active application and database
C, 90
VB, 1128
sequence of Grid API functions, 994
sequence of outline function calls, 682
sequence of outline operation tasks, 682
sequence of typical API tasks
C, 89
VB, 1127
server
architecture with API, 25
server outline queries, 680
VB Outline API, 1446
setup functions
C Outline API, 718
VB Outline API, 1469
share constants
C constants, 696
VB constants, 1455
shared libraries provided with API, 42
Shared Services
API example, migration and user management, 1719
Shared Services functions
C Main API, 214
sharing context handles
C, 82
VB, 1122
SHLIB_PATH, 33
simple data types
C, 95
VB, 1160
size flag
C constants, 106
VB constants, 1154
Solaris programs
building, 33, 35
solutions and problems
C, 93
VB, 1132
sorting options
C constants, 697
VB constants, 1456
static libraries provided with API, 42
Statistics
Performance Buffer, 412

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