# Contents

## Preface

<table>
<thead>
<tr>
<th>Audience</th>
<th>xxiii</th>
</tr>
</thead>
<tbody>
<tr>
<td>Documentation Accessibility</td>
<td>xxiii</td>
</tr>
<tr>
<td>Related Resources</td>
<td>xxiii</td>
</tr>
<tr>
<td>Conventions</td>
<td>xxiv</td>
</tr>
</tbody>
</table>

## 1 Technical Reference Overview

<table>
<thead>
<tr>
<th>About the Technical Reference</th>
<th>1-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>What You Should Know Before You Start</td>
<td>1-1</td>
</tr>
<tr>
<td>Sample Applications</td>
<td>1-1</td>
</tr>
<tr>
<td>Syntax Conventions</td>
<td>1-1</td>
</tr>
<tr>
<td>About Aggregate Storage Databases</td>
<td>1-3</td>
</tr>
</tbody>
</table>

## 2 Calculation Functions

<table>
<thead>
<tr>
<th>Calculation Functions Overview</th>
<th>2-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generations and Levels</td>
<td>2-1</td>
</tr>
<tr>
<td>Abbreviations</td>
<td>2-2</td>
</tr>
<tr>
<td>Function Syntax</td>
<td>2-2</td>
</tr>
<tr>
<td>Function Parameters</td>
<td>2-2</td>
</tr>
<tr>
<td>Calculation Operators</td>
<td>2-5</td>
</tr>
<tr>
<td>Mathematical Operators</td>
<td>2-6</td>
</tr>
<tr>
<td>Conditional and Logical Operators</td>
<td>2-6</td>
</tr>
<tr>
<td>Cross-Dimensional Operators</td>
<td>2-6</td>
</tr>
<tr>
<td>Operation Results on #MISSING Values and Zero (0) Values</td>
<td>2-7</td>
</tr>
<tr>
<td>Calculation Function Categories</td>
<td>2-8</td>
</tr>
<tr>
<td>Boolean Functions</td>
<td>2-8</td>
</tr>
<tr>
<td>Relationship Functions</td>
<td>2-9</td>
</tr>
<tr>
<td>Mathematical Functions</td>
<td>2-10</td>
</tr>
<tr>
<td>Member Set Functions</td>
<td>2-11</td>
</tr>
<tr>
<td>Range and Financial Functions</td>
<td>2-14</td>
</tr>
<tr>
<td>Allocation Functions</td>
<td>2-19</td>
</tr>
<tr>
<td>Function List</td>
<td>Page</td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>Forecasting Functions</td>
<td>2-19</td>
</tr>
<tr>
<td>Statistical Functions</td>
<td>2-20</td>
</tr>
<tr>
<td>Date &amp; Time Function</td>
<td>2-21</td>
</tr>
<tr>
<td>Miscellaneous Functions</td>
<td>2-21</td>
</tr>
<tr>
<td>Calculation Function List</td>
<td>2-21</td>
</tr>
<tr>
<td>@ABS</td>
<td>2-23</td>
</tr>
<tr>
<td>@ACCUM</td>
<td>2-23</td>
</tr>
<tr>
<td>@ALLANCESTORS</td>
<td>2-24</td>
</tr>
<tr>
<td>@ALIAS</td>
<td>2-26</td>
</tr>
<tr>
<td>@ALLOCATE</td>
<td>2-27</td>
</tr>
<tr>
<td>@ANCEST</td>
<td>2-30</td>
</tr>
<tr>
<td>@ANCESTORS</td>
<td>2-32</td>
</tr>
<tr>
<td>@ANCESTVAL</td>
<td>2-33</td>
</tr>
<tr>
<td>@ATTRIBUTE</td>
<td>2-34</td>
</tr>
<tr>
<td>@ATTRIBUTESVAL</td>
<td>2-35</td>
</tr>
<tr>
<td>@ATTRIBUTEBVAL</td>
<td>2-36</td>
</tr>
<tr>
<td>@ATTRIBUTESVAL</td>
<td>2-38</td>
</tr>
<tr>
<td>@AVG</td>
<td>2-39</td>
</tr>
<tr>
<td>@AVGRANGE</td>
<td>2-40</td>
</tr>
<tr>
<td>@BETWEEN</td>
<td>2-41</td>
</tr>
<tr>
<td>@CALCMODE</td>
<td>2-42</td>
</tr>
<tr>
<td>@CHILDREN</td>
<td>2-49</td>
</tr>
<tr>
<td>@COMPOUND</td>
<td>2-50</td>
</tr>
<tr>
<td>@COMPOUNDGROWTH</td>
<td>2-52</td>
</tr>
<tr>
<td>@CONCATENATE</td>
<td>2-53</td>
</tr>
<tr>
<td>@CORRELATION</td>
<td>2-54</td>
</tr>
<tr>
<td>@COUNT</td>
<td>2-57</td>
</tr>
<tr>
<td>@CREATEBLOCK</td>
<td>2-59</td>
</tr>
<tr>
<td>@CURGEN</td>
<td>2-61</td>
</tr>
<tr>
<td>@CURLEV</td>
<td>2-62</td>
</tr>
<tr>
<td>@CURRMBR</td>
<td>2-62</td>
</tr>
<tr>
<td>@CURRMBRANGE</td>
<td>2-64</td>
</tr>
<tr>
<td>@DATEDIFF</td>
<td>2-67</td>
</tr>
<tr>
<td>@DATEPART</td>
<td>2-68</td>
</tr>
<tr>
<td>@DATEROLL</td>
<td>2-70</td>
</tr>
<tr>
<td>@DECLINE</td>
<td>2-71</td>
</tr>
<tr>
<td>@DESCENDANTS</td>
<td>2-73</td>
</tr>
<tr>
<td>@DISCOUNT</td>
<td>2-74</td>
</tr>
<tr>
<td>@ENUMVALUE</td>
<td>2-76</td>
</tr>
<tr>
<td>@EQUAL</td>
<td>2-76</td>
</tr>
<tr>
<td>@EXP</td>
<td>2-77</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>-------------------</td>
<td>------</td>
</tr>
<tr>
<td>@EXPAND</td>
<td>2-78</td>
</tr>
<tr>
<td>@FACTORIAL</td>
<td>2-80</td>
</tr>
<tr>
<td>@FORMATDATE</td>
<td>2-81</td>
</tr>
<tr>
<td>@GEN</td>
<td>2-83</td>
</tr>
<tr>
<td>@GENMBRS</td>
<td>2-83</td>
</tr>
<tr>
<td>@GROWTH</td>
<td>2-84</td>
</tr>
<tr>
<td>@IALANCESTORS</td>
<td>2-86</td>
</tr>
<tr>
<td>@IANCESTORS</td>
<td>2-87</td>
</tr>
<tr>
<td>@ICHALDREN</td>
<td>2-88</td>
</tr>
<tr>
<td>@IDESCEENDANTS</td>
<td>2-89</td>
</tr>
<tr>
<td>@IALANCESTORS</td>
<td>2-90</td>
</tr>
<tr>
<td>@ILANCESDANTS</td>
<td>2-92</td>
</tr>
<tr>
<td>@ILSIBLINGS</td>
<td>2-95</td>
</tr>
<tr>
<td>@INT</td>
<td>2-96</td>
</tr>
<tr>
<td>@INTEREST</td>
<td>2-96</td>
</tr>
<tr>
<td>@INTERSECT</td>
<td>2-98</td>
</tr>
<tr>
<td>@IRDESCENDANTS</td>
<td>2-99</td>
</tr>
<tr>
<td>@IRR</td>
<td>2-101</td>
</tr>
<tr>
<td>@IRREX</td>
<td>2-102</td>
</tr>
<tr>
<td>@IRSIBLINGS</td>
<td>2-104</td>
</tr>
<tr>
<td>@ISACCTYPE</td>
<td>2-105</td>
</tr>
<tr>
<td>@ISANCEST</td>
<td>2-105</td>
</tr>
<tr>
<td>@ISATTRIBUTE</td>
<td>2-106</td>
</tr>
<tr>
<td>@ISCCHILD</td>
<td>2-107</td>
</tr>
<tr>
<td>@ISDESC</td>
<td>2-107</td>
</tr>
<tr>
<td>@ISGEN</td>
<td>2-108</td>
</tr>
<tr>
<td>@ISANCEST</td>
<td>2-109</td>
</tr>
<tr>
<td>@ISIBLINGS</td>
<td>2-109</td>
</tr>
<tr>
<td>@ISICHILD</td>
<td>2-110</td>
</tr>
<tr>
<td>@ISDESC</td>
<td>2-111</td>
</tr>
<tr>
<td>@ISPARENT</td>
<td>2-111</td>
</tr>
<tr>
<td>@ISISIBLING</td>
<td>2-112</td>
</tr>
<tr>
<td>@ISLEV</td>
<td>2-112</td>
</tr>
<tr>
<td>@ISMBR</td>
<td>2-113</td>
</tr>
<tr>
<td>@ISMBRUDA</td>
<td>2-114</td>
</tr>
<tr>
<td>@ISMBRWITHATTR</td>
<td>2-114</td>
</tr>
<tr>
<td>@ISPARENT</td>
<td>2-116</td>
</tr>
<tr>
<td>@ISRANGENEMPTY</td>
<td>2-116</td>
</tr>
<tr>
<td>@ISSAMEGEN</td>
<td>2-117</td>
</tr>
<tr>
<td>@ISSAMELEV</td>
<td>2-118</td>
</tr>
<tr>
<td>@ISSIBLING</td>
<td>2-119</td>
</tr>
</tbody>
</table>
3 Calculation Commands

Calculation Commands Overview 3-1
Calculation Operators 3-1
   Mathematical Operators 3-1
   Conditional and Logical Operators 3-2
   Cross-Dimensional Operator 3-2
Calculation Command Groups 3-3
   Conditional Commands 3-3
   Control Flow Commands 3-4
   Data Declaration Commands 3-4
   Functional Commands 3-4
   Member Formulas 3-5
Calculation Command List 3-6
   & (ampersand) 3-7
   AGG 3-8
   ARRAY 3-8
   CALC ALL 3-10
   CALC AVERAGE 3-10
   CALC DIM 3-11
   CALC FIRST 3-12
   CALC LAST 3-12
   CALC TWOPASS 3-13
   CLEARBLOCK 3-13
   CLEARDATA 3-15
   DATACOPY 3-16

Functions Supported in Hybrid Aggregation Mode 2-267
## Configuration Settings

**Configuration Settings Categorical List**  4-1
- Calculation Configuration Settings  4-1
- Data Import and Export Configuration Settings  4-2
- Memory Management Configuration Settings  4-2
- Logging and Error Handling Configuration Settings  4-2
- Miscellaneous Configuration Settings  4-2
- Partitioning Configuration Settings  4-2
- Ports and Connections Configuration Settings  4-2
- Query Management Configuration Settings  4-3

**Aggregate Storage and Block Storage Settings Comparison**  4-3
- Block Storage and Aggregate Storage Configuration Settings  4-3
- Aggregate Storage Configuration Settings  4-4
- Block Storage Configuration Settings  4-4

**Config Settings List**  4-4
- AGENTTHREADS  4-6
- ASODEFAULTCACHESIZE  4-7
- ASODYNAMICAGGINBSO  4-7
- ASODYNAMICAGGINBSOFOLDERPATH  4-8
- AUDITTRAIL  4-9
- AUTOMERGE  4-9
- AUTOMERGEMAXSLICENCENUMBER  4-10
- CALCCACHE  4-11
- CALCCACHEDEFAULT  4-12
- CALCCACHEHIGH  4-14
- CALCCACHELOW  4-16
- CALCLIMITFORMULARECURSION  4-17
- CALCTRA C  4-18
- CUSTOMCALCANDALLOCTHRUINSERT  4-19
- DATACACHESIZE  4-19
- DLSINGLETHREADPERSTAGE  4-20
- DLTHREADSPREPARE  4-22
- DLTHREADSWRITE  4-23
- DYNCALCCACHEMAXSIZE  4-25
- ENABLERTSVLOGGING  4-26
- FORCEALLDENSECALCON2PASSACCOUNTS  4-27
- FORCESHUTDOWNINTERVAL  4-27
- GRIDEXPANSION  4-27
- GRIDEXPANSIONMESSAGES  4-28
- GRIDSUPPRESSINVALID  4-29
HYBRIDBSOINCALCSCRIPT 4-30
IGNORECONSTANTS 4-31
INDEXCACHESIZE 4-32
MAXFORMULACACHESIZE 4-32
MAXLOGINS 4-34
MAXNUMBEROFACTIVEDB 4-34
MAX_REQUEST_GRID_SIZE 4-35
MAX_RESPONSE_GRID_SIZE 4-36
MDXINSERTBUFFERAGGMETHOD 4-37
MDXINSERTREQUESTTIMEOUT 4-38
MDXQRYGOVCOUNT 4-38
NUMBLOCKSTOEXTEND 4-39
QUERYRESULTLIMIT 4-40
QRYGOVEXECBLK 4-41
QRYGOVEXECTIME 4-42
QUERYTRACE 4-44
QUERYTRACETHRESHOLD 4-45
RENEGADELOG 4-45
RESTRUCTURETHREADS 4-46
RTDEPCALCOPTIMIZE 4-47
SERVERTHREADS 4-48
SSANCESTORONTOP 4-49
SSMEMBERIDPROCESSING 4-49
SSOPTIMIZEDGRIDPROCESSING 4-50
SSPROCROWLIMIT 4-51
SUPNA 4-52
SVRIDLETIME 4-53
TARGETASOOPT 4-53
TARGETTIMESERIESOPT 4-54

MaxL

- Overview of MaxL and MDX 5-1
- How to Read MaxL Railroad Diagrams 5-1
  - Anatomy of MaxL Statements 5-2
  - Railroad Diagram Symbols 5-2
  - Sample Railroad Diagram 5-3
- MaxL Data Definition Language (DDL) 5-4
- MaxL Statements 5-4
  - Performance Statistics in MaxL 5-5
    - The Essbase Performance Statistics Tables 5-5
<table>
<thead>
<tr>
<th>Type</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>MaxL Script Example</td>
<td>5-10</td>
</tr>
<tr>
<td>Listed By Verbs</td>
<td>5-11</td>
</tr>
<tr>
<td>Alter</td>
<td>5-11</td>
</tr>
<tr>
<td>Create</td>
<td>5-11</td>
</tr>
<tr>
<td>Display</td>
<td>5-12</td>
</tr>
<tr>
<td>Drop</td>
<td>5-12</td>
</tr>
<tr>
<td>Execute</td>
<td>5-13</td>
</tr>
<tr>
<td>Export</td>
<td>5-13</td>
</tr>
<tr>
<td>Grant</td>
<td>5-13</td>
</tr>
<tr>
<td>Import</td>
<td>5-13</td>
</tr>
<tr>
<td>Query</td>
<td>5-13</td>
</tr>
<tr>
<td>Refresh</td>
<td>5-14</td>
</tr>
<tr>
<td>Listed by Objects</td>
<td>5-14</td>
</tr>
<tr>
<td>Aggregate Build</td>
<td>5-15</td>
</tr>
<tr>
<td>Aggregate Process</td>
<td>5-15</td>
</tr>
<tr>
<td>Aggregate Selection</td>
<td>5-15</td>
</tr>
<tr>
<td>Allocation</td>
<td>5-15</td>
</tr>
<tr>
<td>Application</td>
<td>5-15</td>
</tr>
<tr>
<td>Archive_file</td>
<td>5-15</td>
</tr>
<tr>
<td>Calculation</td>
<td>5-15</td>
</tr>
<tr>
<td>Data</td>
<td>5-15</td>
</tr>
<tr>
<td>Database</td>
<td>5-15</td>
</tr>
<tr>
<td>Dimensions</td>
<td>5-16</td>
</tr>
<tr>
<td>Drillthrough</td>
<td>5-16</td>
</tr>
<tr>
<td>Filter</td>
<td>5-16</td>
</tr>
<tr>
<td>Group</td>
<td>5-16</td>
</tr>
<tr>
<td>Location Alias</td>
<td>5-16</td>
</tr>
<tr>
<td>Lock</td>
<td>5-16</td>
</tr>
<tr>
<td>LRO</td>
<td>5-16</td>
</tr>
<tr>
<td>Object</td>
<td>5-17</td>
</tr>
<tr>
<td>Outline</td>
<td>5-17</td>
</tr>
<tr>
<td>Partition</td>
<td>5-17</td>
</tr>
<tr>
<td>Privilege</td>
<td>5-17</td>
</tr>
<tr>
<td>Session</td>
<td>5-17</td>
</tr>
<tr>
<td>System</td>
<td>5-17</td>
</tr>
<tr>
<td>Tablespace</td>
<td>5-17</td>
</tr>
<tr>
<td>Trigger</td>
<td>5-18</td>
</tr>
<tr>
<td>Trigger Spool</td>
<td>5-18</td>
</tr>
<tr>
<td>User</td>
<td>5-18</td>
</tr>
<tr>
<td>Variable</td>
<td>5-18</td>
</tr>
<tr>
<td>MaxL Statement Reference</td>
<td>5-18</td>
</tr>
</tbody>
</table>
Display User 5-91
Display Variable 5-94
Drop Application 5-95
Drop Calculation 5-95
Drop Database 5-96
Drop Drillthrough 5-96
Drop Filter 5-97
Drop Location Alias 5-97
Drop Lock 5-98
Drop Object 5-99
Drop Partition 5-99
Drop Trigger 5-101
Drop Trigger Spool 5-101
Execute Calculation 5-101
Execute Aggregate Process (Aggregate Storage) 5-104
Execute Aggregate Build 5-106
Execute Aggregate Selection 5-107
Export Data 5-111
Export LRO 5-113
Export Outline 5-115
Grant 5-118
Import Data 5-120
Import Dimensions 5-122
Import LRO 5-123
Query Application 5-124
Query Archive_File 5-125
Query Database 5-126
Refresh Outline 5-131
Refresh Replicated Partition 5-134
MaxL Definitions 5-135
MaxL Syntax Notes 5-135
Numbers in MaxL Syntax 5-136
Terminals 5-137
ACTION 5-137
ALT-NAME-SINGLE 5-137
APP-NAME 5-138
AREA-ALIAS 5-140
BUFFER-ID 5-141
CALC-NAME 5-141
CALC-NAME-SINGLE 5-142
CALC-STRING 5-142
COLUMN-WIDTH 5-143
COMMENT-STRING 5-143
CONDITION 5-144
CUBE-AREA or MDX-SET 5-144
DATE 5-145
DBS-EXPORT-DIR 5-145
DBS-NAME 5-146
DBS-STRING 5-148
DIM-NAME 5-148
EXPORT-DIR 5-148
FILE-NAME 5-149
FILE-NAME-PREFIX 5-149
FILTER-NAME 5-150
FULL-EXPORT-DIR 5-150
GROUP-NAME 5-151
HOST-NAME 5-152
ID-RANGE 5-152
ID-STRING 5-152
IMPORT-DIR 5-152
IMP-FILE 5-153
LOCATION-ALIAS-NAME 5-154
LOC-ALIAS-SINGLE 5-155
LOG-TIME 5-155
ALLOC-NUMERIC 5-155
MEMBER-EXPRESSION 5-156
MEMBER-NAME 5-157
OBJ-NAME 5-157
OBJ-NAME-SINGLE 5-158
OUTLINE-ID 5-158
PASSWORD 5-158
PATHNAME_FILENAME 5-159
PRECISION-DIGITS 5-159
PROPS 5-159
RNUM 5-160
RTSV-LIST 5-160
RULE-FILE-NAME 5-161
SESSION-ID 5-161
SIZE-STRING 5-162
SPOOL-NAME 5-162
STOPPING-VAL 5-163
TABLSP-NAME 5-163
Privileges and Roles 5-168
   Application-Level System Roles 5-168
   Database-Level System Roles 5-169
Quoting and Special Characters Rules for MaxL Language 5-169
   Tokens enclosed in Single Quotation Marks 5-170
   Tokens Enclosed in Double Quotation Marks 5-170
   Use of Backslashes in MaxL 5-170
   Use of Apostrophes (Single Quotation Marks) 5-171
   Use of Dollar Signs 5-171
MaxL Shell Commands 5-171
   Overview of MaxL Shell 5-172
   MaxL Shell Invocation 5-172
   Prerequisites for Using MaxL 5-172
   MaxL Invocation Summary 5-173
   Interactive Input 5-175
   File Input 5-178
   Standard Input 5-179
   Login 5-180
   LoginAs 5-180
   Encryption 5-181
   Query Cancellation 5-181
MaxL Shell Syntax Rules and Variables 5-181
   Semicolons 5-182
   Variables 5-182
   Quoting and Special Characters Rules for MaxL Shell 5-186
MaxL Shell and Unicode 5-187
MaxL Shell Command Reference 5-187
   Spool on/off 5-188
   Set Display Column Width 5-189
   Set Message Level 5-190
   Set Timestamp 5-191
   Echo 5-191
   Nesting 5-191
   Error Checking and Branching 5-192
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>5-194</td>
</tr>
<tr>
<td>Logout</td>
<td>5-194</td>
</tr>
<tr>
<td>Exit</td>
<td>5-195</td>
</tr>
<tr>
<td>ESSCMD Script Conversion</td>
<td>5-195</td>
</tr>
<tr>
<td>ESSCMD Script Utility Usage</td>
<td>5-195</td>
</tr>
<tr>
<td>Things to Note About the ESSCMD shell Script Utility</td>
<td>5-196</td>
</tr>
<tr>
<td>ESSCMD to MaxL Mapping</td>
<td>5-196</td>
</tr>
<tr>
<td>Reserved Words List</td>
<td>5-203</td>
</tr>
<tr>
<td>MaxL BNF</td>
<td>5-212</td>
</tr>
<tr>
<td>MaxL Statements (Aggregate Storage)</td>
<td>5-234</td>
</tr>
<tr>
<td>Alter Application (Aggregate Storage)</td>
<td>5-235</td>
</tr>
<tr>
<td>Alter Database (Aggregate Storage)</td>
<td>5-239</td>
</tr>
<tr>
<td>Alter System (Aggregate Storage)</td>
<td>5-246</td>
</tr>
<tr>
<td>Create Application (Aggregate Storage)</td>
<td>5-252</td>
</tr>
<tr>
<td>Create Database (Aggregate Storage)</td>
<td>5-253</td>
</tr>
<tr>
<td>Create Outline (Aggregate Storage)</td>
<td>5-255</td>
</tr>
<tr>
<td>Display Tablespace (Aggregate Storage)</td>
<td>5-256</td>
</tr>
<tr>
<td>Execute Allocation</td>
<td>5-257</td>
</tr>
<tr>
<td>Execute Calculation (Aggregate Storage)</td>
<td>5-261</td>
</tr>
<tr>
<td>Export Data (Aggregate Storage)</td>
<td>5-263</td>
</tr>
<tr>
<td>Export Query Tracking (Aggregate Storage)</td>
<td>5-266</td>
</tr>
<tr>
<td>Import Data (Aggregate Storage)</td>
<td>5-267</td>
</tr>
<tr>
<td>Import Query Tracking (Aggregate Storage)</td>
<td>5-271</td>
</tr>
<tr>
<td>Query Application (Aggregate Storage)</td>
<td>5-272</td>
</tr>
<tr>
<td>Query Database (Aggregate Storage)</td>
<td>5-274</td>
</tr>
<tr>
<td>Outline Paging Dimension Statistics</td>
<td>5-283</td>
</tr>
<tr>
<td>Aggregate Storage Runtime Statistics</td>
<td>5-284</td>
</tr>
<tr>
<td>Aggregate Storage Slice Information Output</td>
<td>5-286</td>
</tr>
<tr>
<td>Aggregate Storage Group ID Information Output</td>
<td>5-286</td>
</tr>
<tr>
<td>Aggregate Storage Uncommitted Transaction Information Output</td>
<td>5-287</td>
</tr>
<tr>
<td>MaxL Use Cases</td>
<td>5-287</td>
</tr>
<tr>
<td>Creating an Aggregate Storage Sample Using MaxL</td>
<td>5-287</td>
</tr>
<tr>
<td>Loading Data Using Buffers</td>
<td>5-288</td>
</tr>
<tr>
<td>Using Aggregate Storage Data Load Buffers</td>
<td>5-290</td>
</tr>
<tr>
<td>Forcing Deletion of Partitions</td>
<td>5-291</td>
</tr>
<tr>
<td>Metadata Filtering</td>
<td>5-292</td>
</tr>
<tr>
<td>Examples of Triggers</td>
<td>5-294</td>
</tr>
</tbody>
</table>

### 6 MDX

Overview of MDX 6-1
<table>
<thead>
<tr>
<th>MDX Function List</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abs</td>
<td>6-99</td>
</tr>
<tr>
<td>Aggregate</td>
<td>6-100</td>
</tr>
<tr>
<td>Ancestor</td>
<td>6-102</td>
</tr>
<tr>
<td>Ancestors</td>
<td>6-104</td>
</tr>
<tr>
<td>Attribute</td>
<td>6-105</td>
</tr>
<tr>
<td>AttributeEx</td>
<td>6-106</td>
</tr>
<tr>
<td>Avg</td>
<td>6-107</td>
</tr>
<tr>
<td>BottomCount</td>
<td>6-109</td>
</tr>
<tr>
<td>BottomPercent</td>
<td>6-110</td>
</tr>
<tr>
<td>BottomSum</td>
<td>6-112</td>
</tr>
<tr>
<td>Case</td>
<td>6-113</td>
</tr>
<tr>
<td>CellValue</td>
<td>6-116</td>
</tr>
<tr>
<td>Children</td>
<td>6-118</td>
</tr>
<tr>
<td>ClosingPeriod</td>
<td>6-119</td>
</tr>
<tr>
<td>CoalesceEmpty</td>
<td>6-121</td>
</tr>
<tr>
<td>Concat</td>
<td>6-122</td>
</tr>
<tr>
<td>Contains</td>
<td>6-122</td>
</tr>
<tr>
<td>Count</td>
<td>6-123</td>
</tr>
<tr>
<td>Cousin</td>
<td>6-124</td>
</tr>
<tr>
<td>CrossJoin</td>
<td>6-126</td>
</tr>
<tr>
<td>CrossJoinAttribute</td>
<td>6-128</td>
</tr>
<tr>
<td>CurrentAxisMember</td>
<td>6-129</td>
</tr>
<tr>
<td>CurrentMember</td>
<td>6-130</td>
</tr>
<tr>
<td>CurrentTuple</td>
<td>6-131</td>
</tr>
<tr>
<td>DateDiff</td>
<td>6-132</td>
</tr>
<tr>
<td>DatePart</td>
<td>6-133</td>
</tr>
<tr>
<td>DateRoll</td>
<td>6-135</td>
</tr>
<tr>
<td>DateToMember</td>
<td>6-136</td>
</tr>
<tr>
<td>DefaultMember</td>
<td>6-137</td>
</tr>
<tr>
<td>Descendants</td>
<td>6-138</td>
</tr>
<tr>
<td>Distinct</td>
<td>6-143</td>
</tr>
<tr>
<td>Dimension</td>
<td>6-144</td>
</tr>
<tr>
<td>DrilldownByLayer</td>
<td>6-144</td>
</tr>
<tr>
<td>DrilldownMember</td>
<td>6-145</td>
</tr>
<tr>
<td>DrillupByLayer</td>
<td>6-147</td>
</tr>
<tr>
<td>DrillupMember</td>
<td>6-149</td>
</tr>
<tr>
<td>DTS</td>
<td>6-151</td>
</tr>
<tr>
<td>EnumText</td>
<td>6-152</td>
</tr>
<tr>
<td>EnumValue</td>
<td>6-153</td>
</tr>
<tr>
<td>Except</td>
<td>6-153</td>
</tr>
</tbody>
</table>
Exp 6-154
Extract 6-156
Factorial 6-156
Filter 6-157
FirstChild 6-162
FirstSibling 6-163
FormatDate 6-164
Generate 6-166
Generation 6-167
Generations 6-168
GetFirstDate 6-169
GetFirstDay 6-170
GetLastDate 6-171
GetLastDay 6-172
GetNextDay 6-173
GetRoundDate 6-174
Head 6-175
Hierarchize 6-178
IIF 6-180
InStr 6-183
InString 6-183
Int 6-184
Intersect 6-185
Is 6-187
IsAccType 6-188
IsAncestor 6-189
IsChild 6-190
IsEmpty 6-191
IsGeneration 6-192
IsLeaf 6-193
IsLevel 6-194
IsMatch 6-195
IsSibling 6-196
IsUda 6-198
IsValid 6-200
Item 6-200
JulianDate 6-202
Lag 6-203
LastChild 6-205
LastPeriods 6-206
LastSibling 6-207
|
|------------------|---------|
| Lead            | 6-208   |
| Leaves          | 6-210   |
| Left            | 6-213   |
| Len             | 6-213   |
| Level           | 6-214   |
| Levels          | 6-215   |
| LinkMember      | 6-216   |
| Ln              | 6-217   |
| Log             | 6-218   |
| Log10           | 6-218   |
| Lower           | 6-218   |
| LTrim           | 6-219   |
| Max             | 6-220   |
| Median          | 6-220   |
| MemberRange     | 6-221   |
| Members         | 6-223   |
| Min             | 6-225   |
| Mod             | 6-226   |
| NextMember      | 6-227   |
| NonEmptyCount   | 6-229   |
| NonEmptySubset  | 6-231   |
| NTile           | 6-233   |
| NumToStr        | 6-234   |
| OpeningPeriod   | 6-234   |
| Order           | 6-235   |
| Ordinal         | 6-236   |
| ParallelPeriod  | 6-237   |
| Parent          | 6-238   |
| Percentile      | 6-240   |
| PeriodsToDate   | 6-241   |
| Power           | 6-242   |
| PrevMember      | 6-242   |
| Rank            | 6-244   |
| RealValue       | 6-246   |
| RelMemberRange  | 6-247   |
| Remainder       | 6-248   |
| Right           | 6-249   |
| Round           | 6-250   |
| RTrim           | 6-250   |
| Siblings        | 6-250   |
| Stddev          | 6-252   |
7 Query Logging Configuration

Query Logging Overview 7-1
Query Logging Settings Procedure 7-1
Query Log Settings File Syntax 7-2
Query Logging Sample File 7-6
Query Logging Sample Output 7-6
Preface

Learn how to get started with Oracle Analytics Cloud – Essbase.

Topics

• Audience
• Documentation Accessibility
• Related Resources
• Conventions

Audience

Technical Reference for Oracle Analytics Cloud - Essbase is intended for business users, analysts, modelers, and decision-makers across all lines of business within an organization who use Oracle Analytics Cloud – Essbase.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Related Resources

Use these related resources to expand your understanding of Oracle Analytics Cloud - Essbase.

Topics

• Oracle Public Cloud http://cloud.oracle.com
• Using Oracle Analytics Cloud - Essbase
• Accessibility Guide for Oracle Analytics Cloud - Essbase
• Getting Started with Oracle Analytics Cloud
• Designing and Maintaining Essbase Cubes
Conventions

The following text conventions are used in this document:

<table>
<thead>
<tr>
<th>Convention</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>boldface</strong></td>
<td>Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.</td>
</tr>
<tr>
<td><em>italic</em></td>
<td>Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.</td>
</tr>
<tr>
<td><strong>monospace</strong></td>
<td>Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.</td>
</tr>
</tbody>
</table>
Technical Reference Overview

You can use the commands, functions, and configuration aspects of Essbase to customize it. This reference is intended for advanced users who need detailed information and examples about Essbase elements.

- About the Technical Reference
- About Aggregate Storage Databases

About the Technical Reference

The *Technical Reference for Oracle Essbase* describes commands, functions, and configuration aspects of Essbase. This reference is intended for advanced users who need detailed information and examples about Essbase elements.

- What You Should Know Before You Start
- Sample Applications
- Syntax Conventions

What You Should Know Before You Start

To use this document, you need the following:

- A working knowledge of the operating system your server uses and the ones your clients use.
- An understanding of Essbase concepts and features.
- An understanding of the typical database administration requirements and tasks, including calculation, reporting, security, and maintenance.

Sample Applications

This document provides examples based on the Sample and Demo applications provided with Essbase. The Sample application contains three databases: Basic, Interntl, and Xchgrate. The Demo application contains one database: Basic. If, when you connect to the Essbase Server, any of the following problems occur, contact your administrator.

- You cannot find the Sample or Demo application
- You don't have adequate access to the Sample or Demo application
- You don't see any data in the Sample or Demo databases

Syntax Conventions

This document uses several formatting styles to indicate actions you should take or types of information you need.
<table>
<thead>
<tr>
<th>Syntax</th>
<th>Purpose</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>UPPERCASE</strong></td>
<td>Command or function names in syntax.</td>
<td>BEGINARCHIVE</td>
</tr>
<tr>
<td><em>italic</em></td>
<td>Terms, such as parameters, that you replace with a value</td>
<td>ESSGETSTRING (hCtx, pString); directorypath\filename</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The dimList argument...</td>
</tr>
<tr>
<td><code>&quot; &quot;</code></td>
<td>Double quotation marks enclose text parameters or single parameters that include a space</td>
<td>&quot;appName&quot;</td>
</tr>
<tr>
<td>!</td>
<td>Report Writer: The report output character (bang) signals the start of report processing; this character must be on its own line</td>
<td>SETDEFAULTCALC &quot;CALC ALL&quot;;</td>
</tr>
<tr>
<td>()</td>
<td>Parentheses are used in a couple of ways:</td>
<td>ESSGETSTRING (hCtx, pString);</td>
</tr>
<tr>
<td></td>
<td>• To enclose function parameters</td>
<td>(a + b) * c</td>
</tr>
<tr>
<td></td>
<td>• To show the order of execution of the enclosed operations</td>
<td></td>
</tr>
<tr>
<td>//</td>
<td>Comment markers in report scripts.</td>
<td>// Get results</td>
</tr>
<tr>
<td>/* ... */</td>
<td>Comment markers in calculation scripts.</td>
<td>/<em>Get results</em>/</td>
</tr>
<tr>
<td>;</td>
<td>Statement terminator</td>
<td>EXIT;</td>
</tr>
<tr>
<td>[ ]</td>
<td>Brackets enclose optional parameters in syntax. Used with OR symbol</td>
<td>INDENT [ offset ]</td>
</tr>
<tr>
<td></td>
<td>• if there is more than one optional parameter. Do not type</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• brackets or the OR symbol</td>
<td>.</td>
</tr>
<tr>
<td></td>
<td>[, numeric] Indicates an optional numeric (no quotes) or character (quoted) parameter and the comma which must precede the optional parameter. Do not type the brackets.</td>
<td>[, year]</td>
</tr>
<tr>
<td></td>
<td>[, &quot;text&quot;]</td>
<td>[, &quot;columnName&quot;]</td>
</tr>
<tr>
<td>{}</td>
<td>Braces group statements for processing, enclose alternatives, one of which you must choose Report Writer: Enclose report formatting commands</td>
<td>HELP { ?</td>
</tr>
<tr>
<td></td>
<td></td>
<td>{ SUPFORMATS }</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Syntax: OR. Separates alternatives from which you choose only one. Do not type the OR symbol.</td>
</tr>
<tr>
<td>&lt;</td>
<td>Report Writer: Angle bracket precedes layout and member selection commands.</td>
<td>&lt;PAGE</td>
</tr>
</tbody>
</table>
Table 1-1  (Cont.) Syntax Conventions

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Purpose</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>@</td>
<td>Essbase calculation functions: Precedes many function names</td>
<td>@ABS</td>
</tr>
<tr>
<td>-&gt;</td>
<td>Essbase calculation functions: Cross-dimensional operator (a hyphen followed by a greater-than sign) points to data values of specific member combinations -&gt; (cross-dimensional operator)</td>
<td>Price -&gt; West = AVGRANGE</td>
</tr>
</tbody>
</table>

About Aggregate Storage Databases

This topic explains how the elements discussed in this guide apply to aggregate storage databases.

Consider using the aggregate storage model if the following is true for your database:

- The database is sparse and has many dimensions, and/or the dimensions have many levels of members.
- The database is used primarily for read-only purposes, with few or no data updates.
- The outline contains no formulas except in the dimension tagged as Accounts.
- Calculation of the database is frequent, is based mainly on summation of the data, and does not rely on calculation scripts.

Note the applicability of the following elements for aggregate storage databases:

- **MDX**—Used for querying on block storage and aggregate storage databases. Additionally, MDX numeric-value expressions can be used for developing formulas on aggregate storage outlines. For more information, see *Aggregate Storage and MDX Outline Formulas*.
- **Calculation commands**—Not supported in enterprise analytics databases, because calculation scripts are not relevant to aggregate storage.
- **Calculation functions**—Not supported in enterprise analytics databases. Instead, MDX formulas can be written using MDX numeric-value expressions. Only the Accounts dimension can have formulas in aggregate storage databases.
- **Report Writer commands**—All Report Writer commands (except &lt;SPARSE) are supported for aggregate storage databases.
- **MaxL statements**—Some MaxL grammar is applicable to aggregate storage mode, and some MaxL grammar is not relevant. To learn which statements are supported in aggregate storage application and database operations, see *MaxL Statements (Aggregate Storage)*.
- **Configuration settings**—Some configuration settings are applicable to aggregate storage mode, and some are not. To learn which settings are supported in aggregate storage mode, see *Aggregate Storage and Block Storage Settings Comparison*. 

1-3
Calculation Functions

Using the calculation language with its flexible library of functions, you can analyze complex business scenarios and data relationships.

- Calculation Functions Overview
- Function Syntax
- Function Parameters
- Calculation Operators
- Calculation Function Categories
- Calculation Function List
- Functions Supported in Hybrid Aggregation Mode

Calculation Functions Overview

Essbase provides a suite of functions and calculation operators to facilitate the definition and application of complex member formulas.

The topics for individual functions in this section provide examples that are based on an application and database provided with the Essbase Server software, called Sample Basic. If you do not have access to Sample Basic, contact your administrator.

Generations and Levels

Many Essbase functions identify a member in the database by its position in the database outline. The outline structure represents a hierarchical tree; every dimension represents a subsection of the database tree. Generations and levels provide position references for all database members within the tree. Position references are required because many applications must be able to determine the location of members within the database structure.

The terms "generation" and "level" denote the distance from either the "root" or the "leaves" of the dimension. Thus, you can determine the location of any member within a database tree. You can also specify relationships between groups of related members.

Generations specify the distance of members from the root of their dimension. All members in a database that are the same number of branches from their root have the same generation number. The dimension is generation 1, its children are generation 2, and so on.

Levels measure the number of branches between a member and the lowest member below it, that is, the number of branches between a member and the "leaf" of its hierarchy within the database structure. Level 0 specifies the bottom-most members of a dimension and thus provides ready access to the raw data stored in a database. Leaf members are level 0, then their parents are level 1, and so on up the hierarchy.
You might note that when all sibling members have the same generation number but not necessarily the same level number.

For example, the members in this hierarchy:

```
Dim1
  m11
    m111
    m112
  m12
    m121
    m122
  m13
```

have the following generation and level numbers:

```
Dim1    Gen 1, Level 2
m11    Gen 2, Level 1
  m111  Gen 3, Level 0
  m112  Gen 3, Level 0
m12    Gen 2, Level 1
  m121  Gen 3, Level 0
  m122  Gen 3, Level 0
m13    Gen 2, Level 0
```

**Abbreviations**

Function abbreviations are not supported. Use the full function name to obtain expected behavior.

**Function Syntax**

The individual topics for each function include the required syntax for that function. Function names appear in **bold**; required parameters appear in *italics*; and optional parameters appear in brackets [*] and *italics*. Individual topics also discuss the defaults that are used when optional parameters are not specified. For detailed descriptions of each function, along with examples of usage, please refer to the individual topic.

**Function Parameters**

The following table provides a brief description of some of the common parameters used in various functions.

<table>
<thead>
<tr>
<th>Note:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Member names that are also keywords, such as IF, THEN, ELSE, and RETURN, must be enclosed in quotation marks. Best practice is to always enclose member names in quotation marks.</td>
</tr>
<tr>
<td>Parameter</td>
</tr>
<tr>
<td>--------------</td>
</tr>
<tr>
<td><code>attDimName</code></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><code>attMbrName</code></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><code>dimName</code></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><code>expList</code></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><code>expression</code></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><code>genLevName</code></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Parameter</td>
</tr>
<tr>
<td>-------------</td>
</tr>
<tr>
<td><code>genLevNum</code></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><code>mbrList</code></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><code>mbrName</code></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><code>n</code></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><code>propertyName</code></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
### Table 2-1  (Cont.) Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>propertyValue</code></td>
<td>Optional. Member property value. The value must match the data type of the dimension property specified in <code>propertyName</code>.</td>
</tr>
<tr>
<td></td>
<td>@PROPERTY(Market,Size,Medium)</td>
</tr>
<tr>
<td></td>
<td>@PROPERTYBVAL(&quot;New York&quot;,Color)</td>
</tr>
<tr>
<td><code>rangeList</code></td>
<td>A valid member name, a comma-delimited list of member names, member set functions, and range functions from the same dimension. If <code>rangeList</code> is optional and is not specified, Essbase uses the level 0 members from the dimension tagged as Time. If no dimension is tagged as Time and this parameter is omitted, Essbase reports a syntax error. This definition of <code>rangeList</code> also includes <code>mbrList</code>.</td>
</tr>
<tr>
<td></td>
<td>@ACCUM(Q189:Q491)</td>
</tr>
<tr>
<td></td>
<td>@MAXRANGE(Sales,@CHILDREN(Qtr1))</td>
</tr>
<tr>
<td><code>tag</code></td>
<td>Any valid account tag defined in the current database including First, Last, Average, Expense, and Two-Pass.</td>
</tr>
<tr>
<td></td>
<td>@ISACCTYPE(&quot;EXPENSE&quot;)</td>
</tr>
<tr>
<td><code>XrangeList</code></td>
<td>Similar to <code>rangeList</code>, but supports cross dimensional members. A valid member name, a comma-delimited list of member names, cross dimension members, or a member set function or range function (including <code>@XRANGE</code>) that returns a list of members from the same dimension. If <code>XrangeList</code> is not specified, Essbase uses the level 0 members from the dimension tagged as Time.</td>
</tr>
<tr>
<td></td>
<td>See also Range List Parameters.</td>
</tr>
</tbody>
</table>

### Calculation Operators

Calculation operators (mathematical, conditional and logical, and cross-dimensional) define equations for member formulas and calc scripts.
Mathematical Operators

Mathematical operators perform common arithmetic operations.

Table 2-2  Mathematical Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Adds.</td>
</tr>
<tr>
<td>-</td>
<td>Subtracts.</td>
</tr>
<tr>
<td>*</td>
<td>Multiplies.</td>
</tr>
<tr>
<td>/</td>
<td>Divides.</td>
</tr>
<tr>
<td>%</td>
<td>Evaluates percentage. For example, Member1%Member2 evaluates Member1 as a percentage of Member2.</td>
</tr>
<tr>
<td>( )</td>
<td>Controls the order of calculations and nests equations and formulas.</td>
</tr>
</tbody>
</table>

Conditional and Logical Operators

Conditional operators build logical condition into calculations.

Table 2-3  Conditional and Logical Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF</td>
<td>ELSE</td>
</tr>
<tr>
<td>&gt;</td>
<td>Data value is greater than.</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Data value is greater than or equal to.</td>
</tr>
<tr>
<td>&lt;</td>
<td>Data value is less than.</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Data value is less than or equal to.</td>
</tr>
<tr>
<td>= =</td>
<td>Data value is equal to.</td>
</tr>
<tr>
<td>&lt; &gt; or !=</td>
<td>Data value is not equal to.</td>
</tr>
<tr>
<td>AND</td>
<td>Logical AND linking operator for multiple value tests. Result is TRUE if both conditions are TRUE. Otherwise the result is FALSE.*</td>
</tr>
<tr>
<td>OR</td>
<td>Logical OR linking operator for multiple value tests. Result is TRUE if either condition is TRUE. Otherwise the result is FALSE.*</td>
</tr>
<tr>
<td>NOT</td>
<td>Logical NOT operator. Result is TRUE if condition is FALSE. Result is FALSE if condition is TRUE.*</td>
</tr>
</tbody>
</table>

* The logical constants TRUE and FALSE are interpreted as 1 (TRUE) and 0 (FALSE) where appropriate.

Cross-Dimensional Operators

The cross-dimensional operator ( - > ) points to data values of specific member combinations.
The cross-dimensional operator is created with a hyphen ( - ) and a right angle bracket ( > ), with no space between them.

Operation Results on #MISSING Values and Zero (0) Values

If a data value does not exist for a unique combination of members, Essbase gives the combination a value of #MISSING. A #MISSING value is different from a zero (0) value. Therefore, Essbase treats #MISSING values differently from 0 values.

The following tables shows how Essbase calculates #MISSING values. In this table, X represents any number.

Table 2-4 How Essbase Calculates Missing Values

<table>
<thead>
<tr>
<th>Calculation/Operation</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>X + #MISSING</td>
<td>X</td>
</tr>
<tr>
<td>X - #MISSING</td>
<td>X</td>
</tr>
<tr>
<td>#MISSING - X</td>
<td>-X</td>
</tr>
<tr>
<td>X * #MISSING</td>
<td>#MISSING</td>
</tr>
<tr>
<td>X / #MISSING</td>
<td>#MISSING</td>
</tr>
<tr>
<td>#MISSING / X</td>
<td>#MISSING</td>
</tr>
<tr>
<td>X / 0</td>
<td>#MISSING</td>
</tr>
<tr>
<td>X % #MISSING</td>
<td>#MISSING</td>
</tr>
<tr>
<td>#MISSING % X</td>
<td>#MISSING</td>
</tr>
<tr>
<td>X % 0</td>
<td>#MISSING</td>
</tr>
<tr>
<td>X == #MISSING</td>
<td>False, unless X is #MISSING</td>
</tr>
<tr>
<td>X != #MISSING</td>
<td>True, unless X is #MISSING</td>
</tr>
<tr>
<td>X &lt;&gt; #MISSING</td>
<td>True, unless X is #MISSING</td>
</tr>
<tr>
<td>(X &lt;= #MISSING)</td>
<td>(X &lt;= 0)</td>
</tr>
<tr>
<td>(X &gt;= #MISSING)</td>
<td>(X &gt;= 0) or (X == #MISSING)</td>
</tr>
<tr>
<td>(X &gt; #MISSING)</td>
<td>(X &gt; 0)</td>
</tr>
<tr>
<td>(X &lt; #MISSING)</td>
<td>(X &lt; 0)</td>
</tr>
<tr>
<td>X AND #MISSING:</td>
<td>#MISSING</td>
</tr>
<tr>
<td>1 AND #MISSING (1 represents any nonzero value)</td>
<td>0 #MISSING</td>
</tr>
<tr>
<td>0 AND #MISSING</td>
<td>#MISSING</td>
</tr>
<tr>
<td>#MISSING AND #MISSING</td>
<td></td>
</tr>
<tr>
<td>X OR #MISSING:</td>
<td>1 #MISSING</td>
</tr>
<tr>
<td>1 OR #MISSING (1 represents any nonzero value)</td>
<td>#MISSING</td>
</tr>
<tr>
<td>0 OR #MISSING</td>
<td>#MISSING</td>
</tr>
<tr>
<td>#MISSING OR #MISSING</td>
<td></td>
</tr>
<tr>
<td>IF (#MISSING)</td>
<td>IF (0)</td>
</tr>
</tbody>
</table>
Table 2-4  (Cont.) How Essbase Calculates Missing Values

<table>
<thead>
<tr>
<th>Calculation/Operation</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>f (#MISSING)</td>
<td>#MISSING for any Essbase function of one variable</td>
</tr>
<tr>
<td>f (X)</td>
<td>#MISSING for any X not in the domain of f, and any Essbase function of more than one variable (except where specifically noted)</td>
</tr>
</tbody>
</table>

Calculation Function Categories

This section lists all of the Essbase calculation functions, grouped by function type.

- Conditional and Logical Operators
- Boolean Functions
- Relationship Functions
- Calculation Operators
- Mathematical Functions
- Member Set Functions
- Range and Financial Functions
- Allocation Functions
- Forecasting Functions
- Statistical Functions
- Date & Time Function
- Miscellaneous Functions

Boolean Functions

A Boolean function returns TRUE or FALSE (1 or 0, respectively). Boolean functions are generally used in conjunction with the IF command to provide a conditional test. Because they generate a numeric value, however, Boolean functions can also be used as part of a member formula.

Boolean functions are useful because they can determine which formula to apply based on characteristics of the current member combination. For example, you may want to restrict a calculation to those members in a dimension that contain input data. In this case, you preface the calculation with an IF test that is based on @ISLEV (dimName, 0).

If one of the function parameters is a cross-dimensional member; for example, @@ISMBR (Sales->Budget), all parts of the cross-dimensional member must match all parts of the current cell to return a value of TRUE.

In the following quick-reference table, “the current member” means the member that is currently being calculated by the function. Words in italics, such as member, loosely indicate information you supply to the function. For details, see the individual function topics.
Table 2-5  Boolean Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Condition Tested</th>
</tr>
</thead>
<tbody>
<tr>
<td>@ISACCTYPE</td>
<td>Whether the current member has a particular accounts tag.</td>
</tr>
<tr>
<td>@ISANCEST</td>
<td>Whether the current member is an ancestor of member.</td>
</tr>
<tr>
<td>@ISCHILD</td>
<td>Whether the current member is a child of member.</td>
</tr>
<tr>
<td>@ISDESC</td>
<td>Whether the current member is a descendant of member.</td>
</tr>
<tr>
<td>@ISGEN</td>
<td>Whether the current member of dimension is in generation.</td>
</tr>
<tr>
<td>@ISIANCEST</td>
<td>Whether the current member is the same member or an ancestor of member.</td>
</tr>
<tr>
<td>@ISICHILD</td>
<td>Whether the current member is the same member or a child of member.</td>
</tr>
<tr>
<td>@ISIDESC</td>
<td>Whether the current member is the same member or a descendant of member.</td>
</tr>
<tr>
<td>@ISIPARENT</td>
<td>Whether the current member is the same member or the parent of member.</td>
</tr>
<tr>
<td>@ISISIBLING</td>
<td>Whether the current member is the same member or a sibling of member.</td>
</tr>
<tr>
<td>@ISLEV</td>
<td>Whether the current member of dimension is in level.</td>
</tr>
<tr>
<td>@ISMBR</td>
<td>Whether the current member is member, or is found in member list, or is found in range returned by another function.</td>
</tr>
<tr>
<td>@ISMBRUDA</td>
<td>Whether the specified user-defined attribute string exists for the specified member.</td>
</tr>
<tr>
<td>@ISPARENT</td>
<td>Whether the current member is the parent of member.</td>
</tr>
<tr>
<td>@ISRANGENONEMPTY</td>
<td>Whether data values exist for a specified range.</td>
</tr>
<tr>
<td>@ISSAMEGEN</td>
<td>Whether the current member is in the same generation as member.</td>
</tr>
<tr>
<td>@ISSAMELEV</td>
<td>Whether the current member is in the same level as member.</td>
</tr>
<tr>
<td>@ISSIBLING</td>
<td>Whether the current member is a sibling of member.</td>
</tr>
<tr>
<td>@ISUDA</td>
<td>Whether the current member of dimension has a particular user-defined attribute string.</td>
</tr>
</tbody>
</table>

Relationship Functions

Relationship functions look up specific values within the database based on current cell location and a series of parameters. You can use these functions to refer to another value in a data series. Relationship functions have an implicit current member argument; that is, these functions are dependent on the current member's position.

In the following quick-reference table, words in italics loosely represent information you supply to the function. For details, see the individual function topics.
Table 2-6  Relationship Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>@ANCESTVAL</td>
<td>Ancestor values of a specified one-dimensional member combination.</td>
</tr>
<tr>
<td>@ATTRIBUTEBVAL</td>
<td>Associated attribute value from a Boolean attribute dimension.</td>
</tr>
<tr>
<td>@ATTRIBUTESVAL</td>
<td>Associated attribute value from a text attribute dimension.</td>
</tr>
<tr>
<td>@ATTRIBUTEVAL</td>
<td>Associated attribute value from a numeric or date attribute dimension.</td>
</tr>
<tr>
<td>@CURGEN</td>
<td>Generation number of the current member in dimension.</td>
</tr>
<tr>
<td>@CURLEV</td>
<td>Level number of the current member in dimension.</td>
</tr>
<tr>
<td>@GEN</td>
<td>Generation number of member.</td>
</tr>
<tr>
<td>@LEV</td>
<td>Level number of member.</td>
</tr>
<tr>
<td>@MDANCESTVAL</td>
<td>Ancestor values for any number of multidimensional member combinations.</td>
</tr>
<tr>
<td>@MDPARENTVAL</td>
<td>Parent values for any number of multidimensional member combinations.</td>
</tr>
<tr>
<td>@PARENTVAL</td>
<td>Parent values for member in dimension.</td>
</tr>
<tr>
<td>@SANCESTVAL</td>
<td>Ancestor values for shared members at a certain depth under a root member.</td>
</tr>
<tr>
<td>@SPARENTVAL</td>
<td>Parent values for shared members under a root member.</td>
</tr>
<tr>
<td>@WEIGHTEDSUMX</td>
<td>Aggregates all members in a member list, depending on the unit weight of each member.</td>
</tr>
<tr>
<td>@XREF</td>
<td>Values from a different database than the one being calculated.</td>
</tr>
<tr>
<td>@XWRITE</td>
<td>Writes values to a different database than the one being calculated.</td>
</tr>
</tbody>
</table>

Mathematical Functions

These functions perform specific mathematical calculations. Mathematical functions define and return values that are based on selected member expressions. These functions cover many basic statistical functions and return numeric results that are based on supplied member values. Advanced statistical functions are included in the statistical functions category.

In the following quick-reference table, words in italics loosely represent information you supply to the function. For details, see the individual function topics.

Table 2-7  Mathematical Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>@ABS</td>
<td>Absolute value of expression.</td>
</tr>
<tr>
<td>@AVG</td>
<td>Average of all values in expList.</td>
</tr>
</tbody>
</table>
### Table 2-7  (Cont.) Mathematical Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>@EXP</td>
<td>e (base of natural logarithms) raised to the power of expression.</td>
</tr>
<tr>
<td>@FACTORIAL</td>
<td>Factorial of expression.</td>
</tr>
<tr>
<td>@INT</td>
<td>Next lowest integer value of expression.</td>
</tr>
<tr>
<td>@LN</td>
<td>e (base of natural logarithms) of expression.</td>
</tr>
<tr>
<td>@LOG</td>
<td>Any base logarithm of expression.</td>
</tr>
<tr>
<td>@LOG10</td>
<td>Base-10 logarithm of expression.</td>
</tr>
<tr>
<td>@MAX</td>
<td>Maximum value found in cells of an expression list.</td>
</tr>
<tr>
<td>@MAXS</td>
<td>Maximum value found in cells of an expression list, optionally skipping empty values.</td>
</tr>
<tr>
<td>@MIN</td>
<td>Minimum value found in cells of expression list.</td>
</tr>
<tr>
<td>@MINS</td>
<td>Minimum value found in cells of an expression list, optionally skipping empty values.</td>
</tr>
<tr>
<td>@MOD</td>
<td>Modulus of a division operation between two members.</td>
</tr>
<tr>
<td>@POWER</td>
<td>Expression raised to power.</td>
</tr>
<tr>
<td>@REMAINDER</td>
<td>Remainder value of expression.</td>
</tr>
<tr>
<td>@ROUND</td>
<td>Expression rounded to numDigits.</td>
</tr>
<tr>
<td>@SUM</td>
<td>Sum of values found in cells of an expression list.</td>
</tr>
<tr>
<td>@TRUNCATE</td>
<td>Expression with fractional part removed, returning an integer.</td>
</tr>
<tr>
<td>@VAR</td>
<td>Variance between two members.</td>
</tr>
<tr>
<td>@VARPER</td>
<td>Percent variance between two members.</td>
</tr>
</tbody>
</table>

### Member Set Functions

Member set functions return a list of members. This list is based on the member specified and the function used. You can use operators to specify [Generation and Level Range Operators for Member Set Functions](#) with member set functions.

When a member set function is called as part of a formula, the list of members is generated before the calculation begins. The list never varies because it is based on the specified member and is independent of the current member.

If a member set function (for example, @CHILDREN or @SIBLINGS) is used to specify the list of members to calculate in a calculation script, Essbase bypasses the calculation of any Dynamic Calc or Dynamic Calc and Store members in the resulting list.

Only the @ATTRIBUTE and @WITHATTR functions can use attribute members or members of the Attribute Calculations dimension as parameters in member set functions.

You can use cross-dimensional expressions such as ("1998":"2001" -> @Levmbrs (Year, 0)). The cross-dimensional operator is associative (x -> y) -> z=x -> (y -> z), but not commutative because x -> y = y -> x is a set, but the order of elements is different.
## Table 2-8  Member Set Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>@ALLANCESTORS</td>
<td>All ancestors of member, including ancestors of shared member.</td>
</tr>
<tr>
<td>@ANCEST</td>
<td>Ancestor at distance from the current member or an explicitly specified member.</td>
</tr>
<tr>
<td>@ANCESTORS</td>
<td>All ancestors of member, or those ancestors up to a specified distance.</td>
</tr>
<tr>
<td>@ATTRIBUTE</td>
<td>All base members associated with attribute member name.</td>
</tr>
<tr>
<td>@BETWEEN</td>
<td>All members whose name string value fall between, and are inclusive of, two specified string tokens.</td>
</tr>
<tr>
<td>@CHILDREN</td>
<td>Children of member.</td>
</tr>
<tr>
<td>@CURRMBR</td>
<td>Member currently being calculated in the specified dimension.</td>
</tr>
<tr>
<td>@DESCENDANTS</td>
<td>All descendants of member, or those descendants down to a specified distance.</td>
</tr>
<tr>
<td>@EQUAL</td>
<td>Member names that match the specified token name.</td>
</tr>
<tr>
<td>@EXPAND</td>
<td>Expands a member search by calling a member set function for each member in a member list.</td>
</tr>
<tr>
<td>@GENMBRS</td>
<td>Members of dimension that are at generation.</td>
</tr>
<tr>
<td>@IALLANCESTORS</td>
<td>Member and ancestors of member, including ancestors of shared member.</td>
</tr>
<tr>
<td>@IANCESTORS</td>
<td>Member, and either all member ancestors or those ancestors up to a specified distance.</td>
</tr>
<tr>
<td>@ICLCHILDREN</td>
<td>Member and its children.</td>
</tr>
<tr>
<td>@IDESCEDEANTANTS</td>
<td>Member, and either all member descendants or those descendants down to a specified distance.</td>
</tr>
<tr>
<td>@ILANCESTORS</td>
<td>Members of the specified list of members, and either all ancestors of the specified list of members or those ancestors up to a specified distance.</td>
</tr>
<tr>
<td>@ILDESCENDANTS</td>
<td>Members of the specified list of members, and either all descendants of the specified list of members or those descendants down to a specified distance.</td>
</tr>
<tr>
<td>@ILSIBLINGS</td>
<td>Member and its left siblings.</td>
</tr>
<tr>
<td>@INTERSECT</td>
<td>Members that are at the intersection of two specified lists of members.</td>
</tr>
<tr>
<td>@IRSIBLINGS</td>
<td>Member and its right siblings.</td>
</tr>
<tr>
<td>@IRDESCENDANTS</td>
<td>Member and all its descendants, or those descendants down to a specified distance, including descendants of shared member.</td>
</tr>
<tr>
<td>@ISIBLINGS</td>
<td>Member and its siblings.</td>
</tr>
<tr>
<td>@LANCESTORS</td>
<td>All ancestors of the specified list of members, or those ancestors up to a specified distance.</td>
</tr>
</tbody>
</table>
Table 2-8  (Cont.) Member Set Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>@LDESCENDANTS</td>
<td>All descendants of the specified list of members, or those descendants down to a specified distance.</td>
</tr>
<tr>
<td>@LEVMBRS</td>
<td>Members of dimension that are at level.</td>
</tr>
<tr>
<td>@LIST</td>
<td>A single list compiled from arguments, and can be used for functions requiring an expression list, a member list, or a range list.</td>
</tr>
<tr>
<td>@LSIBLINGS</td>
<td>Left siblings of member.</td>
</tr>
<tr>
<td>@MATCH</td>
<td>Members that match a pattern search performed over a generation, a level, or a member and its descendants.</td>
</tr>
<tr>
<td>@MBRCOMPARE</td>
<td>Member names that match the comparison criteria.</td>
</tr>
<tr>
<td>@MBRPARENT</td>
<td>Parent of the specified member.</td>
</tr>
<tr>
<td>@MEMBER</td>
<td>Member with name string.</td>
</tr>
<tr>
<td>@MEMBERAT</td>
<td>Member at the specified location in a list.</td>
</tr>
<tr>
<td>@MERGE</td>
<td>Merged list from two lists.</td>
</tr>
<tr>
<td>@NEXTSIBLING</td>
<td>Next, or right-most, sibling of member.</td>
</tr>
<tr>
<td>@NOTEQUAL</td>
<td>Member names that do not match the specified token name.</td>
</tr>
<tr>
<td>@PARENT</td>
<td>Parent of the current member being calculated in dimension, optionally crossed with another member.</td>
</tr>
<tr>
<td>@PREVSIBLING</td>
<td>Previous, or left-most, sibling of member.</td>
</tr>
<tr>
<td>@RANGE</td>
<td>Member list that crosses a member from one dimension with a range from another dimension.</td>
</tr>
<tr>
<td>@RDESCENDANTS</td>
<td>All descendants of member, or those down to a specified distance, including descendants of shared member.</td>
</tr>
<tr>
<td>@RELATIVE</td>
<td>All members that are at distance from member.</td>
</tr>
<tr>
<td>@REMOVE</td>
<td>List1, with anything that is also in list2 removed.</td>
</tr>
<tr>
<td>@RSIBLINGS</td>
<td>Right siblings of member.</td>
</tr>
<tr>
<td>@SHIFTSIBLING</td>
<td>Sibling at specified distance from member.</td>
</tr>
<tr>
<td>@SIBLINGS</td>
<td>Siblings of member.</td>
</tr>
<tr>
<td>@UDA</td>
<td>Members of dimension that have UDA.</td>
</tr>
<tr>
<td>@WITHATTR</td>
<td>Base members from dimension that are associated with an attribute meeting a condition.</td>
</tr>
<tr>
<td>@XRANGE</td>
<td>Range of members between (and inclusive of) two members at the same level.</td>
</tr>
</tbody>
</table>

Generation and Level Range Operators for Member Set Functions

The operators : and :: can be used with member set functions, which return a list of members. The : operator returns level-based ranges and the :: operator returns
generation-based ranges. For example, Jan:Dec and Jan::Dec both return all members between and inclusive of Jan and Dec.

The difference is that Jan:Dec returns all members at the same level and Jan::Dec returns all members at the same generation.

For example, if we have the outline:

Q1 - Jan
   Feb
   Mar
Q2 - Apr
   May
   Jun
Q3
Q4 - Oct
   Nov
   Dec

The function `@MOVAVG(Sales, 3, Jan:Dec)` computes `@MOVAVG(Sales, 3, Jan, Feb, Mar, Apr, May, Jun, Q3, Oct, Nov, Dec).

The function `@MOVAVG(Sales, 3, Jan::Dec)` computes `@MOVAVG(Sales, 3, Jan, Feb, Mar, Apr, May, Jun, Oct, Nov, Dec).

### Range and Financial Functions

Range functions take a range of members as an argument. Rather than return a single value, these functions calculate a series of values internally based on the range specified.

Financial functions execute specialized financial calculations.

#### Table 2-9  Range and Financial Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>@ACCUM</td>
<td>The sum of values of a specified member across a range</td>
</tr>
<tr>
<td>@AVGRANGE</td>
<td>The average of values of a specified member across a range</td>
</tr>
<tr>
<td>@COMPOUND</td>
<td>The compound interest of values of a specified member across a range, calculated at a specified rate</td>
</tr>
<tr>
<td>@COMPOUNDGROWTH</td>
<td>A series of values that represent the compound growth of the specified member across a range of members, calculated at a specified rate</td>
</tr>
<tr>
<td>@CURRMBRRANGE</td>
<td>A range of members that is based on the relative position of the member combination Essbase is currently calculating</td>
</tr>
<tr>
<td>@DECLINE</td>
<td>Depreciation of a member over a specified period, calculated using the declining balance method</td>
</tr>
</tbody>
</table>
Table 2-9  (Cont.) Range and Financial Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>@DISCOUNT</td>
<td>Discounted values of a specified member, calculated at a specified rate, across a range of values from the time dimension</td>
</tr>
<tr>
<td>@GROWTH</td>
<td>A series of values that represents the linear growth of the specified value</td>
</tr>
<tr>
<td>@INTEREST</td>
<td>A series of values that represent the linear growth of a specified member, calculated at a specified rate, across a range of members from the time dimension</td>
</tr>
<tr>
<td>@IRR</td>
<td>The Internal Rate of Return on a cash flow that is calculated across the time dimension or a specified range of members and must contain at least one investment (negative) and one income (positive). Includes an initial guess of 0.07 (the initial guess cannot be configured).</td>
</tr>
<tr>
<td>@IRREX</td>
<td>The Internal Rate of Return on a cash flow that is calculated across the time dimension or a specified range of members and must contain at least one investment (negative) and one income (positive). Includes functionality to configure the initial guess and the number of iterations the algorithm can make.</td>
</tr>
<tr>
<td>@MAXRANGE</td>
<td>The maximum value of a member across a range of members</td>
</tr>
<tr>
<td>@MAXSRANGE</td>
<td>The maximum value of a member across a range of members, with the ability to skip zero and #MISSING values</td>
</tr>
<tr>
<td>@MDSHIFT</td>
<td>The next or nth member in a range of members, retaining all other members identical to the current member across multiple dimensions</td>
</tr>
<tr>
<td>@MINRANGE</td>
<td>The minimum value of a member across a range of members</td>
</tr>
<tr>
<td>@MINSRANGE</td>
<td>The minimum value of a member across a range of members, with the ability to skip zero and #MISSING values</td>
</tr>
<tr>
<td>@NEXT</td>
<td>The next or nth member in a range of members</td>
</tr>
<tr>
<td>@NEXTS</td>
<td>The next or nth member in a range of members, with the option to skip #MISSING, zero, or both values</td>
</tr>
<tr>
<td>@NPV</td>
<td>The Net Present Value of an investment based on a series of payments and income values</td>
</tr>
<tr>
<td>@PTD</td>
<td>The period-to-date values of members in the time dimension</td>
</tr>
<tr>
<td>@PRIOR</td>
<td>A list of the previous or nth previous members in a range of members</td>
</tr>
<tr>
<td>@PRIORS</td>
<td>A list of the previous or nth previous members in a range of members, with the option to skip #MISSING, zero, or both values</td>
</tr>
</tbody>
</table>
## Table 2-9  (Cont.) Range and Financial Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>@RANGE</td>
<td>A member list that crosses the specified member from one dimension with the specified member range from another dimension</td>
</tr>
<tr>
<td>@RANGEFIRSTVAL</td>
<td>The first value in a range (with options for how to handle zero and #MISSING).</td>
</tr>
<tr>
<td>@RANGELASTVAL</td>
<td>The last value in a range (with options for how to handle zero and #MISSING).</td>
</tr>
<tr>
<td>@SHIFT</td>
<td>A list of the next or nth members in a range of members, retaining all other members identical to the current member and in the specified dimension</td>
</tr>
<tr>
<td>@SHIFTPLUS</td>
<td></td>
</tr>
<tr>
<td>@SHIFTMINUS</td>
<td></td>
</tr>
<tr>
<td>@SLN</td>
<td>Depreciation amounts, across a range period, that an asset in the current period may be depreciated, calculated using the straight-line depreciation method</td>
</tr>
<tr>
<td>@SUMRANGE</td>
<td>A list of summarized values of all specified members across a range of members</td>
</tr>
<tr>
<td>@SYD</td>
<td>Depreciation amounts, across a range of periods, of an asset in the current period, calculated using the sum of the year's digits depreciation method</td>
</tr>
<tr>
<td>@XRANGE</td>
<td>A list of a range of members between specified members at the same level</td>
</tr>
</tbody>
</table>

### Range List Parameters

Some range and forecasting functions recognize the optional parameter `rangeList` or `XrangeList` as the last parameter. `rangeList` is a range of members restricted to one dimension; `XrangeList` is a range of members that can be from one or multiple dimensions. `XrangeList` helps you incorporate time continuum navigation for the calculation functions you use.

If `rangeList` or `XrangeList` are not given, the level 0 (leaf) members from the dimension tagged as Time become the default range. If no dimension is tagged as Time and the last parameter is not given, Essbase reports a syntax error.

### Examples of `rangeList`

The following examples are based on Sample Basic.

- `@CHILDREN(West)` is a `rangeList` that returns the following list:

  - California
  - Oregon
  - Washington
  - Utah
  - Nevada
@CHILDREN(Product) is a rangeList that returns the following list:

Colas
Root Beer
Cream Soda
Fruit Soda
Diet Drinks

As you can see from the above examples, rangeList is a list of members from a single dimension only.

Examples of XrangeList

The following examples are based on Sample Basic.

The following example uses simple members to return the range between Jan and Mar:

@XRANGE(Jan:Mar)

and returns the following members:

Jan
Feb
Mar

The following example uses cross dimensional members to return the range between Actual, Jan and Budget, Mar:

@XRANGE (Actual->Jan, Budget->Mar)

and returns the following members:

Actual, Jan
Actual, Feb
Actual, Mar
Actual, Apr
Actual, May
Actual, Jun
Actual, Jul
Actual, Aug
Actual, Sep
Actual, Oct
Actual, Nov
Actual, Dec
Budget, Jan
Budget, Feb
Budget, Mar

The following example is not based on the Sample Basic database. It is based on database that contains a dimension called Year that contains members for each year,
from 2001 to 2003. The following formula computes the average sales for all months between Mar of 2000 and Jan of 2001:

\[
SalesAvg = \text{MOVAVG}(Sales, 3, \text{XRANGE("2001"->Mar, "2003"->Jan))};
\]

and returns the following members:

<table>
<thead>
<tr>
<th></th>
<th>Colas</th>
<th>New York</th>
<th>Actual</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Sales</td>
<td>SalesAvg</td>
<td></td>
</tr>
<tr>
<td></td>
<td>======</td>
<td>=========</td>
<td>========</td>
</tr>
<tr>
<td>2000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mar</td>
<td>678</td>
<td>678</td>
<td></td>
</tr>
<tr>
<td>Apr</td>
<td>645</td>
<td>645</td>
<td></td>
</tr>
<tr>
<td>May</td>
<td>675</td>
<td>666</td>
<td></td>
</tr>
<tr>
<td>Jun</td>
<td>712</td>
<td>677.3</td>
<td></td>
</tr>
<tr>
<td>Jul</td>
<td>756</td>
<td>714.3</td>
<td></td>
</tr>
<tr>
<td>Aug</td>
<td>890</td>
<td>786</td>
<td></td>
</tr>
<tr>
<td>Sep</td>
<td>924</td>
<td>856.7</td>
<td></td>
</tr>
<tr>
<td>Oct</td>
<td>914</td>
<td>909.3</td>
<td></td>
</tr>
<tr>
<td>Nov</td>
<td>912</td>
<td>916.7</td>
<td></td>
</tr>
<tr>
<td>Dec</td>
<td>723</td>
<td>849.7</td>
<td></td>
</tr>
<tr>
<td>2001</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jan</td>
<td>647</td>
<td>760.7</td>
<td></td>
</tr>
</tbody>
</table>

As you can see from the above examples, XrangeList is a range of members from one or more dimensions, and can help you incorporate time continuum navigation.

**More Examples of rangeList and XrangeList**

The following table provides more examples of valid values for rangeList or XrangeList.

| Table 2-10  Valid Values for rangeList and XrangeList |
|------------|------------------------------------------------------|
| **Example**| **Description**                                      |
| Mar99      | A single member                                       |
| Mar99, Apr99, May99 | A comma-delimited list of members.                |
| Jan99:Dec99| A level range. A level range includes all members on the same level between and including the members defining the range. |
| Q1_99::Q4_2000 | A generation range. A generation range includes the members defining the range and all members that are within the range and of the same generation. |
| Q1_99::Q4_2000, FY98, FY99, FY2000 @SIBLINGS(Dept01), Dept65:Dept73, Total_Dept | A generation range and a comma-delimited list A member set function and one or more range lists |

The following table provides examples of valid values for XrangeList.
Table 2-11  Valid Values for XrangeList

<table>
<thead>
<tr>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jan-&gt;Actual-&gt;Sales, Dec-&gt;Actual-&gt;Sales</td>
<td>A comma-delimited list of members from one or more dimensions.</td>
</tr>
<tr>
<td>Actual-&gt;Jan, @XRANGE(Actual-&gt;December, Budget-&gt;Mar);</td>
<td>A comma-delimited list and a range.</td>
</tr>
<tr>
<td>@XRANGE(Jan-&gt;Actual, Dec-&gt;Budget);</td>
<td>A @XRANGE function.</td>
</tr>
<tr>
<td>@CHILDREN(&quot;Colas&quot;),@CHILDREN(&quot;West&quot;)</td>
<td>A member set function as part of a range list.</td>
</tr>
</tbody>
</table>

Financial functions never return a value; rather, they internally calculate a series of values based on the range specified and write the results to a range of cells. Thus, you cannot apply any operator directly to the function.

Allocation Functions

These functions allocate values that are input at the parent level. The values are allocated across child members in one or more dimensions, based on specified criteria. These functions consolidate the common tasks that are required to perform allocations in Essbase.

Table 2-12  Allocation Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Allocation Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>@ALLOCATE</td>
<td>Allocates values to lower-level members in one level.</td>
</tr>
<tr>
<td>@MDALLOCATE</td>
<td>Allocates values to lower-level members in multiple dimensions.</td>
</tr>
</tbody>
</table>

Forecasting Functions

Forecasting functions manipulate data for the purpose of smoothing, interpolating, or calculating future values. Forecasting functions are often used in planning, analysis, and modeling applications. Some forecasting functions recognize the optional Range List Parameters rangeList or XrangeList).

Table 2-13  Forecasting Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Data Manipulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>@MOVAVG</td>
<td>Applies a moving average to a data set, replacing each term in the list with a trailing average. This function modifies the data set for smoothing purposes.</td>
</tr>
<tr>
<td>@MOVMAX</td>
<td>Applies a moving maximum to a data set, replacing each term in the list with a trailing maximum. This function modifies the data set for smoothing purposes.</td>
</tr>
</tbody>
</table>
Table 2-13  (Cont.) Forecasting Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Data Manipulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>@MOVMED</td>
<td>Applies a moving median to a data set, replacing each term in the list with a trailing median. This function modifies the data set for smoothing purposes.</td>
</tr>
<tr>
<td>@MOVMIN</td>
<td>Applies a moving minimum to a data set, replacing each term in the list with a trailing minimum. This function modifies the data set for smoothing purposes.</td>
</tr>
<tr>
<td>@MOVSUM</td>
<td>Applies a moving sum to a data set. This function modifies the data set for smoothing purposes.</td>
</tr>
<tr>
<td>@MOVSUMX</td>
<td>Applies a moving sum to a data set, enabling specification of values for trailing members. This function modifies the data set for smoothing purposes.</td>
</tr>
<tr>
<td>@SPLINE</td>
<td>Applies a smoothing spline to a set of data points. A spline is a mathematical curve that is used to smooth or interpolate data.</td>
</tr>
<tr>
<td>@TREND</td>
<td>Calculates future values, basing the calculation on curve-fitting to historical values</td>
</tr>
</tbody>
</table>

Statistical Functions

Statistical functions calculate advanced statistical values, such as correlation or variance. These functions are often used in sales and marketing applications.

Table 2-14  Statistical Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>@CORRELATION</td>
<td>The correlation coefficient between two parallel data sets</td>
</tr>
<tr>
<td>@COUNT</td>
<td>The number of data values in the specified data set</td>
</tr>
<tr>
<td>@MEDIAN</td>
<td>The median (middle value) of the specified data set</td>
</tr>
<tr>
<td>@MODE</td>
<td>The mode (the most frequently occurring value) in the specified data set</td>
</tr>
<tr>
<td>@RANK</td>
<td>The rank (position in the sorted data set) of the specified members or the specified value among the values in the specified data set.</td>
</tr>
<tr>
<td>@STDEV</td>
<td>The standard deviation of the specified data set</td>
</tr>
<tr>
<td>@STDEVP</td>
<td>The standard deviation of the specified data set, calculated over the entire population</td>
</tr>
<tr>
<td>@STDEVRANGE</td>
<td>The standard deviation of all values of the specified member across the specified data set. The specified mbrName is crossed with a range list to obtain the sample across which the standard deviation is calculated.</td>
</tr>
</tbody>
</table>
Table 2-14  (Cont.) Statistical Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>@VARIANCE</td>
<td>The statistical variance of the specified data set (expList), based upon a sample of a population</td>
</tr>
<tr>
<td>@VARIANCEP</td>
<td>The statistical variance of the specified data set (expList), based upon the entire population</td>
</tr>
</tbody>
</table>

Date & Time Function

The date function, @TODATE, converts date strings to numbers that can be used in calculation formulas.

Miscellaneous Functions

- **@CALCMODE**—This function enables you to specify whether a formula is calculated in cell mode or block mode and whether a formula is calculated bottom-up or top-down
- **@CONCATENATE, @SUBSTRING, and @NAME**—These functions enable manipulation of character strings.
- **@RETURN**—This function enables termination of a calculation, with a custom error message.
- **@ALLOCATE and @CREATEBLOCK**—These functions populate cells with values or #MISSING.

Calculation Function List

Consult the Contents pane for a categorical list of calculation functions.

Table 2-15  Calculation Function List

<table>
<thead>
<tr>
<th>Alphabetical List of Calculation Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>@ABS @ISATTRIBUTE @NEXT @ACCUM @ISCHILD @NEXTS</td>
</tr>
<tr>
<td>@ALLOCA @ISDESC @NEXTSIBLING @ANCESTORS @ISGEN @POWER</td>
</tr>
<tr>
<td>@ANCEST @ISANCEST @PREVSIBLING @ALIAS @ISCHILD @PRIOR</td>
</tr>
<tr>
<td>@ANCESTORS @ISDESC @PRIORS @ANCESTVAL @ISIDESC @PTD</td>
</tr>
<tr>
<td>@ATTRIBUTE @ISIANCEST @RANGE @ATTRIBUTESVAL @ISIPARENT @ISMBR</td>
</tr>
<tr>
<td>@ATTRIBUTESVAL @ISISIBLING @ISMISSING</td>
</tr>
</tbody>
</table>
Table 2-15  (Cont.) Calculation Function List

<table>
<thead>
<tr>
<th>Alphabetical List of Calculation Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>@BETWEEN</td>
</tr>
<tr>
<td>@CALCMODE</td>
</tr>
<tr>
<td>@CHILDREN</td>
</tr>
<tr>
<td>@COMPOUND</td>
</tr>
<tr>
<td>@COMPOUNDGROWTH</td>
</tr>
<tr>
<td>@CONCATENATE</td>
</tr>
<tr>
<td>@CORRELATION</td>
</tr>
<tr>
<td>@COUNT</td>
</tr>
<tr>
<td>@CREATEBLOCK</td>
</tr>
<tr>
<td>@CURGEN</td>
</tr>
<tr>
<td>@CURLEV</td>
</tr>
<tr>
<td>@CURRMBR</td>
</tr>
<tr>
<td>@CURRMBRRANGE</td>
</tr>
<tr>
<td>@DATEDIFF</td>
</tr>
<tr>
<td>@DATEPART</td>
</tr>
<tr>
<td>@DATEROLL</td>
</tr>
<tr>
<td>@DECLINE</td>
</tr>
<tr>
<td>@DESCENDANTS</td>
</tr>
<tr>
<td>@DISCOUNT</td>
</tr>
<tr>
<td>@ENUMVALUE</td>
</tr>
<tr>
<td>@EQUAL</td>
</tr>
<tr>
<td>@EXP</td>
</tr>
<tr>
<td>@EXPAND</td>
</tr>
<tr>
<td>@FACTORIAL</td>
</tr>
<tr>
<td>@FORMATDATE</td>
</tr>
<tr>
<td>@GEN</td>
</tr>
<tr>
<td>@GENMBRS</td>
</tr>
<tr>
<td>@GROWTH</td>
</tr>
<tr>
<td>@IALLANCESTORS</td>
</tr>
<tr>
<td>@ANCESTORS</td>
</tr>
<tr>
<td>@ICLIENTREN</td>
</tr>
<tr>
<td>@IDESCENTANTS</td>
</tr>
<tr>
<td>@ILANCESTORS</td>
</tr>
<tr>
<td>@ILDESCENDANTS</td>
</tr>
<tr>
<td>@ILSIBLINGS</td>
</tr>
<tr>
<td>@INT</td>
</tr>
<tr>
<td>@INTEREST</td>
</tr>
<tr>
<td>@INTERSECT</td>
</tr>
<tr>
<td>@IRDESCENDANTS</td>
</tr>
<tr>
<td>@IRR</td>
</tr>
<tr>
<td>@IRREX</td>
</tr>
<tr>
<td>@IRSIBLINGS</td>
</tr>
<tr>
<td>@ISACCTYPE</td>
</tr>
</tbody>
</table>

Chapter 2
Calculation Function List
Table 2-15  (Cont.) Calculation Function List

<table>
<thead>
<tr>
<th>Alphabetical List of Calculation Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>@ISANCEST</td>
</tr>
</tbody>
</table>

@ABS

Returns the absolute value of expression. The absolute value of a number is that number less its sign. A negative number becomes positive, while a positive number remains positive.

Syntax

@ABS (expression)

Parameters

expression
Member name or mathematical expression that generates a numeric value.

Example

The following example is based on the Demo Basic database. In this example, Variance needs to be presented as a positive number. The @ABS function is used because otherwise some combinations of Actual - Budget would return negative values.

Variance=@ABS(Actual-Budget);

This example produces the following report:

<table>
<thead>
<tr>
<th>Sales</th>
<th>VCR</th>
<th>San_Francisco</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Jan</td>
<td>Feb</td>
</tr>
<tr>
<td>===</td>
<td>===</td>
<td>===</td>
</tr>
<tr>
<td>Actual</td>
<td>1,323</td>
<td>1,290</td>
</tr>
<tr>
<td>Budget</td>
<td>1,200</td>
<td>1,100</td>
</tr>
<tr>
<td>Variance</td>
<td>123</td>
<td>190</td>
</tr>
</tbody>
</table>

@ACCUM

Accumulates the values of mbrName within rangeList, up to the current member in the dimension of which rangeList is a part.

Syntax

@ACCUM (mbrName [, rangeList])
Parameters

mbrName
Any valid single member name (or a function that returns a single member) whose value is to be accumulated.

rangeList
Optional comma-delimited list of members, member set functions, or range functions, across which the accumulation occurs. If rangeList is not specified, Essbase uses the level 0 members from the dimension tagged as Time.

Notes

• Financial functions never return a value; rather, they calculate a series of values internally based on the range specified.
• @ACCUM accepts the @ATTRIBUTE member set function as a member range.
• If you use an Essbase member set function to generate a member list for the rangeList parameter (for example, @SIBLINGS), to ensure correct results, consider the order in which Essbase sorts the generated member list. For more information, see the Technical Reference for Oracle Essbase topic for the member set function you are using.
• You cannot apply an operator (for example divide or multiply) to @Accum. For example, the formula Budget=@ACCUM(Actual, Jan:Feb)/2 is not valid.

Example

In this example, Accum Asset is calculated using the following formula:

"Accum Asset" = @ACCUM(Asset, FY1997:FY2002);

This example produces the following report. This report shows that the values for Asset are accumulated starting with FY1997 and the yearly accumulation value is placed in Accum Asset for FY1997 through FY2002:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Asset</td>
<td>9,000</td>
<td>0</td>
<td>1,000</td>
<td>0</td>
<td>2,500</td>
<td>1,500</td>
</tr>
<tr>
<td>Residual</td>
<td>750</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>#MISSING</td>
<td>#MISSING</td>
</tr>
<tr>
<td>Life</td>
<td>5</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>#MISSING</td>
<td>#MISSING</td>
</tr>
<tr>
<td>Accum Asset</td>
<td>#MISSING</td>
<td>#MISSING</td>
<td>1,000</td>
<td>1,000</td>
<td>3,500</td>
<td>5,000</td>
</tr>
</tbody>
</table>

The value of Accum Asset is #MISSING for FY1997 because that is the starting year. The value of Accum Asset is #MISSING for FY1998 because there was no accumulation that year. For FY1999, the value of the asset grew by 1,000, so Accum Asset has a value of 1000.

@ALLANCESTORS

Returns all ancestors of the specified member, including ancestors of any occurrences of the specified member as a shared member. This function excludes the specified member.
Syntax

@ALLANCESTORS (mbrName)

Parameters

mbrName
Any valid single member name, or a function that returns a single member.

Notes

- Essbase sorts the generated list of members in ascending order of the member number in the outline. Using Sample Basic as an example, if you specify 100-20 for mbrName, 100, Diet, and Product are returned (in that order). However, the order in which shared ancestors are returned is not guaranteed. This order is important to consider when you use the @ALLANCESTORS member set function with certain forecasting and statistical functions.
- You can use @ALLANCESTORS as a parameter of another function, where that parameter is a list of members.

Example

The following example is based on the Sample Basic database. Sample Basic has a shared level of diet drinks, which includes 100-20 (Diet Cola). So 100-20 (Diet Cola) is a descendant of 100 (Colas) and is a shared member descendant of Diet:

```
100
  100-10
  100-20
  --
Diet
  100-20 (Shared Member)
  --
```

The following calculation script increases by 5% the Budget->Sales values of all ancestors of 100-20, including Diet.

```
FIX(Budget,@ALLANCESTORS("100-20"))
Sales = Sales * 1.05;
ENDFIX
```

This example produces the following report. This report shows that the Budget->Sales values for 100, Diet, and Product (the ancestors of 100-20) have been increased by 5%. The original values were 8980, 8260, and 28480, respectively.

<table>
<thead>
<tr>
<th></th>
<th>Actual Sales</th>
<th>Budget Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>Market</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100-10</td>
<td>4860</td>
<td>5200</td>
</tr>
<tr>
<td>100-20</td>
<td>2372</td>
<td>2610</td>
</tr>
<tr>
<td>100-30</td>
<td>1082</td>
<td>1170</td>
</tr>
</tbody>
</table>
Returns the alias name, as a string, for the specified member name.

Syntax

@ALIAS (mbrName [, altName])

Parameters

mbrName
Any valid member name, or a function returning a member.

altName
Optional. Alias table name. This parameter is case insensitive.

Notes

• If no alias name is found, this function returns an empty string.
• Because functions that take strings as arguments may not function correctly if the string matches a member alias, use the function @ALIAS to pass member alias names as strings, for example when passing alias names as strings to functions such as @ISUDA, @UDA, @CONCATENATE, @SUBSTRING, @MATCH, or @NAME.

Example

The following example returns the alias of member "US$" from the alias table "Long Names."

IF(@ISUDA(@ALIAS("US$", "Long Names")))
... ENDIF

In the following example, assume "Book_Inventory" is a dimension name, and there are four alias tables in the outline ("Long Names" is one of them). The example code checks if the current member being calculated in the "Title" dimension has an alias
name in "Long Names" that matches with the UDA associated with the "Book_Inventory" dimension's currently calculating member.

@ISUDA("Book_Inventory",@ALIAS(@NAME(@CURRMBR("Title")), "Long Names"))

@ALLOCATE

Allocates values from a member, from a cross-dimensional member, or from a value across a member list. The allocation is based on a variety of criteria.

This function allocates values that are input at an upper level to lower-level members. The allocation is based upon a specified share or spread of another variable. For example, you can allocate values loaded to a parent member to all of that member’s children. You can specify a rounding parameter for allocated values and account for rounding errors.

Syntax

@ALLOCATE (amount, allocationRange, basisMbr, [roundMbr], method [, methodParams] [, round [, numDigits][, roundErr]])

Parameters

amount
A value, member, or cross-dimensional member that contains the value to be allocated into allocationRange. The value may also be a constant.

- If amount is a member, the member must be from the dimension to which allocationRange belongs.
- If amount is a cross-dimensional member, at least one of its members must be from the dimension to which allocationRange belongs.
- If no member or cross-dimensional member is from the dimension to which allocationRange belongs, a warning message is displayed.

If the amount parameter is a loaded value, it cannot be a Dynamic Calc member.

allocationRange
A comma-delimited list of members, member set functions, or range functions, into which value(s) from amount are allocated. allocationRange should be from only one level (for example, @CHILDREN(Total Expenses) rather than from multiple levels (for example, @DESCENDANTS(Product)).

basisMbr
A value, member, or cross-dimensional member that contains the values that provide the basis for the allocation. The method you specify determines how the basis data is used.

roundMbr
Optional. The member or cross-dimensional member to which rounding errors are added. The member (or at least one member of a cross-dimensional member) must be included in allocationRange.
method
The expression that determines how values are allocated. One of the following:

- **share**: Uses \textit{basisMbr} to calculate a percentage share. The percentage share is calculated by dividing the value in \textit{basisMbr} for the current member in \textit{allocationRange} by the sum across the \textit{allocationRange} for that basis member:

  \[
  \text{amount} \times \left( \frac{\text{CURRMBR}(\textit{basisMbr})}{\text{SUM}(\textit{allocationRange} \rightarrow \textit{basisMbr})} \right)
  \]

- **spread**: Spreads \textit{amount} across \textit{allocationRange}:

  \[
  \text{amount} \times \left( \frac{1}{\text{COUNT}(\text{SKIP}, \textit{allocationRange})} \right)
  \]

  SKIPNONE | SKIPMISSING | SKIPZERO | SKIPBOTH: Values to be ignored during calculation of the spread. You must specify a SKIP parameter only for \textit{spread}.

  - SKIPNONE: Includes all cells.
  - SKIPMISSING: Excludes all \#MISSING values in \textit{basisMbr}, and stores \#MISSING for values in \textit{allocationRange} for which the \textit{basisMbr} is missing.
  - SKIPZERO: Excludes all zero (0) values in \textit{basisMbr}, and stores \#MISSING for values in \textit{allocationRange} for which the \textit{basisMbr} is zero.
  - SKIPBOTH: Excludes all zero (0) values and all \#MISSING values, and stores \#MISSING for values in \textit{allocationRange} for which the \textit{basisMbr} is zero (0) or \#MISSING.

- **percent**: Takes a percentage value from \textit{basisMbr} for each member in \textit{allocationRange} and applies the percentage value to \textit{amount}:

  \[
  \text{amount} \times (\text{CURRMBR}(\textit{basisMbr}) \times .01)
  \]

- **add**: Takes the value from \textit{basisMbr} for each member of \textit{allocationRange} and adds the value to \textit{amount}:

  \[
  \text{amount} + \text{CURRMBR}(\textit{basisMbr})
  \]

- **subtract**: Takes the value from \textit{basisMbr} for each member of \textit{allocationRange} and subtracts the value from \textit{amount}:

  \[
  \text{amount} - \text{CURRMBR}(\textit{basisMbr})
  \]

- **multiply**: Takes the value from \textit{basisMbr} for each member of \textit{allocationRange} and multiplies the value by \textit{amount}:

  \[
  \text{amount} \times \text{CURRMBR}(\textit{basisMbr})
  \]

- **divide**: Takes the value from \textit{basisMbr} for each member of \textit{allocationRange} and divides the value by \textit{amount}:

  \[
  \text{amount} / \text{CURRMBR}(\textit{basisMbr})
  \]

**round**
Optional. One of the following:

- **noRound**: No rounding. noRound is the default.
• **roundAmt**: Indicates that you want to round the allocated values. If you specify `roundAmt`, you also must specify `numDigits` to indicate the number of decimal places to round to.

**numDigits**

An integer that represents the number of decimal places to round to. You must specify `numDigits` if you specify `roundAmt`.

- If `numDigits` is 0, the allocated values are rounded to the nearest integer. The default value for `numDigits` is 0.
- If `numDigits` is greater than 0, the allocated values are rounded to the specified number of decimal places.
- If `numDigits` is a negative value, the allocated values are rounded to a power of 10.

If you specify `roundAmt`, you also can specify a `roundErr` parameter.

**roundErr**

Optional. An expression that specifies where rounding errors should be placed. You must specify `roundAmt` in order to specify `roundErr`. If you do not specify `roundErr`, rounding errors are discarded.

To specify `roundErr`, choose from one of the following:

- `errorsToHigh`: Adds rounding errors to the member with the highest allocated value. If allocated values are identical, adds rounding errors to the first value in `allocationRange`. (For this option, Essbase does not distinguish between #MI and zero values.)
- `errorsToLow`: Adds rounding errors to the member with the lowest allocated value. If allocated values are identical, adds rounding errors to the first value in `allocationRange`. #MISSING is treated as the lowest value in a list; if multiple values are #MISSING, rounding errors are added to the first #MISSING value in the list.
- `errorsToMbr`: Adds rounding errors to the specified `roundMbr`, which must be included in `allocationRange`.

**Notes**

- When you use `@ALLOCATE` in a calculation script, use it within a FIX statement; for example, FIX on the member to which the allocation amount is loaded. Although FIX is not required, using it may improve calculation performance.
- If you use `@ALLOCATE` in a member formula, your formula should look like this:

  ```
  Member Name = @ALLOCATE (...)
  ```

  This is because allocation functions never return a value; rather, they calculate a series of values internally based on the range specified.
- For an example that explains the use of rounding error processing with the `@ALLOCATE` function, see Allocating Values within a Dimension in *Designing and Maintaining Essbase Cubes*.

**Example**

Consider the following example from the Sample Basic database. The example assumes that the Scenario dimension contains an additional member, PY Actual, for
the prior year's actual expenses. Data values of 7000 and 8000 are loaded into
Budget->Total Expenses for Jan and Feb, respectively. (For this example, assume that
Total Expenses is not a Dynamic Calc member.)

You need to allocate values to each expense category (to each child of Total
Expenses). The allocation for each of child of Total Expenses is based on the child's
share of actual expenses for the prior year (PY Actual):

FIX("Total Expenses")
Budget = @ALLOCATE(Budget->"Total Expenses",@CHILDREN("Total Expenses"),
"PY Actual",,share);
ENDFIX

This example produces the following report:

<table>
<thead>
<tr>
<th>Product</th>
<th>Market</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PY Actual</td>
</tr>
<tr>
<td></td>
<td>Jan</td>
</tr>
<tr>
<td>===</td>
<td>===</td>
</tr>
<tr>
<td>Marketing</td>
<td>5223</td>
</tr>
<tr>
<td>Payroll</td>
<td>4056</td>
</tr>
<tr>
<td>Misc</td>
<td>75</td>
</tr>
<tr>
<td>Total Expenses</td>
<td>9354</td>
</tr>
</tbody>
</table>

See Also
• @CREATEBLOCK
• @MDALLOCATE

@ANCEST

Returns the ancestor at the specified generation or level of the current member being
calculated in the specified dimension. If you specify the optional mbrName, that
ancestor is combined with the specified member.

This member set function can be used as a parameter of another function, where that
parameter is a member or list of members.

Syntax

@ANCEST (dimName, genLevNum [, mbrName])

Parameters

dimName
Single dimension name specification.

genLevNum
An integer value that defines the generation or level number from which the ancestor
value is returned. A positive integer defines a generation number. A value of 0 or a
negative integer defines a level number.
**mbrName**
Optional. Any valid single member name, or a function that returns a single member. This member is crossed with the ancestor returned.

**Notes**
- You cannot use the @ANCEST function in a FIX statement.
- You can use the @ANCEST function on both the left-hand and right-hand sides of a formula. If you use this function on the left-hand side of a formula in a calculation script, associate it with a member. For example:

  `Sales(@ANCEST(Product) = 5);`

- In some cases, the @ANCEST function is equivalent to the @ANCESTVAL function, except in terms of calculation performance. For example, the following two formulas are equivalent:

  `Sales = @ANCEST(Product,2);`

  `Sales = @ANCESTVAL(Product,2);`

  In this case, using the latter formula results in better calculation performance. In general, use @ANCEST as a member rather than as an implied value of a cell. For example:

  `Sales = @AVG(SKIPMISSING, @ISIBLINGS(@ANCEST(Product,2)));`

- The time required for retrieval and calculation may be significantly longer if this function is in a formula attached to a member tagged as Dynamic Calc or Dynamic Calc and Store.

**Example**

In the Sample Basic database:

<table>
<thead>
<tr>
<th>Function</th>
<th>Generated List</th>
</tr>
</thead>
<tbody>
<tr>
<td>@ANCEST(Product,2,Sales)</td>
<td>Colas-&gt;Sales, if the current member of Product being calculated is Diet Cola.</td>
</tr>
<tr>
<td>@ANCEST(Measures,3,East)</td>
<td>Total Expenses-&gt;East, if the current Measures being calculated is Payroll.</td>
</tr>
</tbody>
</table>

**See Also**
- @ANCESTORS
- @CHILDREN
- @DESCENDANTS
@ANCESTORS

Returns all ancestors of the specified member (mbrName) or those up to a specified generation or level. You can use this member set function as a parameter of another function, where that parameter is a list of members.

Syntax

@ANCESTORS (mbrName [, genLevNum | genLevName])

Parameters

mbrName
Any valid single member name, or a function that returns a single member.

genLevNum
Optional. An integer value that defines the absolute generation or level number up to which to select the members. A positive integer defines a generation number. A value of 0 or a negative integer defines a level number.

genLevName
Optional. Level name or generation name up to which to select the members.

Notes

The generated list of members is sorted starting with the nearest ancestor of the member, followed by the next nearest ancestor of the member, and so on. Using Sample Basic as an example, if you specify @ANCESTORS(200-30), Essbase returns 200, Product (in that order). This order is important to consider when you use the @ANCESTORS member set function with certain forecasting and statistical functions.

Example

In the Sample Basic database:

@ANCESTORS("New York")

returns East, Market (in that order).

@ANCESTORS(Qtr4)

returns Year.

@ANCESTORS("100-10",1)

returns 100, Product (in that order).

@ANCESTORS(Sales,-2)

returns Margin, Profit (in that order).
See Also

• @CHILDREN
• @DESCENDANTS
• @IANCESTORS
• @ILANCESTORS
• @ISANCEST
• @LANCESTORS
• @SIBLINGS

@ANCESTVAL

Returns the ancestor values of a specified member combination.

Syntax

@ANCESTVAL (dimName, genLevNum [, mbrName])

Parameters

dimName
A single dimension name that defines the focus dimension of ancestor values.

genLevNum
Integer value that defines the generation or level number from which the ancestor values are to be returned. A positive integer defines a generation reference. A negative number or value of 0 defines a level reference. To use this function or any other ancestor value function in a ragged hierarchy, use generation references instead of level references to avoid unexpected results.

mbrName
Optional. Any valid single member name or member combination (or a function that returns a single member or member combination).

Example

In this example, SKU Share is derived by taking Sales in each SKU as a percentage of its product family. Families are at generation 2; therefore, each descendant of family is calculated as a percentage its respective ancestor. Consolidated results must be calculated for Sales by Product before the SKU Share calculation occurs.

"SKU Share" = Sales % @ANCESTVAL(Product,2,Sales);

This example produces the following report:

<table>
<thead>
<tr>
<th>Sales</th>
<th>SKU Share</th>
</tr>
</thead>
<tbody>
<tr>
<td>SKU101</td>
<td>510</td>
</tr>
<tr>
<td></td>
<td>26.0</td>
</tr>
<tr>
<td>SKU102</td>
<td>520</td>
</tr>
<tr>
<td></td>
<td>26.5</td>
</tr>
<tr>
<td>Group01</td>
<td>1030</td>
</tr>
<tr>
<td></td>
<td>52.5</td>
</tr>
</tbody>
</table>
SKU120          430        21.9  
SKU123          500        25.5  
Group02       930        47.4  
Family1       1960      100.00

See Also

•  @MDANCESTVAL
•  @SANCESTVAL

@ATTRIBUTE

Lists all base members that are associated with the specified attribute member (attmbrName). Can be used as a parameter of another function, where that parameter is a member or list of members.

Syntax

@ATTRIBUTE (attMbrName)

Parameters

attMbrName

Single attribute member name.

Notes

When used with a non-level 0 member of an attribute dimension, this function returns all base members that are associated with the children of the attribute member. For example, in the Sample Basic database, @ATTRIBUTE(Large) returns all base members that fall into one of the population ranges for the attribute parent Large.

If you specify the name of a Boolean attribute dimension (for example, Caffeinated), this function returns all base members that are associated with either Caffeinated member (for example, True or False). To return only one, specify the member name (for example, @ATTRIBUTE(Caffeinated_True)).

You may have duplicate Boolean, date, and numeric attribute member names in your outline. For example, 12 can be the attribute value for the size (in ounces) of a product as well as the value for the number of packing units for a product. To distinguish duplicate member names, specify the full attribute member name (for example, @ATTRIBUTE(Ounces_12)).

The generated list of members is sorted in ascending order from the database outline. This order is important to consider when you use this function with certain forecasting and statistical functions.

Example

In the Sample Basic database,

@ATTRIBUTE(Can);

returns all base members with the Can attribute: Cola, Diet Cola, and Diet Cream.
Consider the following two calculation scripts, which are based on the Sample Basic database:

/* To increase the marketing budget for markets with large populations */
FIX (@ATTRIBUTE(Large))
Marketing = Marketing * 1.1;
ENDFIX

/* To calculate the average sales of bottled products */
"Bottle Sales" = @AVG(SKIPBOTH,@ATTRIBUTE(Bottle));

See Also
• @ATTRIBUTEVAL
• @WITHATTR

@ATTRIBUTEBVAL

Returns, for the current member being calculated, the associated attribute value from the specified Boolean attribute dimension.

Syntax

@ATTRIBUTEBVAL (attDimName)

Parameters

attDimName
The name of a Boolean attribute dimension.

Notes

• This function works only with Boolean attribute dimensions. To return values from numeric or date attribute dimensions, use @ATTRIBUTEVAL. To return values from text attribute dimensions, use @ATTRIBUTESVAL.
• If no attribute is associated with the member being calculated or if the attribute associated with the member is a text, numeric, or date attribute, this function returns #MISSING.
• Only level 0 members of attribute dimensions can be associated as attributes of members of a base dimension.

Example

This example is based on the Sample Basic database.

The Product dimension is associated with the Caffeinated Boolean attribute dimension, as shown in the following example:

Product {Caffeinated}
  100
    100-10 {Caffeinated:True}
    100-20 {Caffeinated:True}
For the current member of the base dimension Product, the function @ATTRIBUTEBVAL(Caffeinated) returns the associated attribute value from the Boolean attribute dimension, Caffeinated. The following table shows the value that would be returned.

Table 2-16  Value Returned by @ATTRIBUTEBVAL(Caffeinated) Function

<table>
<thead>
<tr>
<th>Current Member</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>100-10</td>
<td>True</td>
</tr>
<tr>
<td>100-20</td>
<td>True</td>
</tr>
<tr>
<td>100-30</td>
<td>False</td>
</tr>
<tr>
<td>100-30</td>
<td>#MISSING</td>
</tr>
<tr>
<td>200-10</td>
<td>True</td>
</tr>
<tr>
<td>200-20</td>
<td>True</td>
</tr>
<tr>
<td>200-30</td>
<td>False</td>
</tr>
<tr>
<td>200-40</td>
<td>False</td>
</tr>
<tr>
<td>200-40</td>
<td>#MISSING</td>
</tr>
<tr>
<td>Product</td>
<td>#MISSING</td>
</tr>
</tbody>
</table>

For any member that does not have an associated attribute, #MISSING is returned. Only one value is returned at a time.

See Also
- @ATTRIBUTEVAL
- @ATTRIBUTESVAL

@ATTRIBUTESVAL

Returns, for the current member being calculated, the associated attribute value from the specified text attribute dimension.

Syntax

@ATTRIBUTESVAL (attDimName)

Parameters

attDimName
The name of a text attribute dimension.
Notes

- This function works only with text attribute dimensions. To return values from numeric or date attribute dimensions, use `@ATTRIBUTEVAL`. To return values from Boolean attribute dimensions, use `@ATTRIBUTEBVAL`.

- If no attribute is associated with the member being calculated or if the attribute associated with the member is a numeric, Boolean, or date attribute, this function returns an empty string.

- Only level 0 members of attribute dimensions can be associated as attributes of members of a base dimension.

Example

This example is based on the Sample Basic database.

The Product dimension is associated with the Pkg Type text attribute dimension, as shown in the following example:

```
Product {Pkg Type}
  100
    100-10 {Pkg Type:Can}
    100-20 {Pkg Type:Can}
    100-30 {Pkg Type:Bottle}
  200
    200-10 {Pkg Type:Bottle}
    200-20 {Pkg Type:Bottle}
    200-30 {Pkg Type:Bottle}
    200-40 {Pkg Type:Bottle}
Pkg Type Attribute {Type: Text}
  Bottle
  Can
```

For the current member of the base dimension, Product, `@ATTRIBUTESVAL("Pkg Type")` returns the associated attribute value from the text attribute dimension, Pkg Type. The following table shows the value that would be returned:

<table>
<thead>
<tr>
<th>Current Member</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>100-10</td>
<td>Can</td>
</tr>
<tr>
<td>100-20</td>
<td>Can</td>
</tr>
<tr>
<td>100-30</td>
<td>Bottle</td>
</tr>
<tr>
<td>100</td>
<td>(empty string)</td>
</tr>
<tr>
<td>200-10</td>
<td>Bottle</td>
</tr>
<tr>
<td>200-20</td>
<td>Bottle</td>
</tr>
<tr>
<td>200-30</td>
<td>Bottle</td>
</tr>
<tr>
<td>200-40</td>
<td>Bottle</td>
</tr>
<tr>
<td>200</td>
<td>(empty string)</td>
</tr>
<tr>
<td>Product</td>
<td>(empty string)</td>
</tr>
</tbody>
</table>
For any member that does not have an associated attribute, an empty string is returned.

See Also

- @ATTRIBUTEVAL
- @ATTRIBUTEBVAL

@ATTRIBUTEVAL

Returns, for the current member being calculated, the associated attribute value from the specified numeric or date attribute dimension.

Syntax

@ATTRIBUTEVAL (attDimName)

Parameters

attDimName
Single dimension specification for a numeric or date attribute dimension.

Notes

- This function works only with numeric and date attribute dimensions. To return values from text attribute dimensions, use @ATTRIBUTESVAL. To return values from Boolean attribute dimensions, use @ATTRIBUTEBVAL.
- Only level 0 members of attribute dimensions can be associated as attributes of members of a base dimension.
- If a text attribute, or no attribute, is associated with the member being calculated, this function returns #MISSING.
- When this function is used with a date attribute dimension, it converts the date string to the number of seconds elapsed since midnight, January 1, 1970.

Example

Example 1

The following example is based on the Sample Basic database:

"Profit Per Ounce" = Profit/@ATTRIBUTEVAL(NAME(Ounces));

In this formula, for the current member being calculated, @ATTRIBUTEVAL returns the associated attribute from the Ounces numeric attribute dimension. For example, if the member being calculated is Cola and if the Ounces attribute value associated with Cola is 12, @ATTRIBUTEVAL returns 12. The value returned is then divided into Profit to yield Profit Per Ounce.
Note:

@NAME is required to process the string “Ounces” before passing it to @ATTRIBUTEVAL.

This example produces the following report:

<table>
<thead>
<tr>
<th>Actual</th>
<th>Year</th>
<th>West</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Profit Per Ounce</td>
</tr>
<tr>
<td>========</td>
<td>======</td>
<td>---------------</td>
</tr>
<tr>
<td>Cola</td>
<td>4593</td>
<td>382.75</td>
</tr>
</tbody>
</table>

Example 2

The following MaxL execute calculation statement applies a formula to members that are 16 Oz products:

```maxl
execute calculation
'Misc
  { IF
    (@ATTRIBUTEVAL(Ounces) == 16)
      Misc = .5;
    ENDIF;
  };
' on sample.basic;
```

@AVG

Returns the average of all values in expList.

Syntax

@AVG (SKIPNONE | SKIPMISSING | SKIPZERO | SKIPBOTH, expList)

Parameters

**SKIPNONE**
Includes all cells specified in the average operation regardless of their content.

**SKIPMISSING**
Excludes all values that are #MISSING in the average operation.

**SKIPZERO**
Excludes values of zero from the average calculation.

**SKIPBOTH**
Excludes all values of zero or #MISSING from the average calculation.
expList
Comma-delimited list of member names, variable names, functions, or numeric expressions. expList provides a list of numeric values across which the average is calculated.

Example
The following example is based on the Sample Basic database. The calculation averages the values for the individual states making up the western region and places the results in West:

```plaintext
FIX(Sales)
West=@AVG(SKIPBOTH,California:Nevada);
ENDFIX
```

This example produces the following report:

<table>
<thead>
<tr>
<th></th>
<th>Sales</th>
<th>Jan</th>
<th>Actual</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cola</td>
<td>Cola</td>
<td>Diet Cola</td>
<td>Caffeine Free Cola</td>
</tr>
<tr>
<td>California</td>
<td>678</td>
<td>118</td>
<td>145</td>
</tr>
<tr>
<td>Oregon</td>
<td>160</td>
<td>140</td>
<td>150</td>
</tr>
<tr>
<td>Washington</td>
<td>130</td>
<td>190</td>
<td>#MI</td>
</tr>
<tr>
<td>Utah</td>
<td>130</td>
<td>190</td>
<td>170</td>
</tr>
<tr>
<td>Nevada</td>
<td>76</td>
<td>62</td>
<td>#MI</td>
</tr>
<tr>
<td>West</td>
<td>234.8</td>
<td>140</td>
<td>155</td>
</tr>
</tbody>
</table>

See Also
@AVGRANGE

@AVGRANGE

Returns the average value of the specified member (mbrName) across the specified range (XrangeList).

Syntax

```
@AVGRANGE ( SKIPNONE | SKIPMISSING | SKIPZERO | SKIPBOTH, mbrName [, XrangeList])
```

Parameters

**SKIPNONE**
Includes all cells specified in the average operation regardless of their content.

**SKIPMISSING**
Excludes all values that are #MISSING in the average operation.

**SKIPZERO**
Excludes values of zero from the average calculation.
**SKIPBOTH**

Excludes all values of zero or #MISSING from the average calculation.

**mbrName**

Any valid single member.

**XrangeList**

Optional. A valid member name, a comma-delimited list of member names, cross dimension members, or a member set function or range function (including @X RANGE) that returns a list of members from the same dimension. If XrangeList is not specified, Essbase uses the level 0 members from the dimension tagged as Time.

**Notes**

This function accepts @ATTRIBUTE as a member range.

**Example**

The following example is based on the Sample Basic database. The calculation script determines the average sales of Colas in the West.

```plaintext
FIX(Sales)
West=@AVGRANGE(SKIPNONE,Sales,@CHILDREN(West));
ENDFIX
```

This example produces the following report:

<table>
<thead>
<tr>
<th></th>
<th>Colas</th>
<th>Actual</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jan</td>
<td>Feb</td>
<td>Mar</td>
</tr>
<tr>
<td>===</td>
<td>===</td>
<td>===</td>
</tr>
<tr>
<td>California</td>
<td>941</td>
<td>899</td>
</tr>
<tr>
<td>Oregon</td>
<td>450</td>
<td>412</td>
</tr>
<tr>
<td>Washington</td>
<td>320</td>
<td>362</td>
</tr>
<tr>
<td>Utah</td>
<td>490</td>
<td>488</td>
</tr>
<tr>
<td>Nevada</td>
<td>138</td>
<td>137</td>
</tr>
<tr>
<td>West</td>
<td>467.8</td>
<td>459.6</td>
</tr>
</tbody>
</table>

**See Also**

@AVG

**@BETWEEN**

Returns a member set of all members whose name string value fall between, and are inclusive of, the two specified string tokens. Member names are evaluated alphanumerically.

This function can be used on unique and duplicate-name outlines.

**Syntax**

```plaintext
@BETWEEN (firstToken , secondToken, topMbrInHierarchy)
```
Parameters

firstToken
First token string value with which to compare to members in the outline, starting with
the member specified in topMbr.

secondToken
Second token string value with which to compare to members in the outline, starting
with the member specified in topMbr.

topMbrInHierarchy
A fully qualified name of a member in the outline on which to base the member
search. The specified member and its aliases, and all of its descendants, are included
in the search.
To search the entire outline, provide an empty string ("") for this parameter. For
example, @BETWEEN("200-10","200-20", ").

Example
The following example is based on the following duplicate-name outline:

Product
  100
    100-10
    100-10-10
    100-20
    100-30
  200
    200-10
    200-20
    200-30
  300
    300-10
    300-20
Diet
  100-10
    100-10-11
  200-10
  300-10
Bottle
  200-10
  300-20

@BETWEEN("200-10", "200-20", "Product")

Returns the members [200].[200-10], [200].[200-20], [Diet].
[200-10], and [Bottle].[200-10].

@CALCMODE

Enables the choice of an execution mode of a formula. @CALCMODE can control two
types of modes:
• Whether a formula is calculated in block calculation or cell calculation mode when calculating formulas that contain certain functions (for example, @ISMBR)

• Whether a formula assigned to a sparse member is calculated in bottom-up or top-down mode

Understanding Block Calculation and Cell Calculation Modes

Using block calculation mode, Essbase groups the cells within a block and simultaneously calculates the cells in each group. Block calculation mode is fast, but you must carefully consider data dependencies within the block to ensure that the resulting data is accurate.

Using cell calculation mode, Essbase calculates each cell sequentially, following the calculation order, which is based on the order of the dense dimensions in the outline. For more information on calculation order, see Designing and Maintaining Essbase Cubes.

Understanding Bottom-Up and Top-Down Calculation Modes

Essbase uses one of two methods to do a full calculation of an outline: bottom-up calculation (the default) or top-down calculation. If the outline contains a complex member formula, Essbase performs a top-down calculation for that member. When a formula is compiled, if the formula is to be calculated top-down, Essbase logs a message in the application log file.

For a bottom-up calculation, Essbase determines which existing data blocks need to be calculated before it calculates the database. Essbase then calculates only the blocks that need to be calculated during the full database calculation. The calculation begins with the lowest existing block number and works up through each subsequent block until the last existing block is reached.

In contrast, a top-down calculation calculates the formula on all potential datablocks with the member. A top-down calculation may be less efficient than a bottom-up calculation because more blocks may be calculated than is necessary. Although a top-down calculation is less efficient than a bottom-up calculation, in some cases top-down calculations are necessary to ensure that calculation results are correct. See Example 4.

Syntax

@CALCMODE (CELL|BLOCK|TOPDOWN|BOTTOMUP)

Parameters

CELL
Turns on the cell calculation mode

BLOCK
Turns on the block calculation mode

TOPDOWN
Turns on the top-down calculation mode

BOTTOMUP
Turns on the bottom-up calculation mode
Notes

Cell and block modes are mutually exclusive. Top-down and bottom-up modes are mutually exclusive. Within one @CALCMODE specification, you can specify only one option. To specify both types of modes, perform the instruction twice; for example:

```plaintext
@CALCMODE (CELL)
@CALCMODE (TOPDOWN)
```

Knowing When Essbase uses Cell or Block Mode and Top-down or Bottom-up Mode

- When Essbase compiles a formula, it prints a message in the application log file explaining the mode of execution for the formula similar to the following message:

  Formula on member Profit % will be executed in CELL and TOPDOWN mode.

When Essbase determines that the formula will be executed in block and bottom-up mode, no message is written in the application log file.

- In calculation scripts, @CALCMODE statements must be placed within parentheses and associated with a specific database member.

- By default, for a simple formula such as \( A = B + C \), Essbase does a bottom-up calculation. \( A \) is calculated only if \( B \) or \( C \) exists in the database. The dependency of the formula on \( B \) and \( C \) is known before the calculation is started.

  For a complex formula such as \( A = B->D + C->D \), Essbase performs a top-down calculation because every possible combination of \( A \) must be examined to see whether \( B->D \) or \( C->D \) exists.

- By default, Essbase uses cell calculation mode for formulas containing:
  - @ANCEST
  - @CURRMBR
  - @ISMBR on a dense member
  - @MDANCESTVAL
  - @MDPARENTVAL
  - @MDSHIFT
  - @NEXT
  - @PARENT
  - @PARENTVAL
  - @PRIOR
  - @SANCESTVAL
  - @SPARENTVAL
  - @SHIFT
  - @XWRITE

For all other formulas, Essbase uses block calculation mode by default.
Understanding Data Dependency Issues With Block Calculation Mode

Data dependency occurs if the accurate calculation of one or more members depends on another member or other on members being calculated previously. Most data dependency issues with block calculation mode occur when a formula contains IF ELSE or IF ELSEIF conditions. However, data dependencies can occur in other formulas; for example, when using the @PRIOR function.

Data Dependency Issues With IF ELSE and IF ELSEIF

When Essbase uses block calculation mode to calculate a formula that contains IF ELSE or IF ELSEIF conditions, it separates the members being calculated into two groups. The first group contains the members that satisfy the IF condition. The second group contains the members that satisfy the ELSE or ELSEIF conditions.

Essbase simultaneously calculates the members in the first group before simultaneously calculating the members in the second group. See Example 1.

If a formula contains data dependencies, ensure that the following conditions are met:

- Members on which the accurate calculation of other members depends are in the first group.
- Dependent members are in the second group.

If an IF condition has multiple ELSEIF conditions, Essbase evaluates each ELSEIF condition, placing the members that satisfy the ELSEIF condition in the first group and the members that satisfy subsequent ELSEIF or ELSE conditions in the second group. See Example 2.

Understanding Other Data Dependency Issues

Data dependencies can occur in formulas that do not contain IF ELSE conditions. See Example 3 for an example of data dependency in a formula containing @PRIOR.

Example

Example 1, Example 2, and Example 3 illustrate use of the BLOCK and CELL options of @CALCMODE. Example 4 illustrates use of the BOTTOMUP and TOPDOWN options.

Example 1

Consider a database with two dense dimensions, Time and Accounts. The following formula is placed on the Budget Sales member of the Accounts dimension. Because this is a formula containing @ISMBR applied to a dense member (Budget Sales), by default Essbase uses cell calculation mode. Use @CALCMODE(BLOCK) to specify block calculation mode for this formula.

```plaintext
@CALCMODE(BLOCK);
IF(@ISMBR(Feb))
    "Budget Sales"=100;
ELSE
    "Budget Sales"=Feb+10;
```

According to the above formula, we expect that if the member being calculated is Feb, the Budget Sales value is 100. If the member being calculated is not Feb, the Budget Sales value is 100+10 (the value for Feb + 10).
Assume that we load the values 10, 20, and 30 into the Budget Sales data block for Jan, Feb and Mar, as follows:

Table 2-18   Values loaded in the Budget Sales Data Block

<table>
<thead>
<tr>
<th>(axis)</th>
<th>Jan</th>
<th>Feb</th>
<th>Mar</th>
</tr>
</thead>
<tbody>
<tr>
<td>Budget Sales</td>
<td>10</td>
<td>20</td>
<td>30</td>
</tr>
</tbody>
</table>

Using block calculation mode, Essbase calculates the members satisfying the IF condition first. In this example, Feb is the only member that satisfies the IF condition. After calculating Feb, Essbase calculates the members Jan and Mar. In this example, the results are as expected:

Table 2-19   Results of Block Calculation Mode

<table>
<thead>
<tr>
<th>(axis)</th>
<th>Jan</th>
<th>Feb</th>
<th>Mar</th>
</tr>
</thead>
<tbody>
<tr>
<td>Budget Sales</td>
<td>110</td>
<td>100</td>
<td>110</td>
</tr>
</tbody>
</table>

Example 2

Now consider the same database as in Example 1, but we place the following formula on the Budget Sales member of the Accounts dimension. As in Example 1, because this is a formula containing @ISMBR applied to a dense dimension member (Budget Sales), by default Essbase uses cell calculation mode. However, we use @CALCMODE(BLOCK) to specify the block calculation mode for this formula.

```plaintext
@CALCMODE(BLOCK);
IF(@ISMBR(Mar))
   "Budget"->"Sales"=Feb+20;
ELSEIF(@ISMBR(Jan))
   "Budget"->"Sales"=Feb+10;
ELSE
   "Budget"->"Sales"=100;
ENDIF
```

According to this formula, we want the Jan and Mar Budget Sales values to be calculated based on the Feb Budget Sales value, which is 100. We want to see the following results:

Table 2-20   Desired Calculation Results

<table>
<thead>
<tr>
<th>(axis)</th>
<th>Jan</th>
<th>Feb</th>
<th>Mar</th>
</tr>
</thead>
<tbody>
<tr>
<td>Budget Sales</td>
<td>110</td>
<td>100</td>
<td>120</td>
</tr>
</tbody>
</table>

Assume that we load the values 10, 20, and 30 into the Budget Sales data block for Jan, Feb, and Mar, as follows:
Using block calculation mode, Essbase calculates the members satisfying the IF condition first, followed by the members satisfying the ELSEIF condition, followed by the members satisfying the ELSE condition. In this example, Essbase calculates the members in the following order: Mar, Jan, Feb. The results are not what we want, because the calculation of Jan and Mar is dependent on the calculation of Feb, and Feb is calculated after Jan and Mar. The inaccurate results are as follows:

**Table 2-22 Inaccurate Calculation of Budget Sales Data Block**

<table>
<thead>
<tr>
<th>(axis)</th>
<th>Jan</th>
<th>Feb</th>
<th>Mar</th>
</tr>
</thead>
<tbody>
<tr>
<td>Budget Sales</td>
<td>30</td>
<td>100</td>
<td>40</td>
</tr>
</tbody>
</table>

To achieve the desired results, use @CALCMODE(CELL).

**Example 3**

The following formula calculates the members Opening Inventory and Ending Inventory using the @PRIOR function. There is a data dependency between Opening Inventory and Ending Inventory. The formula is placed on the Opening Inventory member. The example shows the results for January, February, and March.

```plaintext
@CALCMODE(BLOCK)
"Opening Inventory"=@PRIOR("Ending Inventory")+10;
"Ending Inventory"="Opening Inventory";
```

Before the calculation, there is no data for these members (the data is #MISSING or #MI):

**Table 2-23 Missing Data Before Inventory Calculation**

<table>
<thead>
<tr>
<th>(axis)</th>
<th>Jan</th>
<th>Feb</th>
<th>Mar</th>
</tr>
</thead>
<tbody>
<tr>
<td>Opening Inventory</td>
<td>#MI</td>
<td>#MI</td>
<td>#MI</td>
</tr>
<tr>
<td>Ending Inventory</td>
<td>#MI</td>
<td>#MI</td>
<td>#MI</td>
</tr>
</tbody>
</table>

Using block calculation mode, Essbase calculates the members simultaneously, taking the previous month's Ending Inventory #MISSING value as 0 for all member combinations and adding 10. This is not the desired result.

**Table 2-24 Inaccurate Results for Inventory Calculation**

<table>
<thead>
<tr>
<th>(axis)</th>
<th>Jan</th>
<th>Feb</th>
<th>Mar</th>
</tr>
</thead>
<tbody>
<tr>
<td>Opening Inventory</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>Ending Inventory</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
</tbody>
</table>
The following formula on the Opening Inventory member causes Essbase to use cell calculation mode (the default for formulas containing \texttt{@PRIOR}):

\begin{verbatim}
"Opening Inventory"=@PRIOR("Ending Inventory")+10;
"Ending Inventory"="Opening Inventory";
\end{verbatim}

The results are as follows:

\begin{table}[h]
\centering
\caption{Cell Calculation Mode Inventory Results}
\begin{tabular}{|c|c|c|c|}
\hline
\textbf{(axis)} & \textbf{Jan} & \textbf{Feb} & \textbf{Mar} \\
\hline
Opening Inventory & 10 & 20 & 30 \\
Ending Inventory & 10 & 20 & 30 \\
\hline
\end{tabular}
\end{table}

Example 4

Depending on the formula and the structure of the data, calculating a formula top-down versus bottom-up may involve two issues: performance (reflecting the number of calculations that must be made) and accuracy. This example compares calculation results to illustrate both of these issues.

Before the calculation, assume that Actual and Budget are members of a sparse dimension and they contain the following data:

\begin{table}[h]
\centering
\caption{Data for Actual and Budget Members}
\begin{tabular}{|c|c|c|}
\hline
\textbf{(axis)} & \textbf{Cola} & \textbf{New York Sales} \\
\hline
\textbf{(axis)} & \textbf{Actual} & \textbf{Budget} \\
Jan & #MISSING & 50 \\
Feb & 200 & #MISSING \\
Mar & 400 & 450 \\
\hline
\end{tabular}
\end{table}

The following formula is calculated bottom-up.

\begin{verbatim}
Budget(  
  @CALCMODE(BOTTOMUP);  
  Budget=Actual*1.10;
)
\end{verbatim}

In a bottom-up calculation, Essbase executes formulas only from existing data blocks. Therefore, only two values—Jan and Mar—are calculated, based on existing combinations of Budget.

\begin{table}[h]
\centering
\caption{Bottom-up Calculation Results for Actual and Budget}
\begin{tabular}{|c|c|c|c|}
\hline
\textbf{(axis)} & \textbf{Cola} & \textbf{New York Sales} & \textbf{(Comment)} \\
\hline
\textbf{(axis)} & \textbf{Actual} & \textbf{Budget} & - \\
Jan & #MISSING & #MISSING & (#MISSING*1.10) \\
\hline
\end{tabular}
\end{table}
The following formula is calculated top-down.

```
Budget(
    @CALCMODE(TOPDOWN);
    Budget=Actual*1.10;
)
```

In a top-down calculation, Essbase materializes every potential data block that is relevant to the calculation, and executes formulas in those blocks. Therefore, all three values—Jan, Feb, and Mar—are calculated, based on all potential combinations of Budget. The results are:

<table>
<thead>
<tr>
<th>(axis)</th>
<th>Cola</th>
<th>New York Sales</th>
<th>(Comment)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(axis)</td>
<td>Actual</td>
<td>Budget</td>
<td>-</td>
</tr>
<tr>
<td>Feb</td>
<td>200</td>
<td>#MISSING</td>
<td>(No calculation is performed)</td>
</tr>
<tr>
<td>Mar</td>
<td>400</td>
<td>440</td>
<td>(400*1.10)</td>
</tr>
</tbody>
</table>

Table 2-28  Top-down Calculation Results for Actual and Budget

<table>
<thead>
<tr>
<th>(axis)</th>
<th>Cola</th>
<th>New York Sales</th>
<th>(Comment)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(axis)</td>
<td>Actual</td>
<td>Budget</td>
<td>-</td>
</tr>
<tr>
<td>Jan</td>
<td>#MISSING</td>
<td>#MISSING</td>
<td>(#MISSING*1.10)</td>
</tr>
<tr>
<td>Feb</td>
<td>200</td>
<td>220</td>
<td>(200*1.10)</td>
</tr>
<tr>
<td>Mar</td>
<td>400</td>
<td>440</td>
<td>(400*1.10)</td>
</tr>
</tbody>
</table>

@CHILDREN

Returns all children of the specified member, excluding the specified member. This member set function can be used as a parameter of another function, where that parameter is a list of members.

Syntax

```
@CHILDREN (mbrName)
```

Parameters

**mbrName**

Any valid single member name, or a function that returns a single member.

Notes

This function sorts the child members in ascending order. Using Sample Basic as an example, if you specify 100 for *mbrName*, Essbase returns 100-10, 100-20, 100-30 (in that order). This order is important to consider when you use this function with certain forecasting and statistical functions.
Example

In the Sample Basic database:

@CHILDREN(Market)

returns East, West, South, and Central (in that order).

@CHILDREN(Margin)

returns Sales and COGS (in that order).

See Also

• @ANCESTORS
• @DESCENDANTS
• @ICHIILDREN
• @ISCHILD
• @SIBLINGS

@COMPOUND

Compiles the proceeds of a compound interest calculation. The calculation is based on the balances of the specified member at the specified rate across the specified range.

Syntax

@COMPOUND (balanceMbr, rateMbrConst [, XrangeList])

Parameters

balanceMbr
Single member specification representing the beginning balance across a range of periods. The input can be either one deposit or a series of deposits. If balanceMbr is a constant, then Essbase assumes balanceMbr to be a single deposit in the first member of rangeList or XrangeList. This is equivalent to entering the constant value in the first member in the range followed by zeros. The function keeps track of each deposit separately, but returns a composite value. If balanceMbr is a member, or a range, then it is assumed to be a series of deposits.

rateMbrConst
Single member specification, variable name, or numeric expression in decimal form. This represents the interest rate per time period specified in the rangeList or XrangeList. If your interest is compounded monthly, this value would be the annual interest rate divided by 12.

XrangeList
Optional parameter specifying the range over which the interest is compounded. The last value in the range is the total compounded interest for that range. If a range is not specified, Essbase uses the level 0 members from the dimension tagged as Time.
Can be a valid member name, a comma-delimited list of member names, cross dimensional members, or a return value from a member set function or range function (including @XRANGE).
For more information about rangeList and XrangeList, see Range List Parameters in the topic Range and Financial Functions.

Notes
Financial functions never return a value; rather, they calculate a series of values internally based on the range specified.

Example
The following example determines the compound interest of a series of deposits, based on a credit rate of 0.0525, across a series of fiscal years:

"Compound Interest"=@COMPOUND(Deposit,"Credit Rate",FY1998:FY2001,FY2002);

This example produces the following report:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Credit Rate</td>
<td>0.0525</td>
<td>0.0525</td>
<td>0.0525</td>
<td>0.0525</td>
<td>0.0525</td>
</tr>
<tr>
<td>Compound Interest</td>
<td>0</td>
<td>105</td>
<td>110.5125</td>
<td>273.8144</td>
<td>288.1897</td>
</tr>
<tr>
<td>Deposit</td>
<td>0</td>
<td>2,000</td>
<td>0</td>
<td>3,000</td>
<td>0</td>
</tr>
</tbody>
</table>

The following example assumes a Year dimension is added to Sample Basic. It calculates compound interest using a multidimensional range.

FIX ("100-10", "New York")
"Compound Interest" = @COMPOUND(Deposit,"Credit Rate",@XRANGE("2011"->"Sep", "2012"->"Mar"));
ENDFIX

The above calculation is performed across the following multidimensional range specified by XrangeList:

2011->Sep
2011->Oct
2011->Nov
2011->Dec
2012->Jan
2012->Feb
2012->Mar

See Also
@INTEREST
@COMPOUNDGROWTH

Calculates a series of values that represents a compound growth of values (the first nonzero value in the specified member across the specified range of members) across time.

The growth factor is calculated by multiplying the growth rate in the current time period by the previous period's result, yielding a compounded value. You can change the growth rate from period to period by placing a nonzero value in the current period's rateMbrConst cell.

Syntax

@COMPOUNDGROWTH (principalMbr, rateMbrConst [, XrangeList])

Parameters

principalMbr
Member specification representing the initial value to be compounded. The input line must be a single deposit.

rateMbrConst
Single member specification, variable name, or expression which provides a constant value. This value can change across rangeList, making the new value be the new compound rate. If the value in the current period is zero, the compound rate is equal to zero, and the principal does not change.

XrangeList
Optional parameter specifying the time period over which the interest is calculated. If a range is not specified, Essbase uses the level 0 members from the dimension tagged as Time. Can be a valid member name, a comma-delimited list of member names, cross dimensional members, or a return value from a member set function or range function (including @XRANGE). For more information about rangeList and XrangeList, see Range List Parameters in the topic Range and Financial Functions.

Notes

Financial functions never return a value; rather, they calculate a series of values internally based on the range specified.

Example

The following example determines the compound growth of Principal Amount based on Growth Rate across a series of fiscal years.

"Compound Growth"=@COMPOUNDGROWTH("Principal Amount", "Growth Rate",FY1998:FY2003);

This example produces the following report:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>======</td>
<td>======</td>
<td>======</td>
<td>======</td>
<td>======</td>
<td>======</td>
</tr>
</tbody>
</table>
### Calculation Function List

#### 2-53

<table>
<thead>
<tr>
<th>Principal Amount</th>
<th>2,000</th>
<th>2,000</th>
<th>2,000</th>
<th>3,000</th>
<th>2,500</th>
<th>-500</th>
</tr>
</thead>
<tbody>
<tr>
<td>Growth Rate</td>
<td>0.0525</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Compound Growth</td>
<td>2,105</td>
<td>2,105</td>
<td>2,105</td>
<td>2,105</td>
<td>2,105</td>
<td>2,105</td>
</tr>
</tbody>
</table>

The following example assumes a Year dimension is added to Sample Basic. It calculates compound growth using a multidimensional range.

```plaintext
FIX("100-10", "New York")
"Compound Growth" = @COMPOUNDGROWTH("Principal Amount","Growth Rate",@XRANGE("2011"->"Sep", "2012"->"Mar");
ENDFIX
```

The above calculation is performed across the following multidimensional range specified by `XrangeList`:

- 2011->Sep
- 2011->Oct
- 2011->Nov
- 2011->Dec
- 2012->Jan
- 2012->Feb
- 2012->Mar

**See Also**

@GROWTH

### @CONCATENATE

Returns one character string that is the result of appending one character string (`String2`) to the end of another character string (`String1`). This function can be nested to concatenate more than two strings (See Example 2 (@CONCATENATE)).

**Syntax**

```
@CONCATENATE (String1, String2)
```

**Parameters**

- **String1**: A string or a function that returns a string
- **String2**: A string or a function that returns a string

**Notes**

- To use a member name as a character string, use `@NAME` with the member name.
• To use the resulting character string as a member name, use \texttt{@MEMBER} with \texttt{@CONCATENATE}; for example,

\begin{verbatim}
\texttt{@MEMBER(\texttt{@CONCATENATE("2000\_", QTR1)));
\end{verbatim}

Example

The following examples are based on the Sample Basic database:

Example 1 (\texttt{@CONCATENATE})

The following function statement puts the string Item in front of the name of the member currently being processed in the Product dimension; for example, if the current member being calculated is 100-10, the result is Item100-10:

\begin{verbatim}
\texttt{@CONCATENATE("Item",\texttt{@NAME(\texttt{@CURRMBR(Product))})
\end{verbatim}

Example 2 (\texttt{@CONCATENATE})

To concatenate more than two strings, you can nest multiple instances of the \texttt{@CONCATENATE} function. The following function statement returns string values starting with the current member of the Year dimension, followed by an underscore, followed by the current member of the Measures dimension; for example, if the current members being calculated are Qtr1 and Sales, the result is Qtr1_Sales:

\begin{verbatim}
\texttt{@CONCATENATE(\texttt{@NAME(\texttt{@CURRMBR(Year))}},\texttt{@CONCATENATE("\_",\texttt{@NAME(\texttt{@CURRMBR(Measures}})
\end{verbatim}

See Also

• \texttt{@MEMBER}
• \texttt{@NAME}
• \texttt{@SUBSTRING}

@CORRELATION

Returns the correlation coefficient between two parallel data sets (\texttt{XrangeList1} and \texttt{XrangeList2}). The correlation coefficient determines the relationship between two data sets.

Syntax

\begin{verbatim}
@CORRELATION (\texttt{SKIPNONE} | \texttt{SKIPMISSING} | \texttt{SKIPZERO} | \texttt{SKIPBOTH}, \texttt{XrangeList1}, \texttt{XrangeList2})
\end{verbatim}

Parameters

\texttt{SKIPNONE}

Includes all cells specified in the two data sets, regardless of their content, during calculation of the correlation coefficient.
**SKIPMISSING**
Excludes all #MISSING values from the two data sets during calculation of the correlation coefficient.

**SKIPZERO**
Excludes all zero (0) values from the two data sets during calculation of the correlation coefficient.

**SKIPBOTH**
Excludes all zero (0) values and #MISSING values from the two data sets during calculation of the correlation coefficient.

**XrangeList1**
The first of two parallel data sets.
Can be a valid member name, a comma-delimited list of member names, cross dimensional members, or a return value from a member set function or range function (including @XRANGE).
For more information about XrangeList, see Range List Parameters in the topic Range and Financial Functions.

**XrangeList2**
The second of two parallel data sets.
Can be a valid member name, a comma-delimited list of member names, cross dimensional members, or a return value from a member set function or range function (including @XRANGE).

**Notes**
- For complete information about using the @RANGE function, see @RANGE. For more information about XrangeList, see Range List Parameters in the topic Range and Financial Functions.
- The XrangeList1 and XrangeList2 parameters must have the same number of data points. If the two data sets have different numbers of data points, this function returns #MISSING.
- This function returns #MISSING if XrangeList1 and XrangeList2 (1) are empty, (2) contain only #MISSING values, or (3) have a standard deviation of 0 (all values are constant).
- This function treats #MISSING values as zero (0) values, unless SKIPMISSING or SKIPBOTH is specified. If a value in XrangeList1 is #MISSING, and SKIPMISSING is specified, the value's corresponding value in XrangeList1 is treated as #MISSING. (That is, both values are deleted before calculation.) SKIPZERO and SKIPBOTH work similarly.
- This function returns values from -1 to 1.
- If you use a member set function to generate a member list for this function (for example, @SIBLINGS), to ensure correct results, consider the order in which Essbase sorts the generated member list. For more information, see the topic for the member set function you are using.
- The equation for the correlation coefficient is:
\[ \rho_{x,y} = \frac{\text{Cov}(X,Y)}{\sigma_X \sigma_Y} \]

so that

\[-1 \leq \rho_{x,y} \leq 1 \]

and

\[ \text{Cov}(X,Y) = \frac{1}{n} \sum_{i=1}^{n} (x_i - \mu_X)(y_i - \mu_Y) \]

\(\sigma_x\) stands for the standard deviation of \(X = \{x_i\}_{i=1}^n\)

\(\sigma_y\) stands for the standard deviation of \(Y = \{y_i\}_{i=1}^n\)

**Example**

The following example is based on the Sample Basic database. Assume that the Measures dimension contains an additional member, Sales Correl. The calculation script calculates the correlation coefficient for a set of members (Sales for the children of Qtr1 and Qtr2). Because the calculation script fixes on Jun, the results are placed in Sales Correl->Jun.

This example uses the @RANGE function to generate XrangeList1 and XrangeList2:

```
FIX(June)
"Sales Correl"=@CORRELATION(SKIPNONE, @RANGE(Sales,@CHILDREN(Qtr1)),@RANGE(Sales,@CHILDREN(Qtr2))); ENDFIX
```

This example produces the following report:

<table>
<thead>
<tr>
<th></th>
<th>Actual</th>
<th>New York</th>
<th>Sales Correl</th>
</tr>
</thead>
<tbody>
<tr>
<td>Colas</td>
<td>Sales</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>======</td>
<td>=========</td>
<td>=============</td>
</tr>
<tr>
<td>Jan</td>
<td>678</td>
<td>#MI</td>
<td></td>
</tr>
<tr>
<td>Feb</td>
<td>645</td>
<td>#MI</td>
<td></td>
</tr>
<tr>
<td>Mar</td>
<td>675</td>
<td>#MI</td>
<td></td>
</tr>
<tr>
<td>Apr</td>
<td>712</td>
<td>#MI</td>
<td></td>
</tr>
<tr>
<td>May</td>
<td>756</td>
<td>#MI</td>
<td></td>
</tr>
<tr>
<td>Jun</td>
<td>890</td>
<td>0.200368468</td>
<td></td>
</tr>
</tbody>
</table>
The following example assumes a Year dimension is added to Sample Basic. It calculates a correlation coefficient using cross-dimensional members in the data sets.

```plaintext
FIX(Product)
"Sales Correl" = @CORRELATION(SKIPNONE, @XRANGE("2011"->"Sep", "2012"->"Mar"), @XRANGE("2012"->"Sep", "2013"->"Mar"));
ENDFIX
```

The correlation above is calculated across the following two multidimensional ranges specified by `XrangeList1` and `XrangeList2`:

- 2011->Sep
- 2011->Oct
- 2011->Nov
- 2011->Dec
- 2012->Jan
- 2012->Feb
- 2012->Mar

- 2012->Sep
- 2012->Oct
- 2012->Nov
- 2012->Dec
- 2013->Jan
- 2013->Feb
- 2013->Mar

**See Also**

- `@RANGE`  

**@COUNT**

Returns the number of data values in the specified data set (`XrangeList`).

**Syntax**

```plaintext
@COUNT (SKIPNONE | SKIPMISSING | SKIPZERO | SKIPBOTH, XrangeList)
```

**Parameters**

- **SKIPNONE**
  Includes all cells specified in the data set, regardless of their content, during calculation of the count.

- **SKIPMISSING**
  Excludes all `#MISSING` values from the data set during calculation of the count.
**SKIPZERO**
Excludes all zero (0) values from the data set during calculation of the count.

**SKIPBOTH**
Excludes all zero (0) values and #MISSING values from the data set during calculation of the count.

**XrangeList**
A list of numeric values. Referred to generically throughout this topic as "the data set." Can be a valid member name, a comma-delimited list of member names, cross dimensional members, or a return value from a member set function or range function (including @XRANGE).
For more information about XrangeList, see Range List Parameters in the topic Range and Financial Functions.

**Notes**
This function always returns an integer greater than or equal to 0.

**Example**
The following example is based on the Sample Basic database. Assume that the Measures dimension contains an additional member, Prod Count. This example calculates the count of all products for which a data value exists and uses the @RANGE function to generate expList:

```
FIX(Product)
"Prod Count" = @COUNT(SKIPMISSING,@RANGE(Sales,@CHILDREN(Product)));
ENDFIX
```

This example produces the following report. Since SKIPMISSING is specified in the calculation script, the #MI values for Diet Drinks are skipped during the product count.

<table>
<thead>
<tr>
<th>Jan Actual</th>
<th>New York Budget</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sales</td>
<td></td>
</tr>
<tr>
<td>Colas</td>
<td>678</td>
</tr>
<tr>
<td>Root Beer</td>
<td>551</td>
</tr>
<tr>
<td>Cream Soda</td>
<td>663</td>
</tr>
<tr>
<td>Fruit Soda</td>
<td>587</td>
</tr>
<tr>
<td>Diet Drinks</td>
<td>#MI</td>
</tr>
<tr>
<td>Product</td>
<td>2479</td>
</tr>
</tbody>
</table>

The following example assumes a Year dimension is added to Sample Basic. It counts data values using cross-dimensional members in the data set.

```
FIX(Product)
"Count" = @COUNT(SKIPMISSING,@XRANGE("2011"->"Sep", "2012"->"Mar"));
ENDFIX
```
The above calculation is performed across the following multidimensional range specified by XrangeList:

2011->Sep
2011->Oct
2011->Nov
2011->Dec
2012->Jan
2012->Feb
2012->Mar

See Also

• @ISRANGENONEMPTY
• @RANGE

@CREATEBLOCK

Creates a block or blocks for a sparse member name or a sparse member combination, and sets dense values in the newly created block to #MISSING.

Sometimes, new blocks are not desired; for example, when they contain no other values. In large databases, creation and processing of unneeded blocks can increase processing time and storage requirements.

This advanced-level function can help you use bottom-up calculation to achieve faster performance. It is useful for generating empty target blocks that can then be traversed during bottom-up processing, and populated with data at that time. It is most useful in those situations where blocks are not automatically created by the calculator; for example, during processing of a dense formula where the target blocks are from a different, sparse dimension.

Whereas the allocation functions (@ALLOCATE and @MDALLOCATE) also create the necessary target blocks, those functions are intended specifically for allocating values. The purpose of @CREATEBLOCK is only to enable rapid block creation, without reading or writing data.

Note:

• This function is not supported in outline member formulas.
• The DATACOPY calculation command also creates blocks on demand.

Syntax

@CREATEBLOCK(mbrName|mbrList)
Parameters

**mbrName**
Any single, sparse member name or a sparse member combination or a function that returns a single member, member list, or member combination. For example:

- Single member name: ["200-20"]
- Combination of sparse members: ["100-10"->"New York"]
- Member function returning *mbrName* or *mbrList*: @ANCESTORS("New York")

Notes

- This function does nothing if the block for the specified member combination already exists.
- *mbrName|mbrList* can be explicitly stated or can be returned by a function.
- If *mbrName* is a cross-dimensional member (such as "100-10"->"New York"), this function creates a block for the combination specified.
- When you use this function in a calculation script, use it within a FIX statement; for example, FIX on the member for which blocks should be created. Although FIX is not required, using it may improve calculation performance.
- If you use this function in a member formula, your formula should look like this: @CREATEBLOCK (...).
- This function does not return a value; rather, it creates the required blocks in the database with a #MISSING value.
- On sparse dimension members, a formula is executed in top-down mode, creating all possible blocks. However, if the dimension member is dense, it is executed as bottom-up, creating new blocks only based on the existing ones. Therefore, @CREATEBLOCK will not create dense blocks on an empty database.
- For more discussion of top-down and bottom-up processing, see @CALCMODE.

Example

The following calculation script example uses the Sample.Basic database, but assumes that only the 100-10 and New York block is loaded. The member formula for Sales is @CREATEBLOCK("100").

```sql
/* Calling @CREATEBLOCK inside member formula (Sales) */
FIX("100-10", "New York")
   "Sales" (
      @CREATEBLOCK ("100");
   )
ENDFIX
```

The script creates all possible sparse blocks matching the FIX...ENDFIX statement. In this case, only the block "100"->"New York" is created.
In the following calculation script example, @CREATEBLOCK is not used in any member formula, so it must be assigned in the script using mbrName =.

/* Calling @CREATEBLOCK outside member formula */
Budget = @CREATEBLOCK (*100*);

The existing value for Budget member in the current processing block is unchanged, because @CREATEBLOCK does not return a value (see first Note).

@CURGEN

Returns the generation number of the current member combination for the specified dimension. This number represents the number of members separating the current member from the top-most member of the dimension.

Syntax

@CURGEN (dimName)

Parameters

dimName
Single dimension name specification. dimName must be the name of the top-most member of the dimension. It cannot be another member name from within the dimension.

Example

Given the following database structure:

Year
  Qtr1
    Jan, Feb, Mar
  Qtr2
    Apr, May, Jun
  Qtr3
    Jul, Aug, Sep
  Qtr4
    Oct, Nov, Dec

@CURGEN provides the following results for the members shown:

<table>
<thead>
<tr>
<th>Formula</th>
<th>Current Member</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Position = @CURGEN(Year);</td>
<td>Year</td>
<td>1</td>
</tr>
<tr>
<td>Position = @CURGEN(Year);</td>
<td>Qtr2</td>
<td>2</td>
</tr>
<tr>
<td>Position = @CURGEN(Year);</td>
<td>Oct</td>
<td>3</td>
</tr>
</tbody>
</table>

See Also

- @CURLEV
- @GEN
@CURLEV

Returns the level number of the current member combination for the specified dimension. This number represents the number of members that separates the current member from its bottom-most descendant.

Syntax

@CURLEV(dimName)

Parameters

dimName
Single dimension name specification. *dimName* must be the name of the top-most member of the dimension. It cannot be another member name from within the dimension.

Example

Given the following database structure:

```
Year
  Qtr1
    Jan, Feb, Mar
  Qtr2
    Apr, May, Jun
  Qtr3
    Jul, Aug, Sep
  Qtr4
    Oct, Nov, Dec
```

@CURLEV provides the following results for the members shown:

<table>
<thead>
<tr>
<th>Formula</th>
<th>Current Member</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Position = @CURLEV(Year);</td>
<td>Year</td>
<td>2</td>
</tr>
<tr>
<td>Position = @CURLEV(Year);</td>
<td>Qtr3</td>
<td>1</td>
</tr>
<tr>
<td>Position = @CURLEV(Year);</td>
<td>Aug</td>
<td>0</td>
</tr>
</tbody>
</table>

See Also

- @CURGEN
- @LEV

@CURRMBR

Returns the member that is currently being calculated in the specified dimension (*dimName*). This function can be used as a parameter of another function, where that parameter is a single member or a list of members.
Syntax

@CURRMBR (dimName)

Parameters

dimName
A single dimension name.

Notes

• You cannot use this function in a FIX statement.
• You cannot use this function on the left-hand side of a formula.
• The time required for retrieval and calculation may be significantly longer if this function is in a formula attached to a member tagged as Dynamic Calc or Dynamic Calc and Store.

Caution:

If you use this function to return a member name which is then concatenated with other names to get a final member name, it may result in an invalid member name, depending on the current intersection being calculated. For example: @MEMBER(@CONCATENATE(@NAME (@CURRMBR (*Account"))),"_Total")

Example

In the Sample Basic database,

@CURRMBR(Year);

returns Jan if the current member of Year being calculated is Jan.

As a more complex example, consider the following formula in the context of the Sample Basic database. Assume that the Measures dimension contains an additional member, Average Sales.

"Average Sales"
(IF(@ISLEV(Product,0))
Sales;
ELSE
@AVGRANGE(SKIPNONE,Sales,@CHILDREN(@CURRMBR(Product)));
ENDIF;);

This formula populates each upper-level member of the Product dimension (100, 200) at Average Sales. To calculate Average Sales, the Sales values for the level 0 members of Product are averaged and placed in their respective parent members. The Average Sales values for the level 0 Product members are the same as the Sales values, as specified by the IF statement in the calculation script.
This example produces the following report:

<table>
<thead>
<tr>
<th></th>
<th>New York</th>
<th>Actual</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sales</td>
<td>Average Sales</td>
<td></td>
</tr>
<tr>
<td>======</td>
<td>=============</td>
<td></td>
</tr>
<tr>
<td>100-10</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>100-20</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>100-30</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>100</td>
<td>30</td>
<td>10</td>
</tr>
<tr>
<td>200-10</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>200-20</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>200-30</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>200-40</td>
<td>35</td>
<td>35</td>
</tr>
<tr>
<td>200</td>
<td>110</td>
<td>27.5</td>
</tr>
<tr>
<td>300</td>
<td>#MI</td>
<td>#MI</td>
</tr>
<tr>
<td>400</td>
<td>#MI</td>
<td>#MI</td>
</tr>
<tr>
<td>Diet</td>
<td>35</td>
<td>11.67</td>
</tr>
<tr>
<td>Product</td>
<td>140</td>
<td>35</td>
</tr>
</tbody>
</table>

See Also

@CURRMBRRANGE

@CURRMBRRANGE

Generates a member list that is based on the relative position of the current member being calculated.

Syntax

@CURRMBRRANGE (dimName, (GEN|LEV), genLevNum, [startOffset], [endOffset])

Parameters

dimName
Name of the dimension for which you want to return the range list.

GEN|LEV
Defines whether the range list to be returned is based on a generation or a level within the dimension.

genLevNum
Integer value that defines the absolute generation or level number of the range list to be returned.

startOffset
Defines the first member in the range to be returned.

• A null value returns the first member of the specified genLevNum.

• An integer value returns the member name relative to the current member being calculated.
• A negative value specifies a member prior to the current member being calculated in the dimension.
• A value of 0 returns the name of the member currently being calculated.
• A positive value specifies a member after the current member being calculated in the dimension.

endOffset
Defines the last member in the range to be returned.
• A null value returns the last member of the specified genLevNum.
• An integer value returns the member name relative to the current member being calculated.
• A negative value specifies a member prior to the current member being calculated in the dimension.
• A value of 0 returns the name of the member currently being calculated.
• A positive value specifies a member after the current member being calculated in the dimension.

Notes
• You cannot use this function in a FIX statement.
• The first three parameters of this function (dimName,{GEN|LEV},genLevNum) provide a member range list. The startOffset and endOffset parameters create a subset of this list. For example, consider the following syntax in the context of the Sample Basic database:

@CURRMBRRANGE(Year,LEV,0,-1,1)

In this example, the full range list contains the level 0 members of the Year dimension (Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec). If the current member being calculated in the Year dimension is Jan, the startOffset and endOffset parameters reduce this list to (Jan, Feb). Since there is no member prior to Jan in the full range list, only two members are returned: Jan itself and the member after it, Feb. If the current member being calculated is Feb, the subset list would include three members: Jan, Feb, Mar.
• Currently, this function can be used only within range and financial functions, such as @AVGRANGE, @MAXRANGE, @COMPOUND, and @SHIFT.

Example

Example 1
Average Inventory is calculated by summing opening inventories from the first month of the year to the current period plus one period, and dividing the result by the number of periods to date plus one period. This calculation is accomplished by defining the @CURRMBRRANGE function within the rangeList parameter of the @AVGRANGE function.

"Average Inventory" = @AVGRANGE(SKIPNONE,"Opening Inventory", @CURRMBRRANGE(Year, LEV, 0, , 1));
This example produces the following result:

<table>
<thead>
<tr>
<th>Jan</th>
<th>Feb</th>
<th>Mar</th>
<th>Apr</th>
<th>Nov</th>
<th>Dec</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>110</td>
<td>120</td>
<td>130</td>
<td>. . .</td>
<td>200</td>
</tr>
<tr>
<td>105</td>
<td>110</td>
<td>115</td>
<td>120</td>
<td>. . .</td>
<td>155</td>
</tr>
</tbody>
</table>

Since a null value is specified for `startOffset`, the average operations always begin at the first member of the range list, Jan. The `endOffset` parameter, 1, specifies that the member after the current member being calculated is included in each average operation. So, for `Average Inventory->Jan`, the values for Jan and Feb are averaged; for `<Average Inventory->Feb`, the values for Jan, Feb, and Mar are averaged; and so on. The values for Nov and Dec are the same since there is no member after Dec in the range list.

**Example 2**

Inventory Turnover is calculated by summing period-to-date Sales and dividing the result by the Average Inventory.

\[
\text{Turnover} = \frac{\text{SUMRANGE}(Sales, \text{CURRMBRANGE}(Year, LEV, 0, , 0))}{\text{Average Inventory}}
\]

which produces the following result:

<table>
<thead>
<tr>
<th>Jan</th>
<th>Feb</th>
<th>Mar</th>
<th>Apr</th>
</tr>
</thead>
<tbody>
<tr>
<td>110</td>
<td>116.7</td>
<td>122.5</td>
<td>126</td>
</tr>
<tr>
<td>40</td>
<td>44</td>
<td>48</td>
<td>52</td>
</tr>
<tr>
<td>0.36</td>
<td>0.72</td>
<td>1.08</td>
<td>1.46</td>
</tr>
</tbody>
</table>

**Example 3**

Consider the following formula:

\[
\text{@CURRMBRANGE}(Year, LEV, \text{@CURLEVEL}("Year"), -1, 1)
\]

The full range list contains the members of the Year dimension at a particular level. The level is determined by taking the level of the current member being calculated. For example, if the current member being calculated is Jan, the full range list contains all level 0 members of Year dimension (Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec). The `startOffset` and `endOffset` parameters reduce this list to (Jan, Feb). As there is no member prior to Jan in the full range list, only two members are returned: Jan and Feb. If the current member being calculated is Feb, the subset list includes three members: Jan, Feb, Mar.

**Note:**

The usage demonstrated by this example would require `RTDEPCALCOPTIMIZE` to be set to FALSE.
@DATEDIFF

Returns the difference (number) between two input dates in terms of the specified date-parts, following a standard Gregorian calendar.

Syntax

@DATEDIFF ( date1, date2, date_part )

Parameters

date1
A number representing the input date between January 1, 1970 and Dec 31, 2037. The number is the number of seconds elapsed since midnight, January 1, 1970. To retrieve this number, use any of the following functions: @TODAY, @TODATEEX, @DATEROLL.
Date-time attribute properties of a member can also be used to retrieve this number. For example, @AttributeVal("Intro Date"); returns the product introduction date for the current product in context.

date2
A second input date. See date1.

date_part
Defined using the following rule:

date_part_ex ::= DP_YEAR | DP_QUARTER | DP_MONTH | DP_WEEK | DP_DAY | DP_DAYOFYEAR | DP_WEEKDAY

Defined time components as per the standard calendar:
• DP_YEAR - Year of the input date.
• DP_QUARTER - Quarter of the input date.
• DP_MONTH - Month of the input date.
• DP_WEEK - Week of the input date.
• DP_DAY - Day of the input date.

Notes

Based on the input date_part, the difference between the two input dates is counted in terms of time component specified.

Example: For input dates June 14, 2005 and Oct 10, 2006,
• DP_YEAR returns the difference in the year component. (2006 - 2005 = 1)
• DP_QUARTER returns the distance between the quarters capturing the input dates. (Quarter 4, 2006 - Quarter 2, 2005 = 6)
• DP_MONTH returns the distance between the months capturing the input dates. (Oct 2006 - June 2005 = 16)
• DP_WEEK returns the distance between the weeks capturing the input dates. Each Standard calendar week is defined to start on Sunday and it spans 7 days. (Oct 10, 2006 - June 14, 2005 = 69)
• DP_DAY returns the difference between the input dates in terms of days. (483 days)

Example
Assume the outline has two date type members, MyDate1 and MyDate2.

Profit=@DateDiff(MyDate1, MyDate2, DP_WEEK);
Profit=@DatePart(MyDate1, DP_YEAR);
MyDate2=@DateRoll(MyDate1, DP_MONTH), 10);

See Also
• @ATTRIBUTEVAL
• @DATEPART
• @DATEROLL
• @FORMATDATE
• @TODATEEX
• @TODAY

@DATEPART
This function returns the Year/Quarter/Month/Week/Day/DayOfYear/Weekday as a number, given the input date and a date part, following the standard Gregorian calendar.

Syntax

@DATEPART ( date, date_part_ex )

Parameters

date
A number representing the input date between January 1, 1970 and Dec 31, 2037. The number is the number of seconds elapsed since midnight, January 1, 1970. To retrieve this number, use any of the following functions: @TODAY, @TODATEEX, @DATEROLL.
Date-time attribute properties of a member can also be used to retrieve this number. For example, @AttributeVal("Intro Date"); returns the product introduction date for the current product in context.
**date_part_ex**
Defined using the following rule:

```
date_part_ex ::= DP_YEAR | DP_QUARTER | DP_MONTH | DP_WEEK | DP_DAY |
                 DP_DAYOFYEAR | DP_WEEKDAY
```

Defined time components as per the standard calendar:
- **DP_YEAR** - Year of the input date.
- **DP_QUARTER** - Quarter of the input date.
- **DP_MONTH** - Month of the input date.
- **DP_WEEK** - Week of the input date.
- **DP_DAY** - Day of the input date.

**Notes**
Based on the requested time component, the output is as follows:
- **DP_YEAR** returns the year of the input date in yyyy format.
- **DP_QUARTER** returns the quarter of the year (1 to 4) for the input date.
- **DP_MONTH** returns the month of the year (1 to 12) for the input date.
- **DP_WEEK** returns the week of the year for the input date (1 to 52).
- **DP_WEEKDAY** returns the week day of the input date. (1 - Sunday, 2 - Monday, ... 6 - Saturday).
- **DP_DAYOFYEAR** returns the day of the year numbering (1 to 366).
- **DP_DAY** returns the day of the month (1 to 31).

**Example:** For June 14, 2005,

- **DP_YEAR** returns 2005 (the year member, in yyyy format).
- **DP_QUARTER** returns 2 (Second quarter of the year)
- **DP_MONTH** returns 6 (Sixth month of the year)
- **DP_WEEK** returns 24 (24th week of the year)
- **DP_WEEKDAY** returns 4 (for Wednesday. Sunday = 1)
- **DP_DAYOFYEAR** returns 165 (165th day of the year)
- **DP_DAY** returns 14 (14th day of the month)

**Example**
Assume the outline has two date type members, MyDate1 and MyDate2.

```
Profit=@DateDiff(MyDate1, MyDate2, DP_WEEK);
Profit=@DatePart(MyDate1, DP_YEAR);
MyDate2=@DateRoll(MyDate1, DP_MONTH), 10);
```
See Also

- @ATTRIBUTEVAL
- @DATEDIFF
- @DATEROLL
- @FORMATDATE
- @TODATEEX
- @TODAY

@DATEROLL

To the given date, rolls (adds or subtracts) a number of specific time intervals, returning another date. This function assumes a standard Gregorian calendar.

Syntax

@DATEROLL ( date, date_part, number )

Parameters

date
A number representing the date between January 1, 1970 and Dec 31, 2037. The number is the number of seconds elapsed since midnight, January 1, 1970. To retrieve this number, use either of the following functions: @TODAY, @TODATEEX. Date-time attribute properties of a member can also be used to retrieve this number. For example, @AttributeVal("Intro Date"); returns the product introduction date for the current product in context.

date_part
Defined using the following rule:

date_part_ex ::= DP_YEAR | DP_QUARTER | DP_MONTH | DP_WEEK | DP_DAY |
               DP_DAYOFYEAR | DP_WEEKDAY

Defined time components as per the standard calendar:

- DP_YEAR - Year of the input date.
- DP_QUARTER - Quarter of the input date.
- DP_MONTH - Month of the input date.
- DP_WEEK - Week of the input date.
- DP_DAY - Day of the input date.

number
Number of time intervals to add or subtract.

Notes

Based on input date_part and dateroll number, the date is moved forward or backward in time.
Example: For input date June 14, 2005 and input dateroll number 5,

- DP_YEAR adds 5 years to the input date. (June 14, 2010)
- DP_QUARTER adds 5 quarters to the input date. (June 14, 2005 + 5 quarters = June 14, 2005 + 15 months = Sept 14, 2006)
- DP_MONTH adds 5 months to the input date (June 14, 2005 + 5 months = Nov 14, 2005)
- DP_WEEK adds 5 weeks to the input date (June 14, 2005 + 5 weeks = June 14, 2005 + 35 days = July 19, 2005)
- DP_DAY adds 5 days to the input date. (June 14, 2005 + 5 days = June 19, 2005)

Example

Assume the outline has two date type members, MyDate1 and MyDate2.

Profit=@DateDiff(MyDate1, MyDate2, DP_WEEK);
Profit=@DatePart(MyDate1, DP_YEAR);
MyDate2=@DateRoll(MyDate1, DP_MONTH, 10);

See Also

- @ATTRIBUTEVAL
- @DATEDIFF
- @DATEPART
- @FORMATDATE
- @TODATEEX
- @TODAY

@DECLINE

Calculates the depreciation of an asset for the specified period using the declining balance method. The factor by which the declining balance depreciates the assets is specified using factorMbrConst. For example, to calculate a double declining balance, set factorMbrConst to 2.

Syntax

@DECLINE (costMbr, salvageMbrConst, lifeMbrConst, factorMbrConst [, XrangeList])

Parameters

costMbr
Single member specification representing the starting values of the assets. More than one asset can be input and depreciated across the specified range. The function calculates each asset separately.
**salvageMbrConst**
Single member specification, variable name, or numeric expression that provides a constant value. This value represents the value of the asset at the end of the depreciation.

**lifeMbrConst**
Single member specification, variable name, or numeric expression that provides a constant value. The value represents the number of periods over which the asset is depreciated.

**factorMbrConst**
Single member specification, variable name, or numeric expression that provides a constant value. The value represents the factor by which the asset is depreciated.

**XrangeList**
Optional parameter specifying the periods over which the function is calculated. More than one asset can be depreciated. If a range is not specified, Essbase uses the level 0 members from the dimension tagged as Time.
Can be a valid member name, a comma-delimited list of member names, cross dimensional members, or a return value from a member set function or range function (including @XRANGE).
For more information about `rangeList` and `XrangeList`, see [Range List Parameters](#) in the topic **Range and Financial Functions**.

**Notes**
Financial functions never return a value; rather, they calculate a series of values internally based on the range specified.

**Example**
The following example calculates the depreciation of Asset for the specified series of fiscal years.

```
"Decline Dep" = @DECLINE(Asset,Residual,Life,
2,FY2000:FY2001,FY2002,FY2003);
```

This example produces the following report:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Asset</td>
<td>9,000</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Residual</td>
<td>750</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Life</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Decline Dep</td>
<td>3,600</td>
<td>2,160</td>
<td>1,296</td>
<td>778</td>
</tr>
</tbody>
</table>

The following example assumes a Year dimension is added to Sample Basic. It calculates depreciation using a multidimensional range.

```
FIX ("100-10", "New York")
"Decline Dep" = @DECLINE(Asset,Residual,Life,2,@XRANGE("2011"->"Sep",
"2012"->"Mar"));
ENDFIX
```
The above calculation is performed across the following multidimensional range specified by *XrangeList*:

- 2011->Sep
- 2011->Oct
- 2011->Nov
- 2011->Dec
- 2012->Jan
- 2012->Feb
- 2012->Mar

See Also

- @GROWTH
- @SLN

@DESCENDANTS

Returns all descendants of the specified member, or those down to the specified generation or level. This function excludes the specified member.

Syntax

```
@DESCENDANTS (mbrName [, genLevNum | genLevName])
```

Parameters

- **mbrName**
  Any valid single member name, or a function that returns a single member.

- **genLevNum**
  Optional. An integer value that defines the absolute generation or level number down to which to select the members. A positive integer defines a generation number. A value of 0 or a negative integer defines a level number.

- **genLevName**
  Optional. Level name or generation name down to which to select the members.

Notes

- You can use this function as a parameter of another function, where that parameter is a list of members.

- Essbase sorts the generated list of members starting with the nearest descendant of the member, followed by the next nearest descendant of the member, and so on. In the Sample.Basic database, if you specify `@DESCENDANTS(100)`, Essbase returns 100-10, 100-20, 100-30 (in that order). This order is important to consider when you use this function with certain forecasting and statistical functions.

- To include the specified member, use `@IDESCENDANTS`.

- To include descendants of shared members, use `@RDESCENDANTS` and `@IRDESCENDANTS`. 
Example

In the Sample Basic database:

@DESCENDANTS(East)

@DESCENDANTS(Profit)
returns Margin, Sales, COGS, Total Expenses, Marketing, Payroll, and Misc (in that order).

@DESCENDANTS(Market,2)
returns East, West, South, and Central (in that order).

@DESCENDANTS(Diet,0)
returns 100-20, 200-20, and 300-30 (in that order).

See Also

• @ANCESTORS
• @CHILDREN
• @IDESCENDANTS
• @ILDESCENDANTS
• @IRDESCENDANTS
• @ISDESC
• @LDESCENDANTS
• @RDESCENDANTS
• @SIBLINGS

@DISCOUNT

Calculates a value discounted by the specified rate, from the first period of the range to the period in which the amount to discount is found. The answer is returned in the same period. More than one value can be discounted simultaneously in this manner.

Syntax

@DISCOUNT (cashMbr, rateMbrConst [, XrangeList])

Parameters

cashMbr
Member specification representing the value you want to discount from the last period in XrangeList to the current period.
rateMbrConst
Member specification, variable name, or numeric expression which provides a constant value. The value represents the rate per period which cashMbr is discounted. It is a decimal value, not a percent.

XrangeList
Optional parameter specifying the period over which the discount is calculated. If a range is not specified, Essbase uses the level 0 members from the dimension tagged as Time.
Can be a valid member name, a comma-delimited list of member names, cross dimensional members, or a return value from a member set function or range function (including @XRANGE).
For more information about rangeList and XrangeList, see Range List Parameters in the topic Range and Financial Functions.

Notes
Financial functions never return a value; rather, they calculate a series of values internally based on the range specified.

Example
The following example discounts the values in Cash by the rates in Credit Rate and places the results in Discount Amount for each fiscal year.

"Discount Amount" = @DISCOUNT(Cash,"Credit Rate",FY1999:FY2002,FY2003);

This example produces the following report:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Cash</td>
<td>0.00</td>
<td>0.00</td>
<td>1000.00</td>
<td>1000.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Credit Rate</td>
<td>0.00</td>
<td>0.00</td>
<td>0.05</td>
<td>0.05</td>
<td>0.00</td>
</tr>
<tr>
<td>Discount Amount</td>
<td>#MI</td>
<td>#MI</td>
<td>863.84</td>
<td>822.70</td>
<td>#MI</td>
</tr>
</tbody>
</table>

The following example assumes a Year dimension is added to Sample Basic. It calculates discount using a multidimensional range.

FIX ("100-10", "New York")
"Discount Amount" = @DISCOUNT(Cash,"Credit Rate",@XRANGE("2011"->"Sep","2012"->"Mar"));
ENDFIX

The above calculation is performed across the following multidimensional range specified by XrangeList:

2011->Sep
2011->Oct
2011->Nov
2011->Dec
2012->Jan
@ENUMVALUE

Returns the internal numeric value for a text value in a text list.

Syntax

@ENUMVALUE (enum_string)

Parameters

denum_string
String of the format text_list_name.char_string_literal, where:

- text_list_name is the name of a text list, or of a member that is associated with a text list.
- char_string_literal is one of the text values represented in the text list.

Example

The following example is based on a variation of ASOSamp.Sample. Assume there is a text list named CustSatRatings, in which text values are mapped to numeric IDs as follows: Good=1, Average=2, Poor=3.

@ENUMVALUE(CustSatRatings, "Good");

returns 1.

@EQUAL

Returns a member set of member names that match the specified token name.

This function can be used on unique and duplicate-name outlines.

Syntax

@EQUAL (tokenName, topMbrinHierarchy)

Parameters

tokenName
Token string value, representing the name of a member, with which to compare to members in the outline, starting with member specified in topMbrinHierarchy. The specified token name must not be qualified for duplicate members.

topMbrinHierarchy
A fully qualified name of a member in the outline on which to base the member search. The specified member and its aliases, and all of its descendants, are included in the search.
To search the entire outline, provide an empty string (""") for this parameter. For example, @EQUAL("100-10", "").

Example

The following examples are based on the following duplicate-name outline:

```
Product
  100
     100-10
     100-10-10
     100-20
     100-30
     200
     200-10
     200-20
     200-30
  300
     300-10
     300-20
Diet
  100-10
     100-10-11
     200-10
     300-10
Bottle
  200-10
  300-20
```

@EQUAL("100-10", "Product")

Returns the members [Diet].[100-10] and [100].[100-10].

@EQUAL("100-10", "Diet")

Returns the member [Diet].[100-10].

See Also

- @EXPAND
- @LIKE
- @MBRCOMPARE
- @MBRPARENT
- @NOTEQUAL

@EXP

Returns the exponent of a specified expression; that is, the value of e (the base of natural logarithms) raised to the power of the specified expression.
Syntax

@EXP (expression)

Parameters

expression
Single member specification, variable name, function, or other numeric expression. If
less than -700 or greater than 700, Essbase returns #MISSING.

Example

The following example is based on a variation of Sample Basic:

Index = @EXP("Variance %"/100);

This example produces the following result:

<table>
<thead>
<tr>
<th></th>
<th>East</th>
<th>West</th>
<th>South</th>
<th>Central</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variance %</td>
<td>10.7</td>
<td>10.9</td>
<td>3.6</td>
<td>3.6</td>
</tr>
<tr>
<td>Index</td>
<td>1.11293</td>
<td>1.11516</td>
<td>1.03666</td>
<td>1.03666</td>
</tr>
</tbody>
</table>

See Also

@LN

@EXPAND

Expands a member search by calling a member set function for each member in a
member list. The members returned by this function are added to the existing member
set. Duplicate members are not removed from the member set.

This function can be used on unique and duplicate-name outlines.

Syntax

@EXPAND (mbrSetFunction, mbrList[, genLevNum][, LAYERONLY | ALL][, topMbrinHierarchy])

Parameters

mbrSetFunction
One of the following member set functions, which return a list of members:

- @ANCESTORS
- @IANCESTORS
- @CHILDREN
- @ICHOILDREN
- @DESCENDANTS
• @IDESCENDANTS
• @EQUAL
• @MBRPARENT
• @SIBLINGS
• @ISIBLINGS

mbrList
A comma-delimited list of members grouped together using @LIST or a member set function (such as @DESCENDANTS) that returns a list of members.

genLevNum
Optional: This argument applies only if you specify @ANCESTORS, @IANCESTORS, @DESCENDANTS, or @IDESCENDANTS for mbrSetFunction. The integer value that defines the absolute generation or level number up to which to select members. A positive integer defines a generation number. A value of 0 or a negative integer defines a level number.

LAYERONLY
Optional: This argument applies only if you specify @ANCESTORS, @IANCESTORS, @DESCENDANTS, or @IDESCENDANTS for mbrSetFunction. Returns only those members at the specified generation or level (genLevNum) that match the selection criteria. If you specify this argument, you must specify genLevNum.

ALL
Optional: This argument applies only if you specify @ANCESTORS, @IANCESTORS, @DESCENDANTS, or @IDESCENDANTS for mbrSetFunction. Returns all of the members that match the member selection criteria, starting with the specified top member (topMbrinHierarchy). If you specify this argument, you must specify topMbrinHierarchy.

topMbrinHierarchy
Optional: This argument applies only if you specify @EQUAL for mbrSetFunction. A fully qualified member name on which to base the member search. The specified member and its aliases, and all of its descendants, are included in the search. If you specify @EQUAL for mbrSetFunction, and you do not specify topMbrinHierarchy, Essbase searches the entire outline.

Example
The following examples are based on the following duplicate-name outline:

```
Product
  100
    100-10
      100-10-10
    100-20
    100-30
  200
    200-10
    200-20
    200-30
  300
    300-10
```
@EXPAND("@DESC", @LIST("Product"), -1, LAYERONLY)

Returns all of the members under the Product dimension that are at level 1, which are [100].[100-10], [Product].[200], [Product].[300], [Diet].[100-10], and [Product].[Bottle].

@EXPAND("@EQUAL", @EXPAND("@CHILDREN", @LIST("[product].[100]", "[product].[200]")), , ,"Product")

Essbase first executes the inner @EXPAND function—@EXPAND("@CHILDREN", @LIST("[product].[100]", "[product].[200]"))—which expands the member list to include all of the children of members 100 and 200 (a total of six members). Then Essbase executes the outer @EXPAND function, which searches the Product hierarchy for a match with any of the six members.

See Also

• @BETWEEN
• @EQUAL
• @NOTEQUAL
• @LIKE
• @MBRCOMPARE
• @MBRPARENT

@FACTORIAL

Returns the factorial of expression. The factorial of a number is equal to 1*2*3*...*number.

Syntax

@FACTORIAL (expression)
Parameters

expression
Single member specification or numeric expression.

Notes
- **expression** can be no larger than 189. If **expression** is larger than 189, Essbase returns #MISSING.
- If **expression** is negative, Essbase returns #MISSING.

Example

```plaintext
@FACTORIAL(1)     1
@FACTORIAL(5)     120
```

See Also

@POWER

@FORMATDATE

Returns a formatted date-string.

Syntax

```plaintext
@FormatDate(date, date_format_string)
```

Parameters

<date>
A number representing the input date between January 1, 1970 and Dec 31, 2037. The number is the number of seconds elapsed since midnight, January 1, 1970. To retrieve this number, use any of the following functions: @TODAY, @TODATEEX, or @DATEROLL.

Date-time attribute properties of a member can also be used to retrieve this number. For example, @AttributeVal("Intro Date"); returns the product introduction date for the current product in context.

date_format_string
One of the following literal strings (excluding ordered-list numbers and parenthetical examples) indicating a supported date format.

1. "mon dd yyyy" (Example: mon = Aug)
2. "Month dd yyyy" (Example: Month = August)
3. "mm/dd/yy"
4. "mm/dd/yyyy"
5. "yy.mm.dd"
6. "dd/mm/yy"
7. "dd.mm.yy"
8. "dd-mm-yy"
9. "dd Month yy"
10. "dd mon yy"
11. "Month dd, yy"
12. "mon dd, yy"
13. "mm-dd-yy"
14. "yy/mm/dd"
15. "yymmdd"
16. "dd Month yyyy"
17. "dd mon yyyy"
18. "yyyy-mm-dd"
19. "yyyy/mm/dd"

20. Long format (Example: WeekDay, Mon dd, yyyy)
21. Short format (Example: m/d/yy)

Notes

• Using an invalid input date returns an error.
• Using extra whitespace not included in the internal format strings returns an error.
• This function interprets years in the range 1970 to 2029 for yy format. Therefore, if the function is invoked using a date format mm/dd/yy for June 20, 2006, the returned date string is "06/20/06".

Example

Assume the outline has a date type member MyDate1.

Profit (If(@ToDateEx("yyyy-mm-dd", @FormatDate(@Today(), "yyyy-mm-dd") == MyDate1 )
    Profit=99;
Endif;)

See Also

• @DATEDIFF
• @DATEPART
• @DATEROLL
• @TODATEEX
• @TODAY
@GEN

Returns the generation number of the specified member.

Syntax

@GEN (mbrName)

Parameters

mbrName
Any valid single member name, or a function that returns a single member.

Example

In the Sample Basic database:

@GEN(Year)

Returns 1.

@GEN(Qtr3)

Returns 2.

See Also

• @CURGEN
• @LEV

@GENMBRS

Returns all members with the specified generation number or generation name in the specified dimension.

Syntax

@GENMBRS (dimName, genName|genNum)

Parameters

dimName
A single dimension name specification.

genName|genNum
Generation name or generation number from dimName. A positive integer defines a generation number.

Notes

• If you specify a name for the genName parameter, Essbase looks for a generation with that name in the specified dimension.
• If you specify a number for the genName parameter (for example, 2), Essbase first looks for a generation with a number string name. If no generation name exists with that numeric name, Essbase checks to see if the parameter is a valid generation number. Check the application event log after running the calculation to make sure that the correct members were calculated.

• Generation 0 is not a valid generation number. Generations begin numbering at 1.

• If you specify a temporary variable for the genName parameter, Essbase does not recognize the value of the variable. It looks in the outline for a generation name with the same name as the temporary variable.

• For more information about generations and defining generation names, see Designing and Maintaining Essbase Cubes.

• Essbase sorts the generated list of members in ascending order. Using Sample Basic as an example, if you specify @GENMBRS(Product,2), Essbase returns 100, 200, 300, 400, Diet (in that order). This order is important to consider when you use the @GENMBRS member set function with certain forecasting and statistical functions.

Example

In the Sample Basic database:

@GENMBRS(Year,Month)
@GENMBRS(Year,3)

both return the following members since generation 3 of the Year dimension is named Month:

Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, and Dec (in that order).

The following example restricts the calculation to members with the combination Budget and one of the members of the Market dimension with a generation name of State.

FIX(Budget,@GENMBRS(Market,State))
CALC DIM (Year,Measures);
ENDFIX

See Also

@LEVMBRS

@GROWTH

Calculates a series of values that represent a linear growth of the first nonzero value encountered in principalMbr across the specified XrangeList. Growth is calculated by multiplying the growth rate in rateMbrConst by the original principalMbr. This value is then added to the previous time period's result, yielding the new value.

Syntax

@GROWTH (principalMbr, rateMbrConst [, XrangeList])
Parameters

principalMbr
Single member specification that represents the initial value of the value to grow. The first nonzero value encountered is the initial value. Other principalMbr values after the first are ignored.

rateMbrConst
Single member specification, variable name, or numeric expression providing a constant value that represents the decimal growth rate to be applied (for example, 10% = .1).

XrangeList
Optional parameter specifying the range over which the function is calculated. If a range is not specified, Essbase uses the level 0 members from the dimension tagged as Time.
Can be a valid member name, a comma-delimited list of member names, cross dimensional members, or a return value from a member set function or range function (including @XRANGE).
For more information about rangeList and XrangeList, see Range List Parameters.

Notes

Financial functions never return a value; rather, they calculate a series of values internally based on the range specified.

Example

The following example calculates the growth of Principal Amount, using the rate found in Growth Rate for each fiscal year. The results are placed in Growth Amount.

"Growth Amount"=@GROWTH("Principal Amount","Growth Rate",FY1998:FY2003);

This example produces the following report:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Principal Amount</td>
<td>1,000</td>
<td>0</td>
<td>2,000</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Growth Amount</td>
<td>1,050</td>
<td>1,120</td>
<td>1,200</td>
<td>1,280</td>
<td>1,380</td>
<td>1,480</td>
</tr>
<tr>
<td>Growth Rate</td>
<td>0.05</td>
<td>0.07</td>
<td>0.08</td>
<td>0.08</td>
<td>0.1</td>
<td>0.1</td>
</tr>
</tbody>
</table>

The following example assumes a Year dimension is added to Sample Basic. It calculates growth using a multidimensional range.

FIX ("100-10", "New York")
"Growth Amount" = @GROWTH("Principal Amount","Growth Rate",@XRANGE("2011"->"Sep", "2012"->"Mar");
ENDFIX
The above calculation is performed across the following multidimensional range specified by XrangeList:

2011->Sep
2011->Oct
2011->Nov
2011->Dec
2012->Jan
2012->Feb
2012->Mar

See Also

- @COMPOUNDGROWTH
- @DECLINE
- @XRANGE

@IALLANCESTORS

Returns the specified member and all the ancestors of that member, including ancestors of any occurrences of the specified member as a shared member. You can use this function as a parameter of another function, where that parameter is a list of members.

Syntax

@IALLANCESTORS (mbrName)

Parameters

mbrName
A valid single member name, or a function that returns a single member.

Notes

Essbase sorts the generated list of members in ascending order of the member number in the outline. Using Sample Basic as an example, if you specify 100-20 for mbrName, Essbase returns 100-20, 100, Diet, Product (in that order). However, the order in which shared ancestors are returned is not guaranteed. This order is important to consider when you use this function with certain forecasting and statistical functions.

Example

The following example is based on the Sample Basic database. Sample Basic has a shared level of diet drinks, which includes 100-20 (Diet Cola). So 100-20 (Diet Cola) is a descendant of 100 (Colas) and is a shared member descendant of Diet:

100
  100-10
  100-20
The following calculation script increases by 5% the Budget Sales values of 100-20 and all its ancestors, including Diet:

```plaintext
FIX(Budget, @IANCESTORS("100-20"))
Sales = Sales * 1.05;
ENDFIX
```

This example produces the following report. This report shows that the Budget->Sales values for 100-20, 100, Diet, and Product (100-20 and its ancestors) have been increased by 5%. The original values were 2610, 8980, 8260, and 28480, respectively.

<table>
<thead>
<tr>
<th>Jan</th>
<th>Actual Sales</th>
<th>Budget Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>Market</td>
<td>4860</td>
<td>5200</td>
</tr>
<tr>
<td>100-10</td>
<td>2372</td>
<td>2740.5 *</td>
</tr>
<tr>
<td>100-20</td>
<td>1082</td>
<td>1170</td>
</tr>
<tr>
<td>100-30</td>
<td>8314</td>
<td>9429 *</td>
</tr>
<tr>
<td>100</td>
<td>2372</td>
<td>2610</td>
</tr>
<tr>
<td>200-20</td>
<td>3122</td>
<td>3090</td>
</tr>
<tr>
<td>300-30</td>
<td>2960</td>
<td>2560</td>
</tr>
<tr>
<td>Diet</td>
<td>8454</td>
<td>8673 *</td>
</tr>
<tr>
<td>Product</td>
<td>31538</td>
<td>30954 *</td>
</tr>
</tbody>
</table>

See Also
- `@ANCESTORS`
- `@IANCESTORS`
- `@ILANCESTORS`
- `@LANCESTORS`

`@IANCESTORS`

Returns the specified member and either all ancestors of the member or all ancestors up to the specified generation or level.

Essbase sorts the generated list of members—starting with the specified member, followed by the nearest ancestor of the member, followed by the next nearest ancestor of the member, and so on. In the Sample.Basic database, if you specify `@IANCESTORS(200-30)`, Essbase returns 200-30, 200, Product (in that order). When using this function with certain forecasting and statistical functions, you must consider order.

You can use this function as a parameter of another function, where the function requires a list of members.
Syntax

@ANCESTORS (mbrName [, genLevNum | genLevName])

Parameters

mbrName
Valid member name, or a function that returns a member.

genLevNum
Optional. The integer value that defines the absolute generation or level number up to which to select members. A positive integer defines a generation number. A value of 0 or a negative integer defines a level number.

genLevName
Optional. The level or generation name up to which to select members.

Example

All examples are from the Sample.Basic database.

@ANCESTORS("New York")
Returns New York, East, Market (in that order).

@ANCESTORS(Qtr4)
Returns Qtr4, Year (in that order).

@ANCESTORS(Sales,-2)
Returns Sales, Margin, Profit (in that order). Members higher than level 2 are not returned.

@ANCESTORS("100-10",1)
Returns 100-10, 100, Product (in that order). All ancestors are returned up to generation 1.

See Also

• @ANCESTORS
• @ALLANCESTORS
• @ILANCESTORS
• @LANCESTORS

@CHILDREN

Returns the specified member and all of its children. This function can be used as a parameter of another function, where that parameter is a list of members.

Syntax

@CHILDREN (mbrName)
Parameters

**mbrName**
Any valid single member name, or a function that returns a single member.

Notes

Essbase sorts the generated list of members starting with the specified member, followed by its children in ascending order. Using Sample Basic as an example, if you specify 100 for *mbrName*, Essbase returns 100, 100-10, 100-20, 100-30 (in that order). This order is important to consider when you use this function with certain forecasting and statistical functions.

Example

In the Sample Basic database:

@ICHILDREN(Market)

Returns Market, East, West, South, and Central (in that order).

@ICHILDREN(Margin)

Returns Margin, Sales, and COGS (in that order).

See Also

@CHILDREN

@IDESCENDANTS

Returns the specified member and either all descendants of the member or all descendants down to the specified generation or level.

Essbase sorts the generated list of members—starting with the specified member, followed by the nearest descendant of the member, followed by the next nearest descendant of the member, and so on. In the Sample.Basic database, if you specify @IDESCENDANTS(100), Essbase returns 100, 100-10, 100-20, 100-30 (in that order). When using this function with certain forecasting and statistical functions, you must consider order.

You can use this function as a parameter of another function, where the function requires a list of members.

Syntax

@IDESCENDANTS (mbrName[, genLevNum | genLevName])

Parameters

**mbrName**
Any valid single member name, or a function that returns a single member.
**genLevNum**
Optional. The integer value that defines the absolute generation or level number up to which to select members. A positive integer defines a generation number. A value of 0 or a negative integer defines a level number.

**genLevName**
Optional. The level or generation name up to which to select members.

**Example**
All examples are from the Sample.Basic database.

@IDESCENDANTS(East)

@IDESCENDANTS(Profit)
Returns Profit, Margin, Sales, COGS, Total Expenses, Marketing, Payroll, and Misc (in that order).

@IDESCENDANTS(Market, 2)
Returns Market, East, West, South, and Central (in that order).

@IDESCENDANTS(South, -1)
Returns South.

**See Also**
- @ANCESTORS
- @CHILDREN
- @DESCENDANTS
- @ILDESCENDANTS
- @IRDESCENDANTS
- @ISDESC
- @LDESCENDANTS
- @RDESCENDANTS
- @SIBLINGS

**@ILANCESTORS**
Returns the members of the specified member list and either all ancestors of the members or all ancestors up to the specified generation or level.

You can use this function as a parameter of another function, where the function requires a list of members.

**Syntax**

@ILANCESTORS ((memberSetFunction) [, genLevNum])
Parameters

**memberSetFunction**

A member set function that returns a list of members. How @ILANCESTORS is used determines which member set functions are allowed. Follow these guidelines:

- If @ILANCESTORS is used alone (not within a FIX statement), you must use the @LIST function and specify member names. For example:

  ```
  @LIST(mbr1, mbr2, ...)
  ```

- If the @ILANCESTORS function is used within a FIX statement, you can use member set functions such as @UDA and @ATTRIBUTE. For example:

  ```
  @UDA(dimName, uda)
  @ATTRIBUTE (attMbrName)
  ```

In this case, you can choose whether to use the @LIST function. For example, both of the following statements are valid, and the statements return the same results.

**Example using only @ATTRIBUTE:**

```
FIX(@ILANCESTORS(@ATTRIBUTE(Caffeinated_True), @ATTRIBUTE(Ounces_12), "200-40"))
... 
ENDFIX;
```

**Example using @LIST and @ATTRIBUTE:**

```
FIX(@ILANCESTORS(@LIST(@ATTRIBUTE(Caffeinated_True), @ATTRIBUTE(Ounces_12), "200-40")))
... 
ENDFIX;
```

⚠️ **Caution:**

All members of the specified member list must be from the same dimension.

**genLevNum**

Optional. The integer value that defines the absolute generation or level number up to which to select members. A positive integer defines a generation number. A value of 0 or a negative integer defines a level number.
Example

All examples are from the Sample.Basic database.

@ILANCESTORS(@LIST("100-10","200-20"))

Returns 100-10 (a specified member); 100 and Product (the ancestors of 100-10); 200-20 (a specified member); and 200 (the ancestor of 200–20). The result does not contain duplicate members.

@ILANCESTORS(@LIST("100","100-10"))

Returns 100 and 100-10 (the specified members); and Product (the ancestor of 100 and 100-10). The result does not contain duplicate members.

@ILANCESTORS(@LIST("100","Product","200"))

Returns 100, Product, and 200 (the specified members). The result does not contain duplicate members.

FIX(@ILANCESTORS(@UDA(Market,"New Market")),2)

... ENDFIX;

Returns Nevada (a member that is assigned the New Market UDA) and West (the ancestor to generation 2 for Nevada); Louisiana (a member that is assigned the New Market UDA) and South (the ancestor to generation 2 for Louisiana); and Colorado (a member that is assigned the New Market UDA) and Central (the ancestor to generation 2 for Colorado).

FIX(@ILANCESTORS(@ATTRIBUTE(Caffeinated_True),@ATTRIBUTE(Ounces_12),"200-40"))

... ENDFIX;

Returns 100-10, 100-20, 200-10, and 300-30 (caffeinated, 12-ounce drinks); and 200-40 (the specified member), and 100, 200, 300, and Product (the ancestors of the members).

See Also

• @ANCESTORS
• @IANCESTORS
• @LANCESTORS

@ILDESCENDANTS

Returns the members of the specified member list and either all descendants of the members or all descendents down to the specified generation or level.
You can use this function as a parameter of another function, where the function requires a list of members.

Syntax

@ILDESCENDANTS ((memberSetFunction) [, genLevNum])

Parameters

memberSetFunction
A member set function that returns a list of members. How this function is used determines which member set functions are allowed. Follow these guidelines:

- If @ILDESCENDANTS is used alone (not within a FIX statement), you must use the @LIST function and specify member names. For example:

  @LIST(mbr1, mbr2, ...)

- If the @ILDESCENDANTS function is used within a FIX statement, you can use member set functions such as @UDA and @ATTRIBUTE. For example:

  @UDA(dimName, uda)

  @ATTRIBUTE (attMbrName)

In this case, you can choose whether to use the @LIST function. For example, both of the following statements are valid, and the statements return the same results.

Example using only @ATTRIBUTE:

FIX
  (@ILDESCENDANTS(@ATTRIBUTE(Caffeinated_True), @ATTRIBUTE(Ounces_12), "200-40"))
  ...
ENDFIX;

Example using @LIST and @ATTRIBUTE:

FIX
  (@ILDESCENDANTS(@LIST(@ATTRIBUTE(Caffeinated_True), @ATTRIBUTE(Ounces_12), "200-40")))
  ...
ENDFIX;

Caution:

All members of the specified member list must be from the same dimension.
**genLevNum**
Optional. The integer value that defines the absolute generation or level number up to which to select members. A positive integer defines a generation number. A value of 0 or a negative integer defines a level number.

**Example**
All examples are from the Sample.Basic database.

@ILDESCENDANTS(@LIST("100", "200", "300"))

Returns 100 (a specified member); 100-10, 100-20, 100-30 (the descendants of 100); 200 (a specified member); and 200-10, 200-20, 200-30, and 200-40 (the descendants of 200); 300 (a specified member); and 300-10, 300-20, 300-30 (the descendants of 300).

@ILDESCENDANTS(@LIST("Market"), -1)

Returns Market (the specified member); and East, West, South, and Central (the descendants of Market to level 1).

**FIX**
(
@ILDESCENDANTS(@UDA(Market, "Major Market"))
...  
ENDFIX;

Returns East (a specified member): New York, Massachusetts, Florida, Connecticut, and New Hampshire (the descendants of East); Central (a specified member); Illinois, Ohio, Wisconsin, Missouri, Iowa, and Colorado (the descendants of Central); California and Texas (specified members, which do not have descendants).

**FIX**
(
@ILDESCENDANTS(@ATTRIBUTE(Caffeinated_True)@ATTRIBUTE(Ounces_12)="200-40")
...  
ENDFIX;

Returns 100-10, 100-20, 200-10, 300-30 (caffeinated, 12-ounce drinks); and 200-40 (a specified member). None of these members have descendants.

**See Also**
- @ANCESTORS
- @CHILDREN
- @IDESCENDANTS
- @ILANCESTORS
- @IRDESCENDANTS
- @ISDESC
- @LANCESTORS
- @LDESCENDANTS
@ILSIBLINGS

Returns the specified member and its left siblings.

Syntax

@ILSIBLINGS (mbrName)

Parameters

mbrName
Any valid single member name, or a function that returns a single member.

Notes

This function returns the specified member and all of the left siblings of the member. Left siblings are children that share the same parent as the member and that precede the member in the database outline.

This member set function can be used as a parameter of another function, where that parameter is a list of members.

Essbase sorts the generated list of members starting with the left siblings of the member (that is, siblings appearing above the member in the database outline) in ascending order. Using Sample Basic as an example, if you specify 200-30 for mbrName, Essbase returns 200-10, 200-20, 200-30 (in that order). This order is important to consider when you use this function with certain forecasting and statistical functions.

Example

In the Sample Basic database:

@ILSIBLINGS(Florida)

Returns New York, Massachusetts, and Florida (in that order). New York and Massachusetts appear above Florida in the Sample Basic outline.

@ILSIBLINGS(Qtr3)

Returns Qtr1, Qtr2, and Qtr3 (in that order). Qtr1 and Qtr2 appear above Qtr3 in the Sample Basic outline.

See Also

@LSIBLINGS
@INT

Returns the next lowest integer value of expression.

Syntax

@INT (expression)

Parameters

expression
Member specification or mathematical expression that generates a numeric value.

Example

The following example is based on the Sample Basic database. Assume that the Profit % member is not tagged as Dynamic Calc.

The following formula rounds the values for West down to the nearest integer.

West=@INT (@SUM (@CHILDREN(West)))

This example produces the following report:

<table>
<thead>
<tr>
<th>Profit %</th>
<th>Cola</th>
<th>Feb</th>
<th>Mar</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Jan</td>
<td></td>
<td></td>
</tr>
<tr>
<td>California</td>
<td>38.64</td>
<td>37.98</td>
<td>38.37</td>
</tr>
<tr>
<td>Oregon</td>
<td>17.50</td>
<td>16.13</td>
<td>16.11</td>
</tr>
<tr>
<td>Washington</td>
<td>29.23</td>
<td>30.90</td>
<td>32.00</td>
</tr>
<tr>
<td>Utah</td>
<td>23.08</td>
<td>23.08</td>
<td>20.97</td>
</tr>
<tr>
<td>Nevada</td>
<td>-3.95</td>
<td>-6.76</td>
<td>-5.33</td>
</tr>
<tr>
<td>West</td>
<td>104</td>
<td>101</td>
<td>102</td>
</tr>
</tbody>
</table>

See Also

• @ABS
• @REMAINDER
• @ROUND
• @TRUNCATE

@INTEREST

Calculates the simple interest in balanceMbr at the rate specified by creditrateMbrConst if the value specified by balanceMbr is positive, or at the rate specified by borrowrateMbrConst if balanceMbr is negative. The interest is calculated for each time period specified by XrangeList.
Syntax

@INTEREST (balanceMbr, creditrateMbrConst, borrowrateMbrConst [, XrangeList])

Parameters

**balanceMbr**
Single member specification representing the balance at the time the interest is calculated.

**creditrateMbrConst**
Single member specification, variable name, or numeric expression providing a constant value. The value must be a decimal number that corresponds to a percentage. The value represents the per-period interest rate.

**borrowrateMbrConst**
Single member specification, variable name, or numeric expression providing a constant value. The value must be a decimal number corresponding to a percentage value. The value represents the per-period interest rate.

**XrangeList**
Optional parameter specifying the time period over which the interest is calculated. If a range is not specified, Essbase uses the level 0 members from the dimension tagged as Time. Can be a valid member name, a comma-delimited list of member names, cross dimensional members, or a return value from a member set function or range function (including @XRANGE). For more information about rangeList and XrangeList, see Range List Parameters in the topic Range and Financial Functions.

Notes

Financial functions never return a value; rather, they calculate a series of values internally based on the range specified.

Example

The following example calculates the interest for Balance, using Credit Rate for positive balances and using Borrow Rate for negative balances. The results are placed in Interest Amount for each fiscal year.

"Interest Amount" = @INTEREST(Balance,"Credit Rate","Borrow Rate", FY1998:FY2001,FY2002,FY2003);

This example produces the following report:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Balance</td>
<td>2000.00</td>
<td>3000.00</td>
<td>-1000.00</td>
<td>3000.00</td>
<td>9000.00</td>
<td>-6000.00</td>
</tr>
<tr>
<td>Credit Rate</td>
<td>0.065</td>
<td>0.065</td>
<td>0.065</td>
<td>0.065</td>
<td>0.065</td>
<td>0.065</td>
</tr>
</tbody>
</table>
The following example assumes a Year dimension is added to Sample Basic. It calculates interest using a multidimensional range.

```plaintext
FIX ("100-10", "New York")
"Interest Amount" = @INTEREST (Balance, "Credit Rate", "Borrow Rate", @XRANGE("2011"->"Sep", "2012"->"Mar"));
ENDFIX
```

The above calculation is performed across the following multidimensional range specified by XrangeList:

- 2011->Sep
- 2011->Oct
- 2011->Nov
- 2011->Dec
- 2012->Jan
- 2012->Feb
- 2012->Mar

See Also

@COMPOUND

@INTERSECT

Returns the intersection of members that appear in two specified lists of members.

Syntax

```plaintext
@INTERSECT(list1, list2)
```

Parameters

**list1**
The first list of members.

**list2**
The second list of members.

Notes

This function treats shared members as distinct from their referenced members; therefore, they do not intersect.

Example

The following examples use the Sample.Basic database.

```plaintext
@INTERSECT(@CHILDREN("100"), @ATTRIBUTE(Can)) returns 100-10 and 100-20.
```
@INTERSECT(@CHILDREN("Colas"), @CHILDREN("Diet Drinks")); returns an empty set, because shared members are considered distinct from their referenced members.

FIX (@INTERSECT(@CHILDREN("100-10"), @CHILDREN("Diet Drinks")))

    Sales = 500;

ENDFIX;

The @INTERSECT expression evaluates to an empty set; therefore, the FIX statement sets all the values of Sales to 500.

See Also
• @MERGE
• @REMOVE

@IRDESCENDANTS

Returns the specified member and all its descendants, or all descendants down to a specified generation or level, including descendants of any occurrences of the specified member as a shared member.

You can use this function as a parameter of another function, where that parameter is a list of members. In the absence of shared members, this function behaves the same as @IDESCENDANTS.

Syntax

@IRDESCENDANTS (mbrName[, genLevNum | genLevName])

Parameters

mbrName
Any valid single member name, or a function that returns a single member.

genLevNum
Optional. An integer value that defines the absolute generation or level number down to which to select the members. A positive integer defines a generation number. A value of 0 or a negative integer defines a level number.

genLevName
Optional. Level name or generation name down to which to select the members.

Notes

• The order of members in the result list is important to consider when you use this function with certain forecasting and statistical functions. Essbase generates the list of members in the following sequence: If a shared member is encountered, the above steps are repeated on the member being shared.

1. The specified member
2. The nearest descendant of the member
3. The next nearest descendant of the member, and so on
• You can use @RDESCENDANTS to exclude the specified member and include descendants of shared members.

• You can use @IDESCENDANTS to include the specified member and exclude descendants of shared members.

• You can use @DESCENDANTS to exclude the specified member and descendants of shared members.

**Example**

**Example 1**

Assume a variation of the Sample Basic database such that the Product dimension includes the following members:

<table>
<thead>
<tr>
<th>Product</th>
<th>100</th>
<th>100-10</th>
<th>100-20</th>
<th>100-30</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>200</td>
<td>200-10</td>
<td>200-20</td>
<td>200-30</td>
</tr>
</tbody>
</table>

Diet

| 100 (Shared Member) |
| 200 (Shared Member) |

Diet has two children "100" and "200" instead of "100-10", "200-20" and "300-30". The members "100" and "200" are shared members.

@IRDESCENDANTS(Diet)

Returns the members: Diet, 100, 100-10, 100-20, 100-30, 200, 200-10, 200-20, 200-30, 200-40 (in that order).

**Example 2**

@IRDESCENDANTS(East)

Returns East, New York, Massachusetts, Florida, Connecticut, and New Hampshire (in that order) and is exactly the same as @IDESCENDANTS(East).

**See Also**

• @DESCENDANTS
• @IANCESTORS
• @ICHILDREN
• @IDESCENDANTS
• @ISDESC
• @ISIBLINGS
• @RDESCENDANTS
@IRR

Calculates the Internal Rate of Return on a cash flow that must contain at least one investment (negative) and one income (positive) value.

Also see @IRREX.

Syntax

@IRR (cashflowMbr, discountFlag[, XrangeList])

Parameters

cashflowMbr
Single member specification.

discountFlag
Member specification, variable name, or numeric expression providing a constant value of either 1 or 0. discountFlag indicates whether the function should discount from the first period. 1 means do not discount from the first period.

XrangeList
Optional parameter specifying the range over which the rate is calculated. If a range is not specified, Essbase uses the level 0 members from the dimension tagged as Time. Can be a valid member name, a comma-delimited list of member names, cross dimensional members, or a return value from a member set function or range function (including @XRANGE).

Notes

• Financial functions never return a value; rather, they calculate a series of values internally based on the range specified.

• This function returns #MISSING if all cash flows are zero.

• This function provides an initial guess of 0.07. This value cannot be changed, in contrast to similar functions in Excel. Because results depend in part on the initial guess, any difference in the initial guess may result in a different result. Even if both Excel and Essbase start with the same initial guess, results may differ. This is because there may be more than one solution to an equation, and the algorithm stops looking when it finds a valid solution. Which solution is found first may differ based on the algorithm. Although leading or trailing zeros do not matter in a mathematical context, the algorithm may behave differently and find a different root because of the presence of leading or trailing zeros. If you need identical solutions regardless of the presence of leading or trailing zeros, you may wish to create a custom-defined function to handle these issues.

Example

This example calculates the Internal Rate of Return (Return) on a cash flow (Cash).

Return = @IRR(Cash, 0, FY1998:FY2000, FY2001:FY2003);
This example produces the following report:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Cash</td>
<td>(1,000)</td>
<td>500</td>
<td>600</td>
<td>500</td>
<td>#MISSING</td>
<td>#MISSING</td>
</tr>
<tr>
<td>Rate</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>#MISSING</td>
<td>#MISSING</td>
</tr>
<tr>
<td>Return</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

The following example assumes a Year dimension is added to Sample Basic. It calculates the return using a multidimensional range.

```plaintext
FIX ("100-10", "New York")
"Return" = @IRR(Cash, 0, @XRANGE("2011"->"Sep", "2012"->"Mar"));
ENDFIX
```

The above calculation is performed across the following multidimensional range specified by XrangeList:

- 2011->Sep
- 2011->Oct
- 2011->Nov
- 2011->Dec
- 2012->Jan
- 2012->Feb
- 2012->Mar

@IRREX

Calculates the Internal Rate of Return on a cash flow that must contain at least one investment (negative) and one income (positive) value. Includes functionality to configure the initial guess and the number of iterations the algorithm can make.

@IRREX is an extension of @IRR, in which the initial guess of 0.07 cannot be changed.

Syntax

```plaintext
@IRREX (cashflowMbr, discountFlag[, [guess], [number_of_iteration], [STORECALCVALUE | STOREMISSING], [XrangeList])
```

Parameters

- **cashflowMbr**
  Single member specification.

- **discountFlag**
  Member specification, variable name, or numeric expression providing a constant value of either 1 or 0. Indicates whether the function should discount from the first period. 0 means discount from the first period, and 1 means do not discount from the first period.
guess
Optional. The starting guess for estimated IRR. If not specified, the default guess of 0.07 is used.

number_of_iteration
Optional. The number of iterations the Newton Raphson algorithm will loop through. (Newton Raphson is the mathematical method used for finding the IRR using the IRREX function.) The default value is 300.

STORECALCVALUE | STOREMISSING
Optional. STORECALCVALUE tells Essbase to always store the calculated value even when the IRR calculation returns 'false' results. This is the default. Optional. STOREMISSING tells Essbase to store #MISSING value when the IRR calculation returns false results after the specified number of iterations.

XrangeList
Optional parameter specifying the range over which the rate is calculated. If a range is not specified, Essbase uses the level 0 members from the dimension tagged as Time. Can be a valid member name, a comma-delimited list of member names, cross dimensional members, or a return value from a member set function or range function (including @XRANGE). For more information about rangeList and XrangeList, see Range List Parameters in the topic Range and Financial Functions.

Notes
• Financial functions never return a value; rather, they calculate a series of values internally based on the range specified.
• This function returns #MISSING if all cash flows are zero.
• This function provides functionality to configure the initial guess and the number of iterations the algorithm can make. Even if both Essbase and similar functions in Excel start with the same initial guess, results may differ. This is because there may be more than one solution to an equation, and the algorithm stops looking when it finds a valid solution. Which solution is found first may differ based on the algorithm. Although leading or trailing zeros do not matter in a mathematical context, the algorithm may behave differently and find a different root because of the presence of leading or trailing zeros. If you need identical solutions regardless of the presence of leading or trailing zeros, you may wish to create a custom-defined function to handle these issues.

Example
@IRREX(IRROut1, 0, 0.02, 500, STOREMISSING, "2006":"2009");

The starting guess is 0.02 (2%). @IRREX iterates 500 times, and stores #MISSING if the solution does not converge.

@IRREX(IRROut1, 0, , ,STOREMISSING, "2006":"2009");

The starting guess and iteration values are omitted (NULL). Note: The commas (,) are required even when passing null arguments.
The following example assumes a Year dimension is added to Sample Basic. The rate is calculated using a multidimensional range.

```
FIX ("100-10", "New York")
Return = @IRREX(IRROut1,0, , ,STOREMISSING, @XRANGE("2011"->"Sep", "2012"->"Mar");
ENDFIX
```

The above calculation is performed across the following multidimensional range specified by XrangeList:

- 2011->Sep
- 2011->Oct
- 2011->Nov
- 2011->Dec
- 2012->Jan
- 2012->Feb
- 2012->Mar

@IRSIBLINGS

Returns the specified member and its right siblings.

**Syntax**

```
@IRSIBLINGS (mbrName)
```

**Parameters**

**mbrName**
Any valid single member name, or a function that returns a single member.

**Notes**

This function returns the specified member and all of the right siblings of the specified member. Right siblings are children that share the same parent as the member and that follow the member in the database outline.

This member set function can be used as a parameter of another function, where that parameter is a list of members.

Essbase sorts the generated list of members starting with the specified member, followed by the right siblings of the member (that is, siblings appearing below the member in the database outline) in ascending order. Using Sample Basic as an example, if you specify 200-20 for mbrName, Essbase returns 200-20, 200-30, 200-40 (in that order). This order is important to consider when you use this function with certain forecasting and statistical functions.
Example
In the Sample Basic database:

@IRSIBLINGS(Florida)


@IRSIBLINGS(Qtr3)

Returns Qtr3 and Qtr4 (in that order). Qtr4 appears below Qtr3 in the Sample Basic outline.

See Also
@RSIBLINGS

@ISACCTYPE

Returns TRUE if the current member has the associated accounts tag.

Syntax

@ISACCTYPE (tag)

Parameters

tag
Valid accounts tag defined in the current database. Any of these values may be used: First, Last, Average, Expense, and Twopass. To ensure that the tag is resolved as a string rather than a member name, enclose the tag in quotation marks.

Example
The following example is based on the Sample Basic database. For members with the Expense accounts tag, the formula uses the @ABS function to calculate Budget as the absolute value of Budget.

IF (@ISACCTYPE("Expense"))
    Budget = @ABS(Budget);
ENDIF;

@ISANCEST

Returns TRUE if the current member is an ancestor of the specified member. This function excludes the specified member.

Syntax

@ISANCEST (mbrName)
Parameters

mbrName
Any valid single member name, or a function that returns a single member.

Example

In the Sample Basic database:

@ISANCEST(California)

Returns TRUE for Market, West

@ISANCEST(West)

Returns FALSE for California, West, East

See Also

@ISANCEST

@ISATTRIBUTE

Returns TRUE if the current member under calculation matches the attribute or varying attribute name specified in \textit{attMbrName}.

Syntax

@ISATTRIBUTE \{attMbrName\}

Parameters

attMbrName
Single attribute member name or member combination.

Notes

- This function provides the same functionality as @ISMBR (@ATTRIBUTE(\textit{attMbrName})), but is faster.
- You may have duplicate Boolean, date, and numeric attribute member names in your outline. For example, 12 can be the attribute value for the size (in ounces) of a product as well as the value for the number of packing units for a product. To distinguish duplicate member names, specify the full attribute member name (for example, @ISATTRIBUTE(Ounces_12)).

Example

Consider the following calculation script, based on the Sample Basic database:

/* To increase the marketing budget for markets with large populations */
Marketing (  
    IF (@ISATTRIBUTE(Large))  
        Marketing = Marketing * 1.1;  
)
See Also

- @ISMBRWITHATTR
- SET SCAPERSPECTIVE

@ISCHILD

Returns TRUE if the current member is a child of the specified member. This function excludes the specified member.

Syntax

@ISCHILD (mbrName)

Parameters

mbrName
Any valid single member name, or a function that returns a single member.

Example

In the Sample Basic database:

@ISCHILD(East)

Returns TRUE for New York, Florida, Connecticut

@ISCHILD(Margin)

Returns FALSE for Measures, Profit, Margin

See Also

@ISICHILD

@ISDESC

Returns TRUE if the current member is a descendant of the specified member. This function excludes the specified member.

Syntax

@ISDESC (mbrName)

Parameters

mbrName
Any valid single member name, or a function that returns a single member.
Example
In the Sample Basic database:

@ISDESC(Market)

Returns TRUE for West, California, Oregon, Washington, Utah, Nevada

@ISDESC(Profit)

Returns FALSE for Measures, Profit, Profit %

@ISGEN

Returns TRUE if the current member of the specified dimension is in the specified generation.

Syntax

@ISGEN (dimName, genName | genNum)

Parameters

dimName
The name of a dimension.

genName or genNum
A generation name or a non-negative integer value that defines the number of a generation.

Example
In the Sample Basic database:

@ISGEN(Measures,3)

Returns TRUE if the current member is Margin, Total Inventory, or Margin %, because these members are all in generation 3 of the Measures dimension.

@ISGEN(Market,2)

Returns FALSE if the current member is New York or Market, because these members are not in generation 2 of the Market dimension.

See Also

• @ISSAMEGEN
• @ISLEV
@ISIANCEST

Returns TRUE if the current member is the specified member or an ancestor of the specified member. This function includes the specified member.

Syntax

@ISIANCEST (mbrName)

Parameters

mbrName
Any valid single member name, or a function that returns a single member.

Example

In the Sample Basic database:

@ISIANCEST(California)

Returns TRUE for Market, West, and California. California is the specified member, and West and Market are ancestors of California.

@ISIANCEST(Qtr1)

Returns FALSE for Jan, Feb, Mar, Qtr2. None of these members is the specified member (Qtr1) or an ancestor of Qtr1.

See Also

@ISANCEST

@ISIBLINGS

Returns the specified member and all siblings of that member. This function can be used as a parameter of another function, where that parameter is a list of members.

Syntax

@ISIBLINGS (mbrName)

Parameters

mbrName
Any valid single member name, or a function that returns a single member.

Notes

Essbase sorts the generated list of members in ascending order. Using Sample Basic as an example, if you specify 200-30 for mbrName, Essbase returns 200-10, 200-20, 200-30, 200-40 (in that order). This order is important to consider when you use this function with certain forecasting and statistical functions.
Example
In the Sample Basic database:

@ISIBLINGS(California)
returns California, Oregon, Washington, Utah, and Nevada (in that order), because these members are siblings of California.

@ISIBLINGS(Qtr2)
returns Qtr1, Qtr2, Qtr3, and Qtr4 (in that order), because these members are siblings of Qtr2.

See Also
• @NEXTSIBLING
• @PREVSIBLING
• @SHIFTSIBLING
• @SIBLINGS

@ISICHILD
Returns TRUE if the current member is the specified member or a child of the specified member.

Syntax
@ISICHILD (mbrName)

Parameters
mbrName
Any valid single member name, or a function that returns a single member.

Example
In the Sample Basic database:

@ISICHILD(South)

Returns TRUE for Texas, Oklahoma, Louisiana, New Mexico, South

@ISICHILD(Profit)

Returns FALSE for Measures, Sales

See Also
@ISCHILD
@ISIDESC

Returns TRUE if the current member is the specified member or a descendant of the specified member.

Syntax

@ISIDESC (mbrName)

Parameters

mbrName
Any valid single member name, or a function that returns a single member.

Example

In the Sample Basic database:

@ISIDESC(South)

Returns TRUE for Texas, Oklahoma, Louisiana, New Mexico, South

@ISIDESC(West)

Returns FALSE for Market, East, South, and Central

See Also

@ISDESC

@ISIPARENT

Returns TRUE if the current member is the specified member or the parent of the specified member.

Syntax

@ISIPARENT (mbrName)

Parameters

mbrName
Any valid single member name, or a function that returns a single member.

Example

In the Sample Basic database:

@ISIPARENT(Qtr1)
@ISIPARENT(Margin)

Returns TRUE for Year, Qtr1.

Returns FALSE for Measures, Sales.

See Also

@ISPARENT

@ISISIBLING

Returns TRUE if the current member is the specified member or a sibling of the specified member.

Syntax

@ISISIBLING (mbrName)

Parameters

mbrName
Any valid single member name, or a function that returns a single member.

Example

In the Sample Basic database:

@ISISIBLING(Qtr2)

Returns TRUE for Qtr1, Qtr2, Qtr3, and Qtr4.

@ISISIBLING(Actual)

Returns FALSE for Scenario.

See Also

@ISSIBLING

@ISLEV

Returns TRUE if the current member of the specified dimension is in the specified level.

Syntax

@ISLEV (dimName, levName | levNum)
Parameters

dimName
Name of a dimension.

levName | levNum
A level name or a non-negative integer value that defines the number of a level.

Example
In the Sample Basic database:

@ISLEV(Market, 0)

Returns TRUE if the current member of Market is New York, California, Texas, or Illinois.

@ISLEV(Year, 1)

Returns FALSE if the current member of Year is Jan, Feb, or Mar.

See Also
• @ISSAMELEV
• @ISGEN

@ISMBR

Returns TRUE if the current member matches any one of the specified members.

Syntax

@ISMBR (mbrName | rangeList | mbrList)

Parameters

mbrName
Any valid single member name or member combination, or a function that returns a single member or member combination.

rangeList
A valid member name, a comma-delimited list of member names, member set functions, and range functions.

mbrList
A comma-delimited list of members.

Notes
If a cross-dimensional (->) member is included, that term evaluates as TRUE only if all the components of the cross-dimensional member match the current member list.

If any term returns TRUE, this function returns TRUE.
Example

In the Sample Basic database:

@ISMBR("New York":"New Hampshire")

Returns TRUE for Florida.

@ISMBR(@CHILDREN(Qtr1))

Returns FALSE for Qtr2, Year.

@ISMBRUDA

Returns TRUE if the specified user-defined attribute (UDA) exists for the specified member at calculation time.

Syntax

@ ISMBRUDA(mbrName, UDAStr)

Parameters

mbrName
Any valid single member name, or a function that returns a single member.

UDAStr
User-defined attribute (UDA) name string.

Notes

If you specify a nonexistent member name, the calculation script verification fails.

Example

The following examples use the Sample.Basic database.

@ISMBRUDA ("New York", "Major Market") and @ISMBRUDA([Market].[New York], "Major Market") both return true.

@ISMBRUDA("New York", "Small Market") AND @ISCHILD("Market")

Because “New York” is not a small market, the first condition returns false.

IF(@ISMBRUDA("New York")

Because UDAStr is omitted, the verification fails.

@ISMBRWITHATTR

Returns TRUE if the current member belongs to the list of base members that are associated with an attribute that satisfies the conditions you specify.
Syntax

@ISMBRWTHATTR (dimName, "operator", value)

Parameters

**dimName**
Single varying attribute dimension name.

**operator**
Operator specification, which must be enclosed in quotation marks (""").

**value**
A value that, in combination with the operator, defines the condition that must be met. The value can be a varying attribute member specification, a constant, or a date-format function (that is, @TODATE).

Notes

- This function provides the same functionality as @ISMR(@WITHATTR()), but is faster.
- This function is a superset of the @ISATTRIBUTE function. The following two formulas return the same member set:
  
  @ISATTRIBUTE(Bottle)
  @ISMBRWTHATTR("Pkg Type","="Bottle)
  
  However, the following formula can be performed only with @ISMBRWTHATTR (not with @ISATTRIBUTE) because you specify a condition:
  
  @ISMBRWTHATTR(Ounces,">","16")
  
- If you specify a date attribute with the @ISMBRWTHATTR function, you must use @TODATE in the string parameter to convert the date string to a number.
- The following operators are supported:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than or equal to</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equal to</td>
</tr>
<tr>
<td>= =</td>
<td>Equal to</td>
</tr>
<tr>
<td>&lt;&gt; or !=</td>
<td>Not equal to</td>
</tr>
<tr>
<td>IN</td>
<td>In</td>
</tr>
</tbody>
</table>

When using Boolean attributes with this function, use only the actual Boolean attribute member name, or use 1 (for True or Yes) or 0 (for False or No). You cannot use True/Yes and False/No interchangeably.
See Also

- @ATTRIBUTE
- @ATTRIBUTEVAL
- @ISATTRIBUTE
- SET SCAPERSPECTIVE
- @TODATE
- @WITHATTR

@ISPARENT

Returns TRUE if the current member is the parent of the specified member. This function excludes the specified member.

Syntax

@ISPARENT (mbrName)

Parameters

mbrName
Any valid single member name, or a function that returns a single member.

Example

In the Sample Basic database:

@ISPARENT("New York")

Returns TRUE for East.

@ISPARENT(Profit)

Returns FALSE for Margin.

See Also

@ISIPARENT

@ISRANGENONEMPTY

Tests for the existence of data values to improve performance of complex dense processing. If this function returns true, values exist for the specified range. If it returns false, the range is empty.

Syntax

@ISRANGENONEMPTY(ZEROASDATA|ZEROASMISSG, mbrList)
Parameters

**ZEROASDATA**
Zero (0) values are treated as data.

**ZEROASMISSG**
Zero (0) values are treated as #MISSING.

**mbrList**
A valid member name, a comma-delimited list of member names, or a member set function that returns a list of members from the same dimension. If you use the range operator or a function, the order of *mbrList* is dictated by the database outline order.

Notes
The definition of "emptiness" depends on your use of the first parameter, which describes how zero (0) values are treated.

Example
The following examples use the Sample.Basic database.

Example 1

```plaintext
@ISRANGENONEMPTY(ZEROASDATA, Sales->Cola)
```

Because the intersection of Cola and Sales contains non-#MISSING values, the condition returns TRUE.

Example 2

```plaintext
//ESS_LOCALE English_UnitedStates.Latin1@Binary
FIX (Budget)
    Sales (IF(@ISRANGENONEMPTY(ZEROASMISSG, Jan:Mar))
        Sales = 500;
    ENDIF);
ENDFIX
```

If there is any value except #MISSING in the range Jan:Mar in the database, the script returns TRUE, and all the Sales->Budget values in the database are changed to 500.

@ISSAMEGEN

Returns TRUE if the current member is the same generation as the specified member.

Syntax

```plaintext
@ISSAMEGEN (mbrName)
```
**Parameters**

**mbrName**
Any valid single member name, or a function that returns a single member.

**Example**
In the Sample Basic database:

@ISSAMEGEN(West)

Returns TRUE for East.

@ISSAMEGEN(West)

Returns FALSE for California.

**See Also**
- @GEN
- @ISGEN
- @ISSAMELEV

---

**@ISSAMELEV**

Returns TRUE if the current member is the same level as the specified member.

**Syntax**

@ISSAMELEV (mbrName)

**Parameters**

**mbrName**
Any valid single member name, or a function that returns a single member.

**Example**
In the Sample Basic database:

@ISSAMELEV(Sales)

Returns FALSE for Total Expenses.

@ISSAMELEV(Jan)

Returns TRUE for Apr, Jul, Oct.

**See Also**
- @ISLEV
@ISSIBLING

Returns TRUE if the current member is a sibling of the specified member. This function excludes the specified member.

Syntax

@ISSIBLING (mbrName)

Parameters

mbrName
Any valid single member name, or a function that returns a single member.

Example

In the Sample Basic database:

@ISSIBLING("New York")

Returns TRUE for Florida, New Hampshire.

@ISSIBLING(Sales)

Returns FALSE for Margin.

See Also

@ISISIBLING

@ISUDA

Returns TRUE if the specified user-defined attribute (UDA) exists for the current member of the specified dimension at the time of the calculation.

Syntax

@ISUDA (dimName, UDAStr)

Parameters

dimName
Dimension name specification that contains the member you are checking.

UDAStr
User-defined attribute (UDA) name string.
Notes

- Essbase checks to see if the UDA is defined for the current member of the specified dimension at calculation time. It returns TRUE if the UDA is defined, FALSE if not.
- For more information about UDAs, see *Designing and Maintaining Essbase Cubes*.

Example

The following example is based on the Sample Basic database. The Market dimension has members that indicate a geographic location. Some members represent major markets. The example below calculates the database and stores a budget amount for the upcoming year based on the actual amount from this year. A different sales growth rate is applied to major markets than to small markets.

```fix
FIX (Budget)
    Sales (IF(@ISUDA(Market,"Major Market"))
        Sales = Sales->Actual * 1.2;
    ELSE
        Sales = Sales->Actual * 1.1;
    ENDIF;)
ENDFIX
```

The preceding example tests to see if the current member of Market has a UDA called "Major Market". If it does, the Budget -> Sales value is set to 120% of Actual -> Sales. If it does not, the Budget -> Sales value is set to 110% of Actual -> Sales.

See Also

- @ISMBRUDA
- @UDA

@ANCESTORS

Returns all ancestors of the members in the specified member list or all ancestors up to a specified generation or level. This function excludes the specified members.

You can use this function as a parameter of another function, where the function requires a list of members.

Syntax

```language
@ANCESTORS (memberSetFunction) [,genLevNum]
```

Parameters

- **memberSetFunction**
  A member set function that returns a list of members.
  How the @ANCESTORS function is used determines which member set functions are allowed. Follow these guidelines:
• If the @LANCESTORS function is used alone (not within a FIX statement), you must use the @LIST function and specify member names. For example:

@LIST(mbr1, mbr2, ...)

• If the @LANCESTORS function is used within a FIX statement, you can use member set functions such as @UDA and @ATTRIBUTE. For example:

@UDA(dimName, uda)

@ATTRIBUTE (attMbrName)

In this case, you can choose whether to use @LIST. For example, both of the following statements are valid, and the statements return the same results.

Example using only @ATTRIBUTE:

FIX(@LANCESTORS(@ATTRIBUTE(Caffeinated_True), @ATTRIBUTE(Ounces_12), "200-40"))

... ENDFIX;

Example using @LIST and @ATTRIBUTE:

FIX(@LANCESTORS(@LIST(@ATTRIBUTE(Caffeinated_True), @ATTRIBUTE(Ounces_12), "200-40")))

... ENDFIX;

⚠️ Caution:

All members of the specified member list must be from the same dimension.

genLevNum
Optional. The integer value that defines the absolute generation or level number up to which to select members. A positive integer defines a generation number. A value of 0 or a negative integer defines a level number.

Example

All examples are from the Sample.Basic database.

@LANCESTORS(@LIST("100-10","200-20"), 2)

Returns 100 (the ancestor of 100-10); and 200 (the ancestor of 200-20). Excludes Product because it is at generation 1.

@LANCESTORS(@LIST("100","100-10"))
Returns Product (the ancestor of 100); and 100 (the ancestor of 100-10). The result does not contain duplicate members.

@@LANCESTORS(@LIST("100","Product","200"))

Returns Product (the ancestor of 100 and 200). The result does not contain duplicate members.

FIX(@LANCESTORS(@UDA(Market,"New Market")),2)

... ENDFIX;

Returns West, South, and Central (the ancestors, to generation 2, for the members in the Market dimension that are associated with the New Market attribute).

FIX(@LANCESTORS(@ATTRIBUTE(Caffeinated_True),@ATTRIBUTE(Ounces_12),"200-40"))

... ENDFIX;

Returns 100, 200, 300, and Product (the ancestors of 100-10, 100-20, 200-10, 300-30 —caffeinated, 12-ounce drinks, and 200-40).

See Also
• @ANCESTORS
• @ANCESTORS
• @ANCESTORS

@LDESCENDANTS

Returns all descendants of the members in the specified member list or all descendents down to the specified generation or level. This function excludes the specified members.

You can use this function as a parameter of another function, where the function requires a list of members.

Syntax

@LDESCENDANTS ((memberSetFunction) [,genLevNum])

Parameters

memberSetFunction
A member set function that returns a list of members.
How this function is used determines which member set functions are allowed. Follow these guidelines:
• If this function is used alone (not within a FIX statement), you must use @LIST and specify member names. For example:

@LIST(mbr1,mbr2,...)

• If @LDESCENDANTS is used within a FIX statement, you can use member set functions such as @UDA and @ATTRIBUTE. For example:

@UDA(dimName, uda)

@ATTRIBUTE (attMbrName)

In this case, you can choose whether to use @LIST. For example, both of the following statements are valid, and the statements return the same results.

Example using only @ATTRIBUTE:

FIX
 (@LDESCENDANTS(@ATTRIBUTE(Caffeinated_True),@ATTRIBUTE(Ounces_12),"200-40")
 ... 
ENDFIX;

Example using @LIST and @ATTRIBUTE:

FIX
 (@LDESCENDANTS(@LIST(@ATTRIBUTE(Caffeinated_True),@ATTRIBUTE(Ounces_12),"200-40")))
 ... 
ENDFIX;

⚠️ Caution:

All members of the specified member list must be from the same dimension.

genLevNum
Optional. The integer value that defines the absolute generation or level number up to which to select members. A positive integer defines a generation number. A value of 0 or a negative integer defines a level number.

Example

All examples are from the Sample.Basic database.

@LDESCENDANTS(@LIST("100","200","300"))
Returns 100-10, 100-20, 100-30 (the descendants of 100); 200-10, 200-20, 200-30, 200-40 (the descendants of 200); and 300-10, 300-20, 300-30 (the descendants of 300).

@LDESCENDANTS(@LIST("Market"),-1)

Returns East, West, South, and Central (the descendants of the specified member Market to level 1).

FIX
{@LDESCENDANTS(@UDA(Market,"Major Market"))}
...
ENDFIX;

Returns New York, Massachusetts, Florida, Connecticut, and New Hampshire (the descendants of the specified member East); and Illinois, Ohio, Wisconsin, Missouri, Iowa, and Colorado (the descendants of the specified member Central). California and Texas (specified members) are excluded because they do not have descendants.

FIX
{@LDESCENDANTS(@ATTRIBUTE(Caffeinated_True),@ATTRIBUTE(Ounces_12),"200-40")}
...
ENDFIX;

Returns an empty list as none of the specified members (100-10, 100-20, 200-10, 300-30, which are caffeinated, 12-ounce drinks, and 200-40) have descendants.

See Also
• @DESCENDANTS
• @IDESCENDANTS
• @ILDESCENDANTS

@LEV

Returns the level number of the specified member.

Syntax

@LEV(mbrName)

Parameters

mbrName
Any valid single member name, or a function that returns a single member.

Example

In the Sample Basic database:
Table 2-30  @LEV Results

<table>
<thead>
<tr>
<th>Function</th>
<th>Level Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>@LEV(Margin)</td>
<td>1</td>
</tr>
<tr>
<td>@LEV(&quot;New York&quot;)</td>
<td>0</td>
</tr>
</tbody>
</table>

See Also

- @CURLEV
- @GEN

@LEVMBRS

Returns all members with the specified level number or level name in the specified dimension.

Syntax

}@LEVMBRS (dimName, levName|levNum)

Parameters

dimName
Dimension name specification.

levName|levNum
A level name or an integer value that defines the number of a level. The integer value must be 0 or a positive integer.

Notes

- If you specify a name for the levName parameter, Essbase looks for a level with that name in the specified dimension.
- If you specify a number for the levName parameter (for example, 2), Essbase first looks for a level with a number string name. If no level name exists with that name, Essbase checks to see if the parameter is a valid level number.
- If you specify a temporary variable for the levName parameter, Essbase does not recognize the value of the variable. It looks in the outline for a level name with the same name as the temporary variable.
- For more information about levels and defining level names, see Generations and Levels in Designing and Maintaining Essbase Cubes.
- Essbase sorts the generated list of members in ascending order. Using Sample Basic as an example, if you specify @LEVMBRS(Product, 1), Essbase returns 100, 200, 300, 400, Diet (in that order). This order is important to consider when you use @LEVMBRS with certain forecasting and statistical functions.
- If you use a negative number for the level number, no syntax error is noted, but the calculation will fail with an error message.
Example

In the Sample Basic database:

@LEVMBRS(Measures,"Profit and Loss")
@LEVMBRS(Measures,0)

both return the following members if level 0 of the Measures dimension is named Profit and Loss:

Sales, COGS, Marketing, Payroll, Misc, Opening Inventory, Additions, Ending Inventory, Margin %, Profit %, and Profit per Ounce (in that order).

@LEVMBRS(Scenario,0)

Returns Actual, Budget, Variance, and Variance %.

The following example restricts the calculation to members with the combination Budget and one of the members of the Market dimension with a level name of "State".

FIX (Budget,@LEVMBRS(Market,State))
  CALC DIM (Year,Measures);
ENDFIX

See Also

@GENMBRS

@LIKE

Returns a member set of member names that match the specified pattern.

This function can be used on unique and duplicate-name outlines.

Syntax

@LIKE(pattern, topMbrinHierarchy, [escChar])

Parameters

pattern
The character pattern with which to compare to members in the outline, including a single wildcard character:

- %: The percentage sign allows matching to a string of any length (including zero length).
- _: The underscore allows matching on a single character in a member name.

topMbrinHierarchy
A fully qualified member name on which to base the member search. The specified member and its aliases, and all of its descendants, are included in the search.
To search the entire outline, provide an empty string (""") for this parameter. For example, @LIKE("100%", "").

ESCChar
Optional: A one-byte-length escape character to use if the wildcard character exists in member names.
If you do not specify an escape character, a backslash (\) is assumed.

Example
The following examples are based on the following duplicate-name outline:

```
Product
  100
    100-10
    100-10-10
    100-20
    100-30
  200
    200-10
    200-20
    200-30
  300
    300-10
    300-20
Diet
  100-10
    100-10-11
    200-10
    300-10
Bottle
  200-10
  300-20
```

@LIKE("100%", "Product")

**Returns members 100, 100-10, 100-20, and 100-30.**

@LIKE("30_", "Product")

**Returns member 300.**

@LIKE("200\_", "Product", "\")

If member 200 has children named 200_10 (note the underscore, _), 200-20 (note the dash, -), 200_30 and 200-40, returns those members whose name contains an underscore: 200_10 and 200_30.
@LIST

Creates and distinguishes lists that are processed by functions that require list arguments. Can be used to create \textit{expLists}, member lists, or \textit{rangeLists}. This function treats a collection of parameters as one entity.

\textbf{Syntax}

\begin{verbatim}
@LIST (argument1, argument2, ..., argumentN)
\end{verbatim}

\textbf{Parameters}

\textit{argument1, argument2, ..., argumentN}

The list of arguments that are collected and treated as one argument so they can be processed by the parent function. Arguments can be member names, member combinations, member set functions, range functions, and numeric expressions.

\textbf{Notes}

@LIST does not check for or eliminate duplicates.

\textbf{Example}

The following example is based on the Sample Basic database. Assume that the Year dimension contains an additional member, Sales Correl. @LIST is used with the \texttt{@CORRELATION} function to determine the sales relationship between a product's two peak periods (Jan-Mar and Apr-May):

\begin{verbatim}
FIX(Sales)
"Sales Correl" = @CORRELATION(SKIPNONE,
   @LIST(Jan,Feb,Mar),@LIST(Apr,May,Jun));
ENDFIX
\end{verbatim}

This example produces the following report:

\begin{center}
\begin{tabular}{lrr}
\hline
Colas & Actual & New York \\
Sales & & \\
\hline
Jan & 678 & \\
Feb & 645 & \\
Mar & 675 & \\
Apr & 712 & \\
\hline
\end{tabular}
\end{center}
@LN

Returns the natural logarithm (base e) of the specified expression.

Syntax

@LN (expression)

Parameters

expression
Single member specification, member combination, or other numeric expression. If less than or equal to 0, Essbase returns #MISSING.

Example

The following example is based on a variation of Sample Basic:

LN_Sales = @LN(Sales);

This example produces the following result:

<table>
<thead>
<tr>
<th></th>
<th>Jan</th>
<th>Feb</th>
<th>Cola East</th>
<th>Nov</th>
<th>Dec</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sales</td>
<td>100</td>
<td>110</td>
<td>120</td>
<td>...</td>
<td>0</td>
</tr>
<tr>
<td>LN_Sales</td>
<td>4.65052</td>
<td>4.70048</td>
<td>4.78749</td>
<td>...</td>
<td>#MISSING</td>
</tr>
</tbody>
</table>

See Also

- @LOG10
- @LOG
- @EXP

@LOG

Returns the result of a logarithm calculation where you can specify both the base to use and the expression to calculate.

Syntax

@LOG (expression [, base])
Parameters

**expression**
Single member specification, variable name, function, or other numeric expression. If less than or equal to 0, Essbase returns #MISSING.

**base**
Optional. Single member specification, member combination, or numeric expression.
- If the base value is #MISSING, less than or equal to 0, or close to 1, Essbase returns #MISSING.
- If the base is omitted, Essbase calculates the base-10 logarithm of the specified expression. @LOG(Sales) is equivalent to @LOG10(Sales).

Notes
The @LOG function returns the logarithm of *expression* calculated using the specified *base*. @LOG(x,b) is equivalent to \( \log_b(x) \).

Example
The following example is based on a variation of Sample Basic:

\[
\text{LOG2}_\text{Sales} = \text{@LOG(Sales,2)};
\]

This example produces the following result:

<table>
<thead>
<tr>
<th>Cola East</th>
<th>Jan</th>
<th>Feb</th>
<th>Mar</th>
<th>Nov</th>
<th>Dec</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sales</td>
<td>100</td>
<td>#MISSING</td>
<td>120</td>
<td>. . .</td>
<td>0</td>
</tr>
<tr>
<td>LOG2_Sales</td>
<td>6.64386</td>
<td>#MISSING</td>
<td>6.90689</td>
<td>. . .</td>
<td>#MISSING</td>
</tr>
</tbody>
</table>

See Also
- @LN
- @LOG10

@LOG10

Returns the base-10 logarithm of the specified expression.

Syntax

@LOG10 (expression)

Parameters

**expression**
Single member specification, variable name, function, or other numeric expression. If less than or equal to 0, Essbase returns #MISSING.
Example

The following example is based on a variation of Sample Basic:

\[
\text{LOG10\_Sales} = \text{@LOG10(Sales)};
\]

This example produces the following result:

<table>
<thead>
<tr>
<th>Product</th>
<th>East</th>
<th>West</th>
<th>South</th>
<th>Central</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sales</td>
<td>87398</td>
<td>132931</td>
<td>50846</td>
<td>129680</td>
</tr>
<tr>
<td>LOG10_Sales</td>
<td>4.94150</td>
<td>5.12363</td>
<td>4.70626</td>
<td>5.11287</td>
</tr>
</tbody>
</table>

See Also

- \[@LOG\]
- \[@LN\]

@LSIBLINGS

Returns the left siblings of the specified member.

Syntax

\[@LSIBLINGS(mbrName)\]

Parameters

mbrName

Any valid single member name, or a function that returns a single member.

Notes

This function returns the left siblings of the specified member. Left siblings are children that share the same parent as the member and that precede the member in the database outline. This function excludes the specified member.

This member set function can be used as a parameter of another function, where that parameter is a list of members.

Essbase sorts the generated list of left siblings in descending order. Using Sample Basic as an example, if you specify 200-30 for mbrName, Essbase returns 200-20, 200-10 (in that order). This order is important to consider when you use this function with certain forecasting and statistical functions.

Example

In the Sample Basic database:

\[@LSIBLINGS(Qtr4)\]
Returns Qtr3, Qtr2, and Qtr1 (in that order). These members appear above Qtr4 in the Sample Basic outline.

@LSIBLINGS(Utah)

Returns Washington, Oregon, and California (in that order). These members appear above Utah in the Sample Basic outline.

See Also
• @ILSIBLINGS
• @NEXTSIBLING
• @PREVSIBLING
• @RSIBLINGS
• @SHIFTSIBLING

@MATCH

Performs wildcard member selections.

Syntax

@MATCH (mbrName|genName|levName, "pattern")

Parameters

mbrName
The default or user-defined name of the member on which to base the search. Essbase searches the member names and alias names of the specified member and its descendants.

genName
The default or user-defined name of the generation to search. Essbase searches all member names and member alias names in the generation.

levName
The default or user-defined name of the level to search. Essbase searches all member names and member alias names in the level.

"pattern"
The character pattern to search for, including a wildcard character (* or ?). 
? substitutes one occurrence of any character. You can use ? anywhere in the pattern.
* substitutes any number of characters. You can use * only at the end of the pattern.
To include spaces in the character pattern, enclose the pattern in double quotation marks (" ").

Notes
This function performs a trailing-wildcard member selection. Essbase searches for member names and alias names that match the pattern you specify and returns the member and alias names it finds.
If the members names in the database you are searching are case-sensitive, the search is case-sensitive. Otherwise, the search is not case-sensitive.

You can call @MATCH more than once in a calculation script.

If Essbase does not find any members that match the chosen character pattern, it returns no member names and continues with the other calculation commands in the calculation script.

**Example**

In the Sample Basic database:

@MATCH(Product,"???-10")

Returns 100-10, 200-10, 300-10, and 400-10

@MATCH(Year,"J*")

Returns Jan, Jun, Jul

@MATCH(Product,"C*")

Returns 100 (Colas), 100-10 (Cola), 100-30 (Caffeine Free Cola), 300 (Cream Soda)

[@MAX](#)

Returns the maximum value among the results of the expressions in the specified member list.

**Syntax**

@MAX (expList)

**Parameters**

expList  
Comma-delimited list of members, variable names, functions, and numeric expressions, all of which return numeric values.

**Notes**

Depending on the values in the list, this function may return zero(0) or #MISSING. For full control over skipping or inclusion of empty values, use @MAXS instead.

**Example**

This example is based on the Sample Basic database:

Qtr1 = @MAX(Jan:Mar);
This example produces the following report:

<table>
<thead>
<tr>
<th>Colas</th>
<th>New York</th>
<th>Actual</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jan</td>
<td>Feb</td>
<td>Mar</td>
</tr>
<tr>
<td>===</td>
<td>===</td>
<td>===</td>
</tr>
<tr>
<td>Sales</td>
<td>678</td>
<td>645</td>
</tr>
</tbody>
</table>

See Also

- @MAXS
- @MAXSRANGE
- @MINS

@MAXRANGE

Returns the maximum value of the specified member across the specified range of members.

Syntax

@MAXRANGE (mbrName [ ,XrangeList])

Parameters

mbrName
Any valid single member name, or a function that returns a single member.

XrangeList
Optional. A valid member name, a comma-delimited list of member names, cross dimension members, or a member set function or range function (including @XRANGE) that returns a list of members from the same dimension. If XrangeList is not specified, Essbase uses the level 0 members from the dimension tagged as Time.

Notes

Depending on the values in the list, @MAXRANGE may return a zero(0) or #MISSING value. For full control over skipping or inclusion of empty values, use @MAXSRANGE instead.

Example

In the Sample Basic database:

Qtr1 = @MAXRANGE(Sales,@CHILDREN(Qtr1));

produces the following report:

<table>
<thead>
<tr>
<th>Colas</th>
<th>New York</th>
<th>Actual</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jan</td>
<td>Feb</td>
<td>Mar</td>
</tr>
<tr>
<td>===</td>
<td>===</td>
<td>===</td>
</tr>
<tr>
<td>Sales</td>
<td>678</td>
<td>645</td>
</tr>
</tbody>
</table>
See Also

- @MAXS
- @MAXSRANGE
- @MINSRANGE

@MAXS

Returns the maximum value among the results of the expressions in the specified member list, with options to skip missing or zero values (in contrast with @MAX, which cannot ignore empty values).

Syntax

@MAXS (SKIPNONE | SKIPMISSING | SKIPZERO | SKIPBOTH, expList)

Parameters

- **SKIPNONE**
  Includes all cells specified in expList in the operation, regardless of their content

- **SKIPMISSING**
  Ignores all #MISSING values

- **SKIPZERO**
  Ignores all 0 values

- **SKIPBOTH**
  Ignores all 0 and #MISSING values

expList

Comma-delimited list of members, variable names, functions, or numeric expressions, all of which return numeric values

Notes

- @MAXS (SKIPMISSING, expList) is equivalent to @MAX (expList).
- Because #MISSING values are greater than negative data values and less than positive data values, if the data being calculated includes only negative and #MISSING values, @MAXS returns #MISSING.
- If the data being calculated includes only negative, 0, and #MISSING values, @MAXS may return either #MISSING or 0 values in an unpredictable manner.

Example

For both examples, assume a database similar to Sample Basic. The Measures dimension includes two members: COGS (cost of goods sold) and OtherInc_Exp (miscellaneous income and expenses). The data can include 0 and #MISSING values.

Example 1

Qtr1_Max = @MAXS(SKIPBOTH, Jan:Mar);
This example ignores #MISSING and 0 values for all members of the Measures dimension. This example produces the following results:

<table>
<thead>
<tr>
<th></th>
<th>Jan</th>
<th>Feb</th>
<th>Mar</th>
<th>Qtr1_Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>COGS</td>
<td>#MISSING</td>
<td>1500</td>
<td>2300</td>
<td>2300</td>
</tr>
<tr>
<td>OtherInc_Exp</td>
<td>-500</td>
<td>-350</td>
<td>0</td>
<td>-350</td>
</tr>
</tbody>
</table>

**Example 2**

Qtr1_Max = @MAXS(SKIPNONE, Jan:Mar);

This example includes #MISSING and 0 values in the calculation, for all members of the Measures dimension. This example produces the following results:

<table>
<thead>
<tr>
<th></th>
<th>Jan</th>
<th>Feb</th>
<th>Mar</th>
<th>Qtr1_Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>COGS</td>
<td>#MISSING</td>
<td>1500</td>
<td>2300</td>
<td>2300</td>
</tr>
<tr>
<td>OtherInc_Exp</td>
<td>-500</td>
<td>-350</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**See Also**

- @MAX
- @MAXSRANGE
- @MINS

**@MAXSRANGE**

Returns the maximum value of the specified member across the specified range of members, with options to skip missing or zero values (in contrast with @MAXRANGE, which cannot ignore empty values).

**Syntax**

@MAXSRANGE (SKIPNONE | SKIPMISSING | SKIPZERO | SKIPBOTH, mbrName [,XrangeList])

**Parameters**

- **SKIPNONE**
  Includes all cells specified in expList in the operation, regardless of their content

- **SKIPMISSING**
  Ignores all #MISSING values

- **SKIPZERO**
  Ignores all 0 values

- **SKIPBOTH**
  Ignores all 0 and #MISSING values
mbrName
Any valid single member name, or a function that returns a single member.

XrangeList
Optional. A valid member name, a comma-delimited list of member names, cross dimension members, or a member set function or range function (including @XRANGE) that returns a list of members from the same dimension. If XrangeList is not specified, Essbase uses the level 0 members from the dimension tagged as Time.

Notes
- @MAXSRANGE (SKIPNONE, mbrName, XrangeList) is equivalent to @MAXRANGE mbrName, (XrangeList).
- #MISSING values are considered to be greater than negative data values and less than positive data values. If the data being calculated includes only negative and #MISSING values, @MAXSRANGE returns #MISSING.
- For all members, @MAXSRANGE returns the value calculated for the specified member and range list.

Example
For both examples, assume a database similar to Sample Basic. The Measures dimension includes two members: COGS (cost of goods sold) and OtherInc_Exp (miscellaneous income and expenses). The data can include 0 and #MISSING values. For both members of the Measures dimension, the result is the same--the maximum value for the OtherInc_Exp member across the specified range.

Example 1
Qtr1_Max = @MAXSRANGE (SKIPBOTH, OtherInc_Exp, @CHILDREN(Qtr1));

This example ignores #MISSING and 0 values and produces the following results:

<table>
<thead>
<tr>
<th></th>
<th>Jan</th>
<th>Feb</th>
<th>Mar</th>
<th>Qtr1_Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>OtherInc_Exp</td>
<td>-500</td>
<td>#MISSING</td>
<td>-250</td>
<td>-250</td>
</tr>
<tr>
<td>COGS</td>
<td>0</td>
<td>1500</td>
<td>2300</td>
<td>-250</td>
</tr>
</tbody>
</table>

Example 2
Qtr1_Max = @MAXSRANGE (SKIPNONE, OtherInc_Exp, @CHILDREN(Qtr1));

Using the same data as Example 1, Example 2 demonstrates what happens if you do not skip 0 and #MISSING values in the data. Example 2 produces the following report:

<table>
<thead>
<tr>
<th></th>
<th>Jan</th>
<th>Feb</th>
<th>Mar</th>
<th>Qtr1_Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>OtherInc_Exp</td>
<td>-500</td>
<td>#MISSING</td>
<td>-250</td>
<td>#MISSING</td>
</tr>
<tr>
<td>COGS</td>
<td>0</td>
<td>1500</td>
<td>2300</td>
<td>#MISSING</td>
</tr>
</tbody>
</table>
See Also

- @MAXS
- @MINSRANGE
- @MAXRANGE

@MBRCOMPARE

Returns a member set of member names that match the comparison criteria. Member names are evaluated alpha-numerically.

This function can be used on unique and duplicate-name outlines.

Syntax

@MBRCOMPARE (compOperator, tokenString, topMbrinHierarchy)

Parameters

compOperator
One of the following strings: < (less than), <= (less than or equal to), > (greater than), >= (greater than or equal to), == (equals), != (not equal to), or CDF (for a custom-defined function).

Note:
Using the == (equal to) comparison operator is the same as using @EQUAL. Using the != (not equal to) comparison operator is the same as using @NOTEQUAL.

tokenString
Token string value with which to compare to members in the outline, starting with the member specified in topMbrinHierarchy.

topMbrinHierarchy
A fully qualified name of a member in the outline on which to base the member search. The specified member and its aliases, and all of its descendants, are included in the search.

Note:
Although aliases of the specified member are included in the search, only outline member names (not aliases) are used when comparing member names.

To search the entire outline, provide an empty string (""") for this parameter. For example, @MBRCOMPARE("<=" , "100-10", "").
Example

The following examples are based on the following duplicate-name outline:

Product
  100
    100-10
      100-10-10
    100-20
    100-30
  200
    200-10
    200-20
    200-30
  300
    300-10
    300-20
Diet
  100-10
    100-10-11
  200-10
  300-10
Bottle
  200-10
  300-20

@MBRCOMPARE("<=" , "100-10", "Product")

| Returns the members 100, [100].[100-10], and [Diet].[100-10]. |

@MBRCOMPARE("==", "100-10", "Product")

| Returns the members [Diet].[100-10] and [100].[100-10]. |

See Also

- @BETWEEN
- @EQUAL
- @EXPAND
- @LIKE
- @MBRPARENT
- @NOTEQUAL

@MBRPARENT

Returns the parent of the specified member.

This function can be used on unique and duplicate-name outlines.
Syntax

@MBRPARENT (mbrName)

Parameters

mbrName
Name of a member in the outline.

Example

The following examples are based on the following duplicate-name outline:

Product
  100
    100-10
    100-10-10
    100-20
    100-30
  200
    200-10
    200-20
    200-30
  300
    300-10
    300-20
Diet
  100-10
    100-10-11
  200-10
  300-10
Bottle
  200-10
  300-20

@MBRPARENT ("100-10", "Product")

Returns the member 100.

@MBRPARENT ("100-10-11")

Returns the member [Diet].[100-10].

See Also

• @BETWEEN
• @EQUAL
• @EXPAND
• @LIKE
@MDALLOCATE

Allocates values from a member, from a cross-dimensional member, or from a value across multiple dimensions. The allocation is based on a variety of criteria.

This function allocates values that are input at an upper level to lower-level members in multiple dimensions. The allocation is based upon a specified share or spread of another variable. You can specify a rounding parameter for allocated values and account for rounding errors.

Syntax

@MDALLOCATE (amount, Ndim, allocationRange1 ... allocationRangeN, basisMbr, [roundMbr], method [, methodParams]
 [, round [, numDigits][, roundErr]])

Parameters

amount
A value, member, or cross-dimensional member that contains the value to be allocated into each allocationRange. The value may also be a constant.

• If amount is a member, the member must be from a dimension to which an allocationRange belongs.

• If amount is a cross-dimensional member, the member must include a member from every dimension of every allocationRange.

• If a member or cross-dimensional member is not from an allocationRange dimension, Essbase displays a warning message.

If the amount parameter is a loaded value, it cannot be a Dynamic Calc member.

Ndim
The number of dimensions across which values are allocated.

allocationRange1 ... allocationRangeN
Comma-delimited lists of members, member set functions, or range functions from the multiple dimensions into which values from amount are allocated.

basisMbr
A value, member, or cross-dimensional member that contains the values that are used as the basis for the allocation. The method you specify determines how the basis data is used.

roundMbr
Optional. The member or cross-dimensional member to which rounding errors are added. This member (or at least one member of a cross-dimensional member) must be included in an allocationRange.

method
The expression that determines how values are allocated. One of the following:
• **share**: Uses `basisMbr` to calculate a percentage share. The percentage share is calculated by dividing the value in `basisMbr` for the current member in `allocationRange->` by the sum across the `allocationRange` for that basis member:

\[
\text{amount} \times (\text{CURRMBR}() - > \text{basisMbr} / \text{SUM}(\text{allocationRange} - > \text{basisMbr}))
\]

• **spread**: Spreads `amount` across `allocationRange`:

\[
\text{amount} \times (1/\text{COUNT}(\text{SKIP, allocationRange}))
\]

• **SKIPNONE | SKIPMISSING | SKIPZERO | SKIPBOTH**: Values to be ignored during calculation of the spread. You must specify a SKIP parameter only for `spread`.
  - **SKIPNONE**: Includes all cells.
  - **SKIPMISSING**: Excludes all #MISSING values in `basisMbr`, and stores #MISSING for values in `allocationRange` for which the `basisMbr` is missing.
  - **SKIPZERO**: Excludes all zero (0) values in `basisMbr`, and stores #MISSING for values in `allocationRange` for which the `basisMbr` is zero.
  - **SKIPBOTH**: Excludes all zero (0) values and all #MISSING values, and stores #MISSING for values in `allocationRange` for which the `basisMbr` is zero (0) or #MISSING.

• **percent**: Takes a percentage value from `basisMbr` for each member in `allocationRange` and applies the percentage value to `amount`:

\[
\text{amount} \times (\text{CURRMBR}() - > \text{basisMbr} \times .01).
\]

• **add**: Takes the value from `basisMbr` for each member of `allocationRange` and adds the value to `amount`:

\[
\text{amount} + \text{CURRMBR}() - > \text{basisMbr}
\]

• **subtract**: Takes the value from `basisMbr` for each member of `allocationRange` and subtracts the value from `amount`:

\[
\text{amount} - \text{CURRMBR}() - > \text{basisMbr}
\]

• **multiply**: Takes the value from `basisMbr` for each member of `allocationRange` and multiplies the value by `amount`:

\[
\text{amount} \times \text{CURRMBR}() - > \text{basisMbr}
\]

• **divide**: Takes the value from `basisMbr` for each member of `allocationRange` and divides the value by `amount`:

\[
\text{amount}/\text{CURRMBR}() - > \text{basisMbr}
\]

**round**

Optional. One of the following:
• **noRound**: No rounding. This is the default.

• **roundAmt**: Indicates that you want to round the allocated values. If you specify `roundAmt`, you also must specify `numDigits` to indicate the number of decimal places to round to.

**numDigits**

An integer that represents the number of decimal places to round to. You must specify `numDigits` if you specify `roundAmt`.

- If `numDigits` is 0, the allocated values are rounded to the nearest integer. The default value for `numDigits` is 0.
- If `numDigits` is greater than 0, the allocated values are rounded to the specified number of decimal places.
- If `numDigits` is a negative value, the allocated values are rounded to a power of 10.

If you specify `roundAmt`, you also can specify a `roundErr` parameter.

**roundErr**

Optional. An expression that specifies where rounding errors should be placed. You must specify `roundAmt` in order to specify `roundErr`. If you do not specify `roundErr`, Essbase discards rounding errors.

To specify `roundErr`, choose from one of the following:

- **errorsToHigh**: Adds rounding errors to the member with the highest allocated value. If allocated values are identical, adds rounding errors to the first value in `allocationRange`.
- **errorsToLow**: Adds rounding errors to the member with the lowest allocated value. If allocated values are identical, adds rounding errors to the first value in `allocationRange`. `#MISSING` is treated as the lowest value in a list; if multiple values are `#MISSING`, rounding errors are added to the first `#MISSING` value in the list.
- **errorsToMbr**: Adds rounding errors to the specified `roundMbr`, which must be included in `allocationRange`.

**Notes**

- When you use this function in a calculation script, use it within a `FIX` statement; for example, `FIX` on the member to which the allocation amount is loaded. Although `FIX` is not required, using it may decrease calculation time.
- For a more complex example using `@MDALLOCATE`, see Allocating Values Across Multiple Dimensions in *Designing and Maintaining Essbase Cubes*.
- If you have very large `allocationRange` lists, Essbase may return error messages during the calculation.

**Example**

Consider the following example from the Sample Basic database. A data value of 500 is loaded to Budget->Total Expenses->East for Jan and Colas. (For this example, assume that Total Expenses is not a Dynamic Calc member.)
You need to allocate the amount across each expense category for each child of East. The allocation for each child of East is based on the child's share of Total Expenses-

>Actual:

```plaintext
FIX("Total Expenses")
Budget = @MDALLOCATE(Budget->"Total Expenses"->East,2,
   @CHILDREN(East),@CHILDREN("Total Expenses"),Actual,,share);
ENDFIX
```

This example produces the following report:

<table>
<thead>
<tr>
<th></th>
<th>Marketing</th>
<th>Jan Payroll</th>
<th>Colas Misc</th>
<th>Total Expenses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actual</td>
<td>New York</td>
<td>94</td>
<td>51</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Massachusetts</td>
<td>23</td>
<td>31</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Florida</td>
<td>53</td>
<td>54</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Connecticut</td>
<td>40</td>
<td>31</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>New Hampshire</td>
<td>27</td>
<td>53</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>East</td>
<td>237</td>
<td>220</td>
<td>3</td>
</tr>
<tr>
<td>Budget</td>
<td>New York</td>
<td>102.174</td>
<td>55.435</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Massachusetts</td>
<td>25</td>
<td>33.696</td>
<td>1.087</td>
</tr>
<tr>
<td></td>
<td>Florida</td>
<td>57.609</td>
<td>58.696</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Connecticut</td>
<td>43.478</td>
<td>33.696</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>New Hampshire</td>
<td>29.348</td>
<td>57.609</td>
<td>2.173</td>
</tr>
<tr>
<td></td>
<td>East</td>
<td>#MI</td>
<td>#MI</td>
<td>#MI</td>
</tr>
</tbody>
</table>

See Also

@ALLOCATE

@MDANCESTVAL

Returns ancestor-level data from multiple dimensions based on the current member being calculated.

Syntax

```plaintext
@MDANCESTVAL (dimCount, dimName1, genLevNum1 . . . dimNameX, genLevNumX [,mbrName])
```

Parameters

`dimCount`

Integer value that defines the number of dimensions from which ancestor values are being returned.

`dimName1, . . . dimNameX`

Defines the dimension names from which the ancestor values are to be returned. You must specify a `genLevNum` for every `dimName`.
**genLevNum, ... genLevNumX**

Integer value that defines the absolute generation or level number from which the ancestor values are to be returned. A positive integer defines a generation reference. A negative number or value of 0 defines a level reference. You must specify a `dimName` for every `genLevNum`.

To use this function or any other ancestor value function in a ragged hierarchy, use generation references instead of level references to avoid unexpected results.

**mbrName**

Optional. Any valid single member name, or a function that returns a single member.

**Example**

Marketing expenses are captured at the Product Family and Region level in a product planning application. The Marketing Expense data must be allocated down to each Product code and State level based on Sales contribution. Data is captured as follows:

<table>
<thead>
<tr>
<th></th>
<th>Sales</th>
<th>Marketing</th>
</tr>
</thead>
<tbody>
<tr>
<td>New York</td>
<td>100-10</td>
<td>300</td>
</tr>
<tr>
<td></td>
<td>100-20</td>
<td>200</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>500</td>
</tr>
<tr>
<td>Boston</td>
<td>100-10</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>100-20</td>
<td>400</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>500</td>
</tr>
<tr>
<td>East</td>
<td>100-10</td>
<td>400</td>
</tr>
<tr>
<td></td>
<td>100-20</td>
<td>600</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>1000</td>
</tr>
</tbody>
</table>

The Marketing Expense value of 200 at East and Product code 100 is allocated down to each Product code and State with the following formula:

```
Marketing = (Sales / @MDANCESTVAL(2, Market, 2, Product, 2, Sales)) * @MDANCESTVAL(2, Market, 2, Product, 2, Marketing);
```

which produces the following result:

<table>
<thead>
<tr>
<th></th>
<th>Sales</th>
<th>Marketing</th>
</tr>
</thead>
<tbody>
<tr>
<td>New York</td>
<td>100-10</td>
<td>300</td>
</tr>
<tr>
<td></td>
<td>100-20</td>
<td>200</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>500</td>
</tr>
<tr>
<td>Boston</td>
<td>100-10</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>100-20</td>
<td>400</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>500</td>
</tr>
<tr>
<td>East</td>
<td>100-10</td>
<td>400</td>
</tr>
<tr>
<td></td>
<td>100-20</td>
<td>600</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>1000</td>
</tr>
</tbody>
</table>

The Marketing expenses can then be reconsolidated across Products and Markets.

**See Also**

- @ANCESTVAL
@MDPARENTVAL

Returns parent-level data from multiple dimensions based on the current member being calculated.

Syntax

@MDPARENTVAL (numDim, dimName1, . . . dimNameX [,mbrName])

Parameters

numDim
Integer value that defines the number of dimensions from which parent values are being returned.

dimName1, . . . dimNameX
Defines the dimension names from which the parent values are to be returned.

mbrName
Optional. Any valid single member name, or a function that returns a single member.

Example

Marketing expenses are captured at the Product Family and Region level in a product planning application. The Marketing Expense data must be allocated down to each Product code and State level based on Sales contribution.

Data is captured as follows:

<table>
<thead>
<tr>
<th></th>
<th>Sales</th>
<th>Marketing</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>======</td>
<td>==========</td>
</tr>
<tr>
<td>New York</td>
<td>100-10</td>
<td>300</td>
</tr>
<tr>
<td></td>
<td>100-20</td>
<td>200</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>500</td>
</tr>
<tr>
<td>Boston</td>
<td>100-10</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>100-20</td>
<td>400</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>500</td>
</tr>
<tr>
<td>East</td>
<td>100-10</td>
<td>400</td>
</tr>
<tr>
<td></td>
<td>100-20</td>
<td>600</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>1000</td>
</tr>
</tbody>
</table>

The Marketing Expense value of 200 at East and Product code 100 is allocated down to each Product code and State with the following formula:

\[
\text{Marketing} = (\text{Sales} / \text{@MDPARENTVAL}(2, \text{Market, Product, Sales})) \times \text{@MDPARENTVAL}(2, \text{Market, Product, Marketing})
\]
which produces the following result:

<table>
<thead>
<tr>
<th></th>
<th>Sales</th>
<th>Marketing</th>
</tr>
</thead>
<tbody>
<tr>
<td>New York</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100-10</td>
<td>300</td>
<td>60</td>
</tr>
<tr>
<td>100-20</td>
<td>200</td>
<td>40</td>
</tr>
<tr>
<td>100</td>
<td>500</td>
<td>N/A</td>
</tr>
<tr>
<td>Boston</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100-10</td>
<td>100</td>
<td>20</td>
</tr>
<tr>
<td>100-20</td>
<td>400</td>
<td>80</td>
</tr>
<tr>
<td>100</td>
<td>500</td>
<td>N/A</td>
</tr>
<tr>
<td>East</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100-10</td>
<td>400</td>
<td>N/A</td>
</tr>
<tr>
<td>100-20</td>
<td>600</td>
<td>N/A</td>
</tr>
<tr>
<td>100</td>
<td>1000</td>
<td>N/A</td>
</tr>
</tbody>
</table>

The Marketing expenses can then be reconsolidated across Products and Markets.

See Also

- @PARENTVAL
- @MDANCESTVAL
- @SPARENTVAL

@MDSHIFT

Shifts a series of data values across multiple dimension ranges.

Syntax

@MDSHIFT (mbrName, shiftCnt1, dimName1, [range1|(range1)], . . . shiftCntX, dimNameX, [rangeX|(rangeX)])

Parameters

mbrName
Any valid single member name, or a function that returns a single member.

shiftCnt1...shiftCntX
Integer that defines the number of member positions to shift.

dimName1, . . . dimNameX
Defines the dimension names in which the shift is to occur.

range1|(range1) . . . rangeX|(rangeX)
Optional. A valid member name, a comma-delimited list of member names, member set functions, and range functions. If rangeList is not specified, Essbase uses the level 0 members from the dimension specified with the dimName parameter. If the range list is comma delimited, then the list must be enclosed in parentheses.
Example

The Budget figures for Ending Inventory need to be calculated by taking Prior Year-Opening Inventory results as a starting point:

<table>
<thead>
<tr>
<th></th>
<th>Jan</th>
<th>Feb</th>
<th>Mar</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prior Year</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Opening Inventory</td>
<td>110</td>
<td>120</td>
<td>130</td>
</tr>
<tr>
<td>Budget</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Ending Inventory</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

The following calculation script assumes that the Scenario dimension is as follows:

```
Scenario
    Prior Year
    Budget

FIX (Budget)
"Ending Inventory" = @MDSHIFT("Opening Inventory", 1, Year, , -1, Scenario,);
ENDFIX
```

In this example, range1 is not specified, so Essbase defaults to the level 0 members of the Year dimension, which was specified as the `dimName1` parameter. Since range2 is also not specified, Essbase defaults to the level 0 members of the Scenario dimension, which was specified as the `dimName2` parameter. This example produces the following result:

<table>
<thead>
<tr>
<th></th>
<th>Jan</th>
<th>Feb</th>
<th>Mar</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prior Year</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Opening Inventory</td>
<td>110</td>
<td>120</td>
<td>130</td>
</tr>
<tr>
<td>Budget</td>
<td>120</td>
<td>130</td>
<td>140</td>
</tr>
<tr>
<td>Ending Inventory</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

See Also

@SHIFT

@MEDIAN

Returns the median (the middle number) of the specified data set. Half the numbers in the data set are larger than the median, and half are smaller.

Syntax

@MEDIAN (SKIPNONE | SKIPMISSING | SKIPZERO | SKIPBOTH, XrangeList)
Parameters

**SKIPNONE**
Includes all cells specified in the data set, regardless of their content, during calculation of the median.

**SKIPMISSING**
Excludes all #MISSING values from the data set during calculation of the median.

**SKIPZERO**
Excludes all zero (0) values from the data set during calculation of the median.

**SKIPBOTH**
Excludes all zero (0) values and #MISSING values from the data set during calculation of the median.

**XrangeList**
A list of numeric values across which the median is calculated. Referred to generically throughout this topic as “the data set.”
Can be a valid member name, a comma-delimited list of member names, cross dimensional members, or a return value from a member set function or range function (including @XRANGE).
For more information about XrangeList, see Range List Parameters.

Notes

- If the member you are calculating and the data set (XrangeList) are not in the same dimension, use @RANGE or @XRANGE to cross the member with the list of members (for example, to cross Sales with the children of 100).

- @MEDIAN sorts the data set in ascending order before calculating the median.

- When the data set contains an even number of values, @MEDIAN calculates the average of the two middle numbers.

- @MEDIAN treats #MISSING values as 0 unless SKIPMISSING or SKIPBOTH is specified.

- When you use this function in a calculation script, use it within a FIX statement. Although FIX is not required, using it may improve calculation performance.

- When you use this function across a large range in a sparse dimension, you may need to increase the size of the calculator cache.

Example

The following example is based on the Sample Basic database. Assume that the Measures dimension contains an additional member, Median. This example calculates the median sales values for all products and uses @RANGE to generate the data set:

```
FIX (Product)
Median = @MEDIAN(SKIPBOTH, @RANGE(Sales, @CHILDREN(Product)));
ENDFIX
```
This example produces the following report:

<table>
<thead>
<tr>
<th>Product</th>
<th>Actual</th>
<th>Budget</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sales Colas</td>
<td>678</td>
<td>640</td>
</tr>
<tr>
<td>Root Beer</td>
<td>551</td>
<td>530</td>
</tr>
<tr>
<td>Cream Soda</td>
<td>663</td>
<td>510</td>
</tr>
<tr>
<td>Fruit Soda</td>
<td>587</td>
<td>620</td>
</tr>
<tr>
<td>Diet Drinks</td>
<td>#MI</td>
<td>#MI</td>
</tr>
<tr>
<td>Product</td>
<td>2479</td>
<td>2300</td>
</tr>
</tbody>
</table>

| Median Product | 625 | 575 |

Because SKIPBOTH is specified in the calculation script, the #MI values for Diet Drinks are skipped. The remaining four products create an even-numbered data set. So, to calculate Median->Product->Actual, the two middle numbers in the set (587 and 663) are averaged to create the median (625). To calculate Median->Product->Budget, the two middle numbers in the set (530 and 620) are averaged to create the median (575).

The following example assumes a Year dimension is added to Sample Basic. It calculates median using cross-dimensional members in the data set.

```plaintext
FIX(Product)
Median = @MEDIAN(@XRANGE("2011"->"Sep", "2012"->"Mar"));
ENDFIX
```

The above calculation is performed across the following multidimensional range specified by `XrangeList`:

- 2011->Sep
- 2011->Oct
- 2011->Nov
- 2011->Dec
- 2012->Jan
- 2012->Feb
- 2012->Mar

See Also
- @RANGE
- @XRANGE

@MEMBER

Returns the member with the name that is provided as a character string.

Syntax

```plaintext
@MEMBER (String)
```
**Parameters**

**String**
A string (enclosed in double quotation marks) or a function that returns a string

**Example**
Typically, `@MEMBER` is used in combination with string functions that are used to manipulate character strings to form the name of a member. In the following example, the member name `QTR1` is appended to the character string `2000_` to form the string `2000_QTR1`. `@MEMBER` returns the member `2000_QTR1` and `QTD` is set to the value of this member.

```plaintext
QTD=@MEMBER(@CONCATENATE("2000_", QTR1));
```

**See Also**
- `@CONCATENATE`
- `@SUBSTRING`

**@MEMBERAT**

Returns the specified member in a list of members.

**Syntax**

```plaintext
@MEMBERAT(mbrList, mbrIndex)
```

**Parameters**

**mbrList**
Member list or function that returns a member list.

**mbrIndex**
Nonzero integer. If positive, enumerates from start of the list (for example, 1 returns the first member in the list). If negative, enumerates from the end of the list (for example, -1 returns the last member in the list).

**Notes**
If `mbrIndex` is 0 or out of bounds, the script or member formula fails during verification or runtime and returns an error.

**Example**
The following examples use the Sample.Basic database.

```plaintext
@MEMBERAT(@CHILDREN("Colas"), 2); returns 100-20 (Diet Cola).

Sales = @MEMBERAT(@CHILDREN("Total Expenses"), -1);
```
The value of the member Misc is assigned to Sales, because Misc is the last child of Total Expenses, and the mbrIndex of -1 causes this function to select the last member in the list.

@MEMBERAT(@CHILDREN("100-10"), 1);

Because @CHILDREN("100-10") is an empty list, returns an error.

See Also
@MEMBER

@MERGE

Merges two member lists that are processed by another function. Duplicates (values found in both lists) are included only once in the merged list.

Syntax

@MERGE (list1, list2)

Parameters

list1
The first list of member specifications to be merged.

list2
The second list of member specifications to be merged.

Notes

• Duplicate values are included only once in the merged list.
• @MERGE can merge only two lists at a time. You can nest @MERGE function calls to merge more than two lists.

Example

Example 1

In the Sample Basic database,

@MERGE(@CHILDREN(Colas),@CHILDREN("Diet Drinks"));

returns Cola, Diet Cola, Caffeine Free Cola, Diet Root Beer, and Diet Cream Soda.

Diet Cola appears only once in the merged list, even though it is a child of both Colas and Diet Drinks.
Example 2

In this example, @MERGE is used with @ISMBR to increase the marketing budget for major markets and for western markets.

Budget
(IF (@ISMBR(@MERGE(@UDA(Market,"Major Market"),
    @DESCENDANTS(West))))
Marketing = Marketing * 1.1;
ENDIF;);

This example produces the following report, which shows only the major markets in the East and all western markets:

<table>
<thead>
<tr>
<th>Product</th>
<th>Year</th>
<th>Budget</th>
</tr>
</thead>
<tbody>
<tr>
<td>New York</td>
<td>6039</td>
<td></td>
</tr>
<tr>
<td>Massachusetts</td>
<td>1276</td>
<td></td>
</tr>
<tr>
<td>Florida</td>
<td>2530</td>
<td></td>
</tr>
<tr>
<td>California</td>
<td>7260</td>
<td></td>
</tr>
<tr>
<td>Oregon</td>
<td>2090</td>
<td></td>
</tr>
<tr>
<td>Washington</td>
<td>2772</td>
<td></td>
</tr>
<tr>
<td>Utah</td>
<td>1837</td>
<td></td>
</tr>
<tr>
<td>Nevada</td>
<td>4521</td>
<td></td>
</tr>
</tbody>
</table>

The values prior to running the calculation script were:

<table>
<thead>
<tr>
<th>Product</th>
<th>Year</th>
<th>Budget</th>
</tr>
</thead>
<tbody>
<tr>
<td>New York</td>
<td>5490</td>
<td></td>
</tr>
<tr>
<td>Massachusetts</td>
<td>1160</td>
<td></td>
</tr>
<tr>
<td>Florida</td>
<td>2300</td>
<td></td>
</tr>
<tr>
<td>California</td>
<td>6600</td>
<td></td>
</tr>
<tr>
<td>Oregon</td>
<td>1900</td>
<td></td>
</tr>
<tr>
<td>Washington</td>
<td>2520</td>
<td></td>
</tr>
<tr>
<td>Utah</td>
<td>1670</td>
<td></td>
</tr>
<tr>
<td>Nevada</td>
<td>4110</td>
<td></td>
</tr>
</tbody>
</table>

See Also

- @INTERSECT
- @LIST
- @RANGE
- @REMOVE

@MIN

Returns the minimum value among the results of the expressions in expList.
Syntax

@MIN (expList)

Parameters

expList
Comma-delimited list of members, variable names, functions, and numeric expressions, all of which return numeric values.

Notes

Depending on the values in the list, @MIN may return a zero(0) or #MISSING value. For full control over skipping or inclusion of empty values, use @MINS.

Example

In the Sample Basic database:

Qtr1 = @MIN(Jan:Mar);

produces the following report:

<table>
<thead>
<tr>
<th></th>
<th>Colas</th>
<th>New York</th>
<th>Actual</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jan</td>
<td>===</td>
<td>===</td>
<td>Qtr1</td>
</tr>
<tr>
<td>Feb</td>
<td>678</td>
<td>645</td>
<td>675</td>
</tr>
<tr>
<td>Mar</td>
<td>675</td>
<td>645</td>
<td>645</td>
</tr>
</tbody>
</table>

See Also

• @MAX
• @MINS
• @MINRANGE

@MINRANGE

Returns the minimum value of mbrName across XrangeList.

Syntax

@MINRANGE (mbrName [,XrangeList])

Parameters

mbrName
Any valid single member name, or a function that returns a single member.

XrangeList
Optional. A valid member name, a comma-delimited list of member names, cross dimension members, or a member set function or range function (including
@XRANGE) that returns a list of members from the same dimension. If XrangeList is not specified, Essbase uses the level 0 members from the dimension tagged as Time.

Notes

Depending on the values in the list, this function may return a zero(0) or #MISSING value. For full control over skipping or inclusion of empty values, use @MINSRANGE.

Example

In the Sample Basic database:

```
Qtr1 = @MINRANGE(Sales,Jan:Mar);
```

produces the following report:

<table>
<thead>
<tr>
<th></th>
<th>Cola</th>
<th>New York</th>
<th>Actual</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jan</td>
<td>678</td>
<td>645</td>
<td>675</td>
</tr>
<tr>
<td>Feb</td>
<td>645</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mar</td>
<td>675</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Qtr1</td>
<td></td>
<td>645</td>
<td></td>
</tr>
</tbody>
</table>

See Also

- @MAXSRANGE
- @MINSRANGE
- @MIN

@MINS

Returns the minimum value across the results of the expressions in expList, with options to skip missing or zero values.

Syntax

```
@MINS (SKIPNONE | SKIPMISSING | SKIPZERO | SKIPBOTH, expList)
```

Parameters

**SKIPNONE**
Includes in the operation all cells specified in expList regardless of their content.

**SKIPMISSING**
Ignores all #MISSING values.

**SKIPZERO**
Ignores all 0 values.

**SKIPBOTH**
Ignores all 0 and #MISSING values.
expList
Comma-delimited list of member names, variable names, functions, or numeric expressions. expList provides a list of numeric values for which Essbase determines the minimum value.

Notes
• This function enables skipping of #MISSING and 0 values, in contrast with @MIN, which always includes empty values.
• @MINS (SKIPNONE, expList) is equivalent to @MIN (expList).
• Because #MISSING values are less than positive data values and more than negative data values, if the data being calculated includes only positive and #MISSING values, @MINS returns #MISSING.
• If the data being calculated includes only negative, 0, and #MISSING values, @MINS may return either #MISSING or 0 values in an unpredictable manner.

Example
For both examples, assume a database similar to Sample Basic. The Measures dimension includes two members: COGS (cost of goods sold) and OtherInc_Exp (miscellaneous income and expenses). The data can include 0 and #MISSING values.

Example 1
Qtr1_Min = @MINS(SKIPBOTH, Jan:Mar);

This example ignores #MISSING and 0 values for all members of the Measures dimension. This example produces the following results:

<table>
<thead>
<tr>
<th></th>
<th>Jan</th>
<th>Feb</th>
<th>Mar</th>
<th>Qtr1_Min</th>
</tr>
</thead>
<tbody>
<tr>
<td>COGS</td>
<td>#MISSING</td>
<td>1500</td>
<td>2300</td>
<td>1500</td>
</tr>
<tr>
<td>OtherInc_Exp</td>
<td>-500</td>
<td>-350</td>
<td>0</td>
<td>-500</td>
</tr>
</tbody>
</table>

Example 2
Qtr1_Min = @MINS(SKIPNONE, Jan:Mar);

For all members of the Measures dimension, this example includes #MISSING and 0 values and produces the following results:

<table>
<thead>
<tr>
<th></th>
<th>Jan</th>
<th>Feb</th>
<th>Mar</th>
<th>Qtr1_Min</th>
</tr>
</thead>
<tbody>
<tr>
<td>COGS</td>
<td>#MISSING</td>
<td>1500</td>
<td>2300</td>
<td>#MISSING</td>
</tr>
<tr>
<td>OtherInc_Exp</td>
<td>-500</td>
<td>-350</td>
<td>0</td>
<td>-500</td>
</tr>
</tbody>
</table>

See Also
• @MAXS
• @MIN
• @MINSRANGE
@MINSRANGE

Returns the minimum value of `mbrName` across `XrangeList`, with options to skip missing or zero values.

Syntax

@MINSRANGE (SKIPNONE | SKIPMISSING | SKIPZERO | SKIPBOTH, `mbrName` [, `XrangeList`])

Parameters

- **SKIPNONE**
  Includes in the operation all specified cells regardless of their content

- **SKIPMISSING**
  Ignores all #MISSING values

- **SKIPZERO**
  Ignores all 0 values

- **SKIPBOTH**
  Ignores all 0 and #MISSING values

- **mbrName**
  Any valid single member name, or a function that returns a single member.

- **XrangeList**
  Optional. A valid member name, a comma-delimited list of member names, cross dimension members, or a member set function or range function (including @XRANGE) that returns a list of members from the same dimension. If `XrangeList` is not specified, Essbase uses the level 0 members from the dimension tagged as Time.

Notes

- This function enables skipping of #MISSING and 0 values, in contrast with @MINRANGE, which always includes empty values in the calculation.

- @MINSRANGE (SKIPNONE, `mbrName`, `rangeList`) is equivalent to @MINRANGE (`mbrName`, `rangeList`).

- #MISSING values are considered to be less than positive data values and more than negative data values. If the data being calculated includes only positive and #MISSING values, this function returns #MISSING.

- For all members, this function returns the value calculated for the specified member and range list.

Example

For both examples, assume a database similar to Sample Basic. The Measures dimension includes two members: COGS (cost of goods sold) and OtherInc_Exp (miscellaneous income and expenses). The data can include 0 and #MISSING values. For both members of the Measures dimension, the result is the same—the minimum value for the OtherInc_Exp member across the specified range.
Example 1

\[ Qtr1\_Min = @\texttt{MINSRANGE}\texttt{(SKIPBOTH, OtherInc}_\texttt{Exp, Jan:Mar}); \]

This example ignores the 0 value for Mar and produces the following results:

<table>
<thead>
<tr>
<th></th>
<th>Jan</th>
<th>Feb</th>
<th>Mar</th>
<th>Qtr1_Min</th>
</tr>
</thead>
<tbody>
<tr>
<td>COGS</td>
<td>#MISSING</td>
<td>1500</td>
<td>2300</td>
<td>350</td>
</tr>
<tr>
<td>OtherInc_Exp</td>
<td>500</td>
<td>350</td>
<td>0</td>
<td>350</td>
</tr>
</tbody>
</table>

Example 2

\[ Qtr1\_Min = @\texttt{MINS}\texttt{(SKIPNONE, OtherInc}_\texttt{Exp, Jan:Mar}); \]

This example does not ignore the 0 value in the calculation. This example produces the following results:

<table>
<thead>
<tr>
<th></th>
<th>Jan</th>
<th>Feb</th>
<th>Mar</th>
<th>Qtr1_Min</th>
</tr>
</thead>
<tbody>
<tr>
<td>COGS</td>
<td>#MISSING</td>
<td>1500</td>
<td>2300</td>
<td>0</td>
</tr>
<tr>
<td>OtherInc_Exp</td>
<td>500</td>
<td>350</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

See Also

- @\texttt{MINS}
- @\texttt{MINRANGE}
- @\texttt{MAXRANGE}

@MOD

Calculates the modulus of a division operation.

**Syntax**

\[ @\texttt{MOD}\texttt{(mbrName}_1\texttt{, mbrName}_2) \]

**Parameters**

\texttt{mbrName}_1\texttt{ and mbrName}_2

Members from the same dimension whose modulus is to be calculated.

**Example**

The following example is based on the Sample Basic database. Assume that the Measures dimension contains an additional member, Factor. The modulus between Profit % and Margin % is calculated with the following formula:

\[ \text{Factor} = @\texttt{MOD("Margin \%", "Profit \%")}; \]
This example produces the following report:

<table>
<thead>
<tr>
<th></th>
<th>Market Margin %</th>
<th>Profit %</th>
<th>Scenario Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jan</td>
<td>55.10</td>
<td>25.44</td>
<td>4.22</td>
</tr>
<tr>
<td>Feb</td>
<td>55.39</td>
<td>26.03</td>
<td>3.34</td>
</tr>
<tr>
<td>Mar</td>
<td>55.27</td>
<td>25.87</td>
<td>3.53</td>
</tr>
</tbody>
</table>

@MODE

Returns the mode (the most frequently occurring value) in the specified data set.

Syntax

@MODE (SKIPNONE | SKIPMISSING | SKIPZERO | SKIPBOTH, XrangeList)

Parameters

SKIPNONE
Includes all cells specified in the data set, regardless of their content, during calculation of the mode.

SKIPMISSING
Excludes all #MISSING values from the data set during calculation of the mode.

SKIPZERO
Excludes all zero (0) values from the data set during calculation of the mode.

SKIPBOTH
Excludes all zero (0) values and #MISSING values from the data set during calculation of the mode.

XrangeList
A list of numeric values across which the mode is calculated. Referred to generically throughout this topic as "the data set." Can be a valid member name, a comma-delimited list of member names, cross dimensional members, or a return value from a member set function or range function (including @XRANGE). For more information about XrangeList, see Range List Parameters.

Notes

- When two or more values in the data set occur at the same frequency, Essbase sorts the list of values in ascending order and chooses the lowest value that occurs with the most frequency as the mode. For example, if the data set contains [2,1,2,2,2,3,3,3,3], Essbase sorts the list as [1,2,2,2,2,3,3,3,3] and chooses the value [2] as the mode.
- If the data set contains no duplicate values, this function returns the smallest value in the list as the mode. For example, if the data set contains [2,4,7,10,14], @MODE returns 2 as the mode.
- If #MISSING is the mode of the data set, this function returns #MISSING unless SKIPMISSING or SKIPBOTH is specified. If you specify SKIPMISSING or
SKIPBOTH and all values in the data set are #MISSING, this function returns #MISSING. If you specify SKIPZERO or SKIPBOTH and all values in the data set are 0, this function returns #MISSING.

- When you use this function in a calculation script, use it within a FIX statement. Although FIX is not required, using it may improve calculation performance.
- When you use this function across a large range in a sparse dimension, you may need to increase the size of the calculator cache.

Example

The following example calculates the mode of the units sold for the Central region and uses @RANGE to generate the data set:

```plaintext
FIX (Central)
"Mode" = @MODE(SKIPMISSING,
   @RANGE(Sales,@CHILDREN(Central)));
ENDFIX
```

This example produces the following report:

<table>
<thead>
<tr>
<th>Units Sold</th>
<th>Illinois</th>
<th>Ohio</th>
<th>Wisconsin</th>
<th>Missouri</th>
<th>Iowa</th>
<th>Colorado</th>
<th>Central</th>
</tr>
</thead>
<tbody>
<tr>
<td>Units Sold</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>#MI</td>
<td>0</td>
<td>6</td>
<td>14</td>
</tr>
</tbody>
</table>

Mode Central 3

The following example assumes a Year dimension is added to Sample Basic. It calculates mode using cross-dimensional members in the data set.

```plaintext
FIX(Product)
"Mode" = @MODE(SKIPMISSING,@XRANGE("2011"->"Sep", "2012"->"Mar");
ENDFIX
```

The above calculation is performed across the following multidimensional range specified by XrangeList:

- 2011->Sep
- 2011->Oct
- 2011->Nov
- 2011->Dec
- 2012->Jan
- 2012->Feb
- 2012->Mar
See Also

- @RANGE
- @XRANGE

@MOVAVG

Applies a moving \( n \)-term average (mean) to an input data set. Each term in the set is replaced by a trailing mean of \( n \) terms, and the first terms (the \( n-1 \) terms) are copies of the input data. @MOVAVG modifies a data set for smoothing purposes.

Syntax

\[
@MOVAVG (mbrName [, \ n [, XrangeList]])
\]

Parameters

**mbrName**

Any valid single member name, or a function that returns a single member.

**n**

Optional. A positive integer value that represents the number of values to average. The default is 3.

**XrangeList**

Optional. A valid member name, a comma-delimited list of member names, cross dimension members, or a member set function or range function (including @XRANGE) that returns a list of members from the same dimension. If XrangeList is not specified, Essbase uses the level 0 members from the dimension tagged as Time.

Notes

- The @MOVAVG function calculates a trailing, rather than a centered, average. For example:

<table>
<thead>
<tr>
<th>Trailing Average</th>
<th>Centered Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

- While calculating the moving average, this function skips #MISSING values and decreases the denominator accordingly. For example, if one value out of three is #MISSING, Essbase adds the remaining two values and divides the sum by two.

- If you use a member set function to generate a member list for the XrangeList parameter (for example, @SIBLINGS), to ensure correct results, consider the order in which Essbase sorts the generated member list. For more information, see the help topic for the member set function you are using.

- When you use @MOVAVG in a calculation script, use it within a FIX statement. Although FIX is not required, using it may improve calculation performance.

- For periods where the width is undefined, the value is the same as for the source member. For example, you can't compute the moving average over the last three months for Jan and Feb because it doesn't exist. When this happens, Essbase simply copies the value for Jan and Feb for the moving average.
• When you use @MOVAVG across a large range in a sparse dimension, you may need to increase the size of the calculator cache.

Example

The following example is based on the Sample Basic database. Assume that the Measures dimension contains an additional member, Mov Avg.

"Mov Avg" = @MOVAVG(Sales,3,Jan:Jun);

In this example, @MOVAVG smoothes sales data for the first six months of the year (Jan through Jun). The results of @MOVAVG can be used with the @TREND function to forecast average sales data for a holiday season (for example, October - December).

This example produces the following report:

<table>
<thead>
<tr>
<th>Colas</th>
<th>New York</th>
<th>Actual</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sales</td>
<td>Mov Avg</td>
<td></td>
</tr>
<tr>
<td>======</td>
<td>=========</td>
<td>--------</td>
</tr>
<tr>
<td>Jan</td>
<td>678</td>
<td>678</td>
</tr>
<tr>
<td>Feb</td>
<td>645</td>
<td>645</td>
</tr>
<tr>
<td>Mar</td>
<td>675</td>
<td>666</td>
</tr>
<tr>
<td>Apr</td>
<td>712</td>
<td>677.3</td>
</tr>
<tr>
<td>May</td>
<td>756</td>
<td>714.3</td>
</tr>
<tr>
<td>Jun</td>
<td>890</td>
<td>786</td>
</tr>
</tbody>
</table>

In this example, Essbase averages three values at a time for the moving average. The first two values (Jan, Feb) for Mov Avg and the first two values for Sales are the same. The value for Mar represents the trailing average of Jan, Feb, and Mar. The value for Apr represents the trailing average of Feb, Mar, and Apr. The remaining values represent the trailing average for each group of three values.

See Also

• @MOVMAX
• @MOVMAX
• @MOVMIN
• @MOVSUM
• @MOVSUMX
• @TREND

@MOVMAX

Applies a moving n-term maximum (highest number) to an input data set. Each term in the set is replaced by a trailing maximum of n terms, and the first terms (the n-1 terms) are copies of the input data. @MOVMAX modifies a data set for smoothing purposes.

Syntax

@MOVMAX (mbrName [, n [, XrangeList]])
Parameters

**mbrName**
Any valid single member name, or a function that returns a single member.

**n**
Optional. A positive integer value that represents the number of values that are used to calculate the moving maximum. The default is 3.

**XrangeList**
Optional. A valid member name, a comma-delimited list of member names, cross dimension members, or a member set function or range function (including @XRANGE) that returns a list of members from the same dimension. If XrangeList is not specified, Essbase uses the level 0 members from the dimension tagged as Time.

Notes
- This function calculates a trailing, rather than a centered, maximum. For example:

<table>
<thead>
<tr>
<th>Trailing Maximum</th>
<th>Centered Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3</td>
<td>1 2 3</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

- While calculating the moving maximum, @MOVMAX skips #MISSING values. For example, if one value out of four is #MISSING, @MOVMAX calculates the maximum of the remaining three values.

- If you use an Essbase member set function to generate a member list for the XrangeList parameter (for example, @SIBLINGS), to ensure correct results, consider the order in which Essbase sorts the generated member list. For more information, see the help topic for the member set function you are using.

- When you use @MOVMAX in a calculation script, use it within a FIX statement. Although FIX is not required, using it may improve calculation performance.

- When you use @MOVMAX across a large range in a sparse dimension, you may need to increase the size of the calculator cache.

Example
The following example is based on the Sample Basic database. Assume that the Measures dimension contains an additional member, Mov Max.

"Mov Max" = @MOVMAX(Sales,3,Jan:Jun);

In this example, the @MOVMAX function smoothes sales data for the first six months of the year (Jan through Jun). The results of @MOVMAX can be used with the @TREND function to forecast maximum sales data for a holiday season (for example, October - December).
In this example, Essbase uses three values at a time to calculate the moving maximum. The first two values (Jan, Feb) for Mov Max and the first two values for Sales are the same. The value for Mar represents the trailing maximum of Jan, Feb, and Mar. The value for Apr represents the trailing maximum of Feb, Mar, and Apr. The remaining values represent the trailing maximum for each group of three values.

See Also

- @MOAVG
- @MOVMED
- @MOVMIN
- @MOVSUM
- @MOVSUMX
- @TREND

@MOVMED

Applies a moving n-term median (middle number) to an input data set. Each term in the list is replaced by a trailing median of n terms, and the first terms (the n-1 terms) are copies of the input data. @MOVMED modifies a data set for smoothing purposes.

Syntax

@MOVMED (mbrName [, n [, XrangeList]])

Parameters

mbrName
Any valid single member name, or a function that returns a single member.

n
Optional. A positive integer value that represents the number of values that are used to calculate the moving median. The default is 3.

XrangeList
Optional. A valid member name, a comma-delimited list of member names, cross dimension members, or a member set function or range function (including @XRANGE) that returns a list of members from the same dimension. If XrangeList is not specified, Essbase uses the level 0 members from the dimension tagged as Time.

Notes

- While calculating the moving median, this function skips #MISSING values. For example, if one value out of four is #MISSING, @MOVMED calculates the median of the remaining three values.
This function calculates a trailing, rather than a centered, median. For example:

<table>
<thead>
<tr>
<th>Trailing Median</th>
<th>Centered Median</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3</td>
<td>1 2 3</td>
</tr>
<tr>
<td></td>
<td>2</td>
</tr>
</tbody>
</table>

- If the group of values being used to calculate the median contains an even number of values, @MOVMED averages the two numbers in the middle.

- If you use an Essbase member set function to generate a member list for the `XrangeList` parameter (for example, `@SIBLINGS`), to ensure correct results, consider the order in which Essbase sorts the generated member list. For more information, see the help topic for the member set function you are using.

Example

The following example is based on the Sample Basic database. Assume that the Measures dimension contains an additional member, Mov Med.

```
"Mov Med" = @MOVMED(Sales,3,Jan:Jun);
```

In this example, @MOVMED smoothes sales data for the first six months of the year (Jan through Jun). The results could be used with the `@TREND` function to forecast sales data for a holiday season (for example, October - December).

This example produces the following report:

<table>
<thead>
<tr>
<th>Colas</th>
<th>New York</th>
<th>Actual</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sales</td>
<td>Mov Med</td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>----------</td>
<td></td>
</tr>
<tr>
<td>Jan</td>
<td>678</td>
<td>678</td>
</tr>
<tr>
<td>Feb</td>
<td>645</td>
<td>645</td>
</tr>
<tr>
<td>Mar</td>
<td>675</td>
<td>675</td>
</tr>
<tr>
<td>Apr</td>
<td>712</td>
<td>712</td>
</tr>
<tr>
<td>May</td>
<td>756</td>
<td>756</td>
</tr>
<tr>
<td>Jun</td>
<td>890</td>
<td>756</td>
</tr>
</tbody>
</table>

In this example, Essbase uses three values at a time to calculate the moving median. The first two values (Jan,Feb) for Mov Med are the same as the first two values for Sales. The value for Mar represents the trailing median of Jan, Feb, and Mar. The value for Apr represents the trailing median of Feb, Mar, and Apr. The remaining values represent the trailing median of each group of three values.

See Also

- @MOVAVG
- @MOVMAX
- @MOVMIN
- @MOVSUM
@MOVMIN

Applies a moving n-term minimum (lowest number) to an input data set. Each term in the list is replaced by a trailing minimum of n terms, and the first terms (the n-1 terms) are copies of the input data. @MOVMIN modifies a data set for smoothing purposes.

Syntax

@MOVMIN (mbrName [, n [, XrangeList]])

Parameters

mbrName
Any valid single member name, or a function that returns a single member.

n
Optional. A positive integer value that represents the number of values that are used to calculate the moving minimum. The default is 3.

XrangeList
Optional. A valid member name, a comma-delimited list of member names, cross dimension members, or a member set function or range function (including @XRANGE) that returns a list of members from the same dimension. If XrangeList is not specified, Essbase uses the level 0 members from the dimension tagged as Time.

Notes

• While calculating the moving minimum, @MOVMIN skips #MISSING values. For example, if one value out of four is #MISSING, @MOVMIN calculates the minimum of the remaining three values.

• This function calculates a trailing, rather than a centered, minimum. For example:

<table>
<thead>
<tr>
<th>Trailing Minimum</th>
<th>Centered Minimum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3</td>
<td>1 2 3</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

• If you use a member set function to generate a member list for the XrangeList parameter (for example, @SIBLINGS), to ensure correct results, consider the order in which Essbase sorts the generated member list. For more information, see the help topic for the member set function you are using.

• When you use @MOVMIN in a calculation script, use it within a FIX statement. Although FIX is not required, using it may improve calculation performance.

• When you use @MOVMIN across a large range in a sparse dimension, you may need to increase the size of the calculator cache.
Example

The following example is based on the Sample Basic database. Assume that the Measures dimension contains an additional member, Mov Min.

"Mov Min" = @MOVMIN(Sales,3,Jan:Jun);

In this example, the @MOVMIN function smoothes sales data for the first six months of the year (Jan through Jun). The results of @MOVMIN can be used with the @TREND to forecast minimum sales data for the holiday season (for example, October - December).

This example produces the following report:

<table>
<thead>
<tr>
<th></th>
<th>Colas Sales</th>
<th>New York Mov Min</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jan</td>
<td>678</td>
<td>678</td>
</tr>
<tr>
<td>Feb</td>
<td>645</td>
<td>645</td>
</tr>
<tr>
<td>Mar</td>
<td>675</td>
<td>645</td>
</tr>
<tr>
<td>Apr</td>
<td>712</td>
<td>645</td>
</tr>
<tr>
<td>May</td>
<td>756</td>
<td>675</td>
</tr>
<tr>
<td>Jun</td>
<td>890</td>
<td>712</td>
</tr>
</tbody>
</table>

In this example, Essbase uses three values at a time to calculate the moving minimum. The first two values (Jan,Feb) for Mov Min and the first two values for Sales are the same. The value for Mar represents the trailing minimum of Jan, Feb, and Mar. The value for Apr represents the trailing minimum of Feb, Mar, and Apr. The remaining values represent the trailing minimum for each group of three values.

See Also

- @MOVAVG
- @MOVMAX
- @MOVMED
- @MOVMED
- @MOVSUM
- @MOVSUMX
- @TREND

@MOVSUM

Applies a moving sum to the specified number of values in an input data set. @MOVSUM modifies a data set for smoothing purposes.

Syntax

@MOVSUM (mbrName [, n [, XrangeList]])
Parameters

**mbrName**
Any valid single member name, or a function that returns a single member.

**n**
Optional. A positive integer value that represents the number of values to sum. The default is 3.

**XrangeList**
Optional. A valid member name, a comma-delimited list of member names, cross dimension members, or a member set function or range function (including @XRANGE) that returns a list of members from the same dimension. If XrangeList is not specified, Essbase uses the level 0 members from the dimension tagged as Time.

Notes

- For example, if you specify 3 members of the Time dimension in the Sample Basic database, @MOVSUM at Mar is the sum of the values for Jan, Feb, and Mar; @MOVSUM at Apr is the sum of the values for Feb, Mar, and Apr. However, Jan and Feb have no @MOVSUM value, and are called trailing members. Trailing members are copies of the input values. If you wish to assign different values to trailing members, use @MOVSUMX instead.

- The @MOVSUM function calculates a trailing, rather than a centered, sum. This example illustrates the difference:

<table>
<thead>
<tr>
<th>Trailing Sum</th>
<th>Centered Sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3</td>
<td>1 2 3</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
</tr>
</tbody>
</table>

- While calculating the moving sum, @MOVSUM skips #MISSING values. For example, if one value out of three is #MISSING, Essbase adds the remaining two values.

- If you use an Essbase member set function to generate a member list for the XrangeList parameter (for example, @SIBLINGS), to ensure correct results, consider the order in which Essbase sorts the generated member list. For more information, see the help topic for the member set function that you are using.

- When you use @MOVSUM in a calculation script, use it within a FIX statement. Although FIX is not required, using it may improve calculation performance.

- When you use @MOVSUM across a large range in a sparse dimension, you may need to increase the size of the calculator cache.

Example

The following example is based on the Sample Basic database. Assume that the Measures dimension contains an additional member, Mov Sum.

"Mov Sum" = @MOVSUM(Sales,3,Jan:Jun);
This example produces the following report:

<table>
<thead>
<tr>
<th>Colas</th>
<th>New York</th>
<th>Actual</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sales</td>
<td>Mov Sum</td>
<td></td>
</tr>
<tr>
<td>======</td>
<td>========</td>
<td>=======</td>
</tr>
<tr>
<td>Jan</td>
<td>678</td>
<td>678</td>
</tr>
<tr>
<td>Feb</td>
<td>645</td>
<td>645</td>
</tr>
<tr>
<td>Mar</td>
<td>675</td>
<td>1998</td>
</tr>
<tr>
<td>Apr</td>
<td>712</td>
<td>2032</td>
</tr>
<tr>
<td>May</td>
<td>756</td>
<td>2143</td>
</tr>
<tr>
<td>Jun</td>
<td>890</td>
<td>2358</td>
</tr>
</tbody>
</table>

**See Also**

- @MOVAVG
- @MOVMAX
- @MOVMED
- @MOVMin
- @MOVSUMX
- @TREND

**@MOVSUMX**

Applies a moving sum to the specified number of values in an input data set. @MOVSUMX modifies a data set for smoothing purposes.

Unlike @MOVSUM, @MOVSUMX allows you to specify the values assigned to trailing members. For example, if you specify three members of the Time dimension in the Sample Basic database, @MOVSUMX at Mar is the sum of the values for Jan, Feb, and Mar; @MOVSUMX at Apr is the sum of the values for Feb, Mar, and Apr. However, Jan and Feb have no @MOVSUMX value, and are called *trailing members*.

**Syntax**

@MOVSUMX (COPYFORWARD | TRAILMISSING | TRAILSUM, mbrName [,n[,Xrangelist]] )

**Parameters**

**COPYFORWARD**

Copies the input value into the trailing members. This behavior is the same as the @MOVSUM function.

**TRAILMISSING**

Sets the value of the trailing members to #MISSING.

**TRAILSUM**

Sums the trailing values.

**mbrName**

Any valid single member name, or a function that returns a single member.
n
Optional. A positive integer value that represents the number of values that are used
to calculate the moving maximum. The default is 3.

XrangeList
Optional. A valid member name, a comma-delimited list of member names, cross
dimension members, or a member set function or range function (including
@XRANGE) that returns a list of members from the same dimension. If XrangeList is
not specified, Essbase uses the level 0 members from the dimension tagged as Time.

Notes
• The @MOVSUMX function calculates a trailing, rather than a centered, sum. This
  example illustrates the difference:

<table>
<thead>
<tr>
<th>Trailing Sum</th>
<th>Centered Sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3</td>
<td>1 2 3</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
</tr>
</tbody>
</table>

• While calculating the moving sum, @MOVSUMX skips #MISSING values. For
  example, if one value out of three is #MISSING, Essbase adds the remaining two
  values.

• If you use a member set function to generate a member list for the XrangeList
  parameter (for example, @SIBLINGS), to ensure correct results, consider the
  order in which Essbase sorts the generated member list. For more information,
  see the help topic for the member set function that you are using.

• When you use @MOVSUMX in a calculation script, use it within a FIX statement.
  Although FIX is not required, using it may improve calculation performance.

• When you use @MOVSUMX across a large range in a sparse dimension, you may
  need to increase the size of the calculator cache.

Example
The following examples are based on the Sample Basic database. Assume that the
Measures dimension contains an additional member, "Last 3 Months of Sales," and
that the original Sales values are as shown.

Last 3 Months of Sales = @MOVSUMX (COPYFORWARD,Sales,3,Jan:Aug);

or:

Last 3 Months of Sales = @MOVSUMX (TRAILMISSING,Sales,3,Jan:Aug);

or:

Last 3 Months of Sales = @MOVSUMX (TRAILSUM,Sales,3,Jan:Aug);

These examples produce the following reports:

Sales

==========
Jan  100
Feb  150
Mar  200
Apr  250
May  300
Jun  350
Jul  400
Aug  450

Last 3 Months of Sales
COPYFORWARD
======================
100
150
450
600
750
900
1050
1200

Last 3 Months of Sales
TRAILMISSING
======================
#MISSING
#MISSING
450
600
750
900
1050
1200

Last 3 Months of Sales
TRAILSUM
======================
100
250
450
600
750
900
1050
1200

See Also
• @MOVAVG
• @MOVMAX
• @MOVMED
@NAME

Passes the enclosed string, or list of member or dimension names, as a list of strings to another function.

Syntax

@NAME (mbrName [,UNIQUE])

Parameters

mbrName
A list of member names, dimension names, or strings.

UNIQUE
Tells @NAME to return a unique member name (using shortcut qualified name format) for mbrName, if mbrName is a duplicate name. If mbrName is not a duplicate name or if duplicate member names is not enabled, UNIQUE is ignored, and only the member name is returned. The following considerations apply:

• Essbase does not support strings in functions. It treats strings as values or an array of values. @NAME processes strings.

• To learn more about the shortcut qualified name format used for unique member names, see Creating and Working With Duplicate Member Outlines in Designing and Maintaining Essbase Cubes.

Example

Example 1

The following example is based on the Sample Basic database. A user-defined function is used to retrieve the price from the table below. The user defined function (J_GetPrice) takes two string parameters, time and product name, to return the price for each product.

Table 2-31  Price Data in Sample Basic Database

<table>
<thead>
<tr>
<th>MonthName</th>
<th>ProductId</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jan</td>
<td>100-10</td>
<td>1.90</td>
</tr>
<tr>
<td>Feb</td>
<td>100-10</td>
<td>1.95</td>
</tr>
<tr>
<td>Mar</td>
<td>100-10</td>
<td>1.98</td>
</tr>
<tr>
<td>Jan</td>
<td>100-20</td>
<td>1.95</td>
</tr>
<tr>
<td>Feb</td>
<td>100-20</td>
<td>2.00</td>
</tr>
<tr>
<td>Mar</td>
<td>100-20</td>
<td>2.05</td>
</tr>
</tbody>
</table>

Price = @J_GetPrice(@NAME(@CURRMBR(Product)),@NAME(@CURRMBR(Year)));
The following report illustrates the above example:

<table>
<thead>
<tr>
<th>Price</th>
<th>Actual</th>
<th>Market</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jan</td>
<td>Feb</td>
<td>Mar</td>
</tr>
<tr>
<td>===</td>
<td>===</td>
<td>===</td>
</tr>
<tr>
<td>100-10</td>
<td>1.90</td>
<td>1.95</td>
</tr>
<tr>
<td>100-20</td>
<td>1.95</td>
<td>2.00</td>
</tr>
</tbody>
</table>

Example 2

The following example is based on the Sample Basic database:

"Profit Per Ounce" = Profit/@ATTRIBUTEVAL(@NAME(Ounces));

The @NAME function processes the string "Ounces" before passing it to @ATTRIBUTEVAL. This example produces the following report:

<table>
<thead>
<tr>
<th>Actual</th>
<th>Year</th>
<th>West</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Profit</td>
<td>Profit Per Ounce</td>
</tr>
<tr>
<td>=======</td>
<td>===========</td>
<td>================</td>
</tr>
<tr>
<td>Cola</td>
<td>4593</td>
<td>382.75</td>
</tr>
</tbody>
</table>

Example 3

For the following example, assume an outline that has duplicate member names enabled, and there are two members named New York in the Market dimension:

![Diagram]

The qualified member names for the New York members are [State].[New York] and [City].[New York].

The following example captures a qualified member name from the current calculation context:

@MEMBER(@NAME(@CURRMBR("Market"), UNIQUE))

If the current member of Market being calculated is the New York State member, the qualified member name, [State].[New York], is passed to @MEMBER, effectively differentiating it from the New York City member.

See Also

- @CURRMBR
- @MEMBER
@NEXT

Returns the \textit{n}th cell value from \textit{mbrName}, in the sequence \textit{XrangeList}, retaining all other members identical to the current member. @NEXT cannot operate outside the given range.

**Syntax**

\[
@\text{NEXT} \ (mbrName \ [, \ n, \ XrangeList])
\]

**Parameters**

\textit{mbrName}

Any valid single member name, or a function that returns a single member.

\textit{n}

Optional signed integer. If you do not specify \textit{n}, then the default is set to 1, which provides the next member in the range. Using a negative value for \textit{n} has the same effect as using the matching positive value in @PRIOR.

\textit{XrangeList}

Optional parameter specifying a sequential range of members. If a range is not specified, Essbase uses the level 0 members from the dimension tagged as Time. Can be a valid member name, a comma-delimited list of member names, cross dimensional members, or a return value from a member set function or range function (including @XRANGE). For more information about rangeList and XrangeList, see Range List Parameters.

**Example**

In this example, Next Cash for each month is derived by taking the Cash value for the following month. Since \textit{n} is not specified, the default is 1, which provides the next member in the range. Since \textit{XrangeList} is not specified, the level 0 members from the dimension tagged as Time are used (Jan,Feb,Mar, ...).

\[
"\text{Next Cash}" = @\text{NEXT}(\text{Cash})
\]

This example produces the following report:

<table>
<thead>
<tr>
<th></th>
<th>Jan</th>
<th>Feb</th>
<th>Mar</th>
<th>Apr</th>
<th>May</th>
<th>Jun</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cash</td>
<td>100</td>
<td>90</td>
<td>120</td>
<td>110</td>
<td>150</td>
<td>100</td>
</tr>
<tr>
<td>Next Cash</td>
<td>90</td>
<td>120</td>
<td>110</td>
<td>150</td>
<td>100</td>
<td>#MI</td>
</tr>
</tbody>
</table>

The following example assumes a Year dimension is added to Sample Basic.

\[
"\text{Next Sales}" = @\text{NEXT}(\text{Sales}, 1, @\text{XRANGE}("2011"->"Sep", "2012"->"Mar"))
\]
The above calculation is performed across the following multidimensional range specified by \textit{XrangeList}:

2011->Sep
2011->Oct
2011->Nov
2011->Dec
2012->Jan
2012->Feb
2012->Mar

\textbf{See Also}

\begin{itemize}
  \item @PRIOR
  \item @SHIFT
  \item @SHIFTMINUS
  \item @SHIFTMINUS
\end{itemize}

\textbf{@NEXTS}

Returns the \(n\)th cell value from \textit{mbrName}, in the sequence \textit{XrangeList}. Provides the option to skip \#MISSING, zero, or both. Works within a designated range, and retains all other members identical to the current member.

\textbf{Syntax}

\begin{verbatim}
@NEXTS (SKIPNONE | SKIPMISSING | SKIPZERO | SKIPBOTH mbrName[,n,XrangeList])
\end{verbatim}

\textbf{Parameters}

\textbf{SKIPNONE}
Includes all cells specified in the sequence, regardless of their content.

\textbf{SKIPMISSING}
Ignores all \#MISSING values in the sequence.

\textbf{SKIPZERO}
Ignores all 0 values in the sequence.

\textbf{SKIPBOTH}
Ignores all \#MISSING and 0 values in the sequence.

\textbf{mbrName}
Any valid single member name, or a function that returns a single member.

\textbf{n}
Optional signed integer. Using a negative value for \(n\) has the same effect as using the matching positive value in \textbf{@PRIORS}. If you do not specify \(n\), then a default value of 1 is assumed, which returns the next prior member from the lowest level of the dimension set as Time in the database outline.
**XrangeList**

Optional parameter specifying a sequential range of members. If a range is not specified, Essbase uses the level 0 members from the dimension tagged as Time. Can be a valid member name, a comma-delimited list of member names, cross dimensional members, or a return value from a member set function or range function (including @XRANGE).

For more information about rangeList and XrangeList, see Range List Parameters.

**Example**

In this example, Next Cash for each month is derived by taking the Cash value for the following month and ignoring both #MISSING and zero values. Because \( n \) is not specified, the default is 1, which provides the next member in the range. Also, because XrangeList is not specified, the level 0 members from the dimension set as Time are used (Jan,Feb,Mar,...).

"Next Cash" = @NEXTS(SKIPBOTH, Cash);

The following report illustrates the above example:

<table>
<thead>
<tr>
<th></th>
<th>Jan</th>
<th>Feb</th>
<th>Mar</th>
<th>Apr</th>
<th>May</th>
<th>Jun</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cash</td>
<td>1100</td>
<td>#MI</td>
<td>1000</td>
<td>1300</td>
<td>0</td>
<td>1400</td>
</tr>
<tr>
<td>Next Cash</td>
<td>1000</td>
<td>1000</td>
<td>1300</td>
<td>1400</td>
<td>1400</td>
<td>#MI</td>
</tr>
</tbody>
</table>

The following example assumes a Year dimension is added to Sample Basic.

```plaintext
FIX(East)
"Next Cash" = @NEXTS(SKIPNONE, Sales, 1, @XRANGE("2011"->"Sep", "2012"->"Mar"));
ENDFIX;
```

The above calculation is performed across the following multidimensional range specified by XrangeList:

- 2011->Sep
- 2011->Oct
- 2011->Nov
- 2011->Dec
- 2012->Jan
- 2012->Feb
- 2012->Mar

**See Also**

- @NEXT
- @PRIORS
- @XRANGE
@NEXTSIBLING

Returns the next sibling (the sibling to the immediate right) of the specified member. This function excludes the specified member. If the specified member is the last sibling, Essbase returns an empty string.

This function returns the next sibling as a string. To pass this function as a parameter of another function, where the function requires a list of members, you must nest the @NEXTSIBLING function call within a @MEMBER function call.

Syntax

@NEXTSIBLING (mbrName)

Parameters

mbrName
Any valid single member name, or a function that returns a single member.

Example

All examples are from the Sample.Basic database.

@NEXTSIBLING("100-20")

Returns 100-30 (the next sibling of 100-20).

@NEXTSIBLING("200")

Returns 300 (the next sibling of 200). @NEXTSIBLING and @SHIFTSIBLING("200",1) return the same results.

@MEMBER(@NEXTSIBLING("100-20"))

Returns 100-30 (the next sibling of 100-20).

@CHILDREN(@MEMBER(@NEXTSIBLING("East")))

Returns all children of West.

See Also

• @MEMBER
• @PREVSIBLING
• @SHIFTSIBLING

@NOTEQUAL

Returns a member set of member names that do not match the specified token name.
This function can be used on unique and duplicate-name outlines.

Syntax

@NOTEQUAL (tokenName, topMbrinHierarchy)

Parameters

tokenName
Token string value, representing the name of a member, with which to compare to members in the outline, starting with member specified in topMbrinHierarchy. The specified token name must not be qualified for duplicate members.

topMbrinHierarchy
A fully qualified name of a member in the outline on which to base the member search. The specified member and its aliases, and all of its descendants, are included in the search.
To search the entire outline, provide an empty string ("") for this parameter. For example, @NOTEQUAL("300-30", ").

Example

The following examples are based on the following duplicate-name outline:

Product
100
100-10
100-10-10
100-20
100-30
200
200-10
200-20
200-30
300
300-10
300-20
Diet
100-10
100-10-11
200-10
300-10
Bottle
200-10
300-20

@NOTEQUAL("200-10", "Product")
>Returns all of the members under the Product dimension, except for the members [Bottle].[200-10], [Diet].[200-10], and [200].[200-10].

@NOTEQUAL("200-10", "Diet")

>Returns the members Diet, [Diet].[100-10], [Diet].[100-10], [100-10-10], and [Diet].[300-10].

See Also
- @EQUAL
- @EXPAND
- @LIKE
- @MBRCOMPARE
- @MBRPARENT

@NPV

Calculates the Net Present Value of an investment based on the series of payments (negative values) and income (positive values).

Syntax

@NPV (cashflowMbr, rateMbrConst, discountFlag [, XrangeList])

Parameters

cashflowMbr
Member specification providing a series of numeric values.

rateMbrConst
Single member specification, variable name, or numeric expression, providing a constant value.

discountFlag
Single member specification, variable name, or numeric expression set to 0 or 1 to indicate whether the function should discount from the first period. 1 means do not discount from the first period.

XrangeList
Optional parameter specifying the range over which the function is calculated. If a range is not specified, Essbase uses the level 0 members from the dimension tagged as Time.

Can be a valid member name, a comma-delimited list of member names, cross dimensional members, or a return value from a member set function or range function (including @XRANGE).

For more information about rangeList and XrangeList, see Range List Parameters.
Notes

Financial functions never return a value; rather, they calculate a series of values internally based on the range specified.

Example

In this example, Value is calculated with the following formula:

\[
\text{Value} = @NPV(\text{Cash}, \text{Rate}, 0, \text{FY1990}\text{:FY1994}, \text{FY1995}\text{:FY2000});
\]

This example produces the following report:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(1,000)</td>
<td>500</td>
<td>600</td>
<td>500</td>
<td>#MISSING</td>
<td>#MISSING</td>
</tr>
<tr>
<td>Rate</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>#MISSING</td>
<td>#MISSING</td>
</tr>
<tr>
<td>Value</td>
<td>296</td>
<td>296</td>
<td>296</td>
<td>296</td>
<td>296</td>
</tr>
</tbody>
</table>

The following example assumes a Year dimension is added to Sample Basic. It calculates NPV using a multidimensional range.

\[
\text{FIX ("100-10", "New York")}
\]

"Value" = @NPV(Cash, Rate, 0, @XRANGE("2011"->"Sep", "2012"->"Mar");
ENDFIX

The above calculation is performed across the following multidimensional range specified by XrangeList:

2011->Sep
2011->Oct
2011->Nov
2011->Dec
2012->Jan
2012->Feb
2012->Mar

See Also

@PTD

@PARENT

Returns the parent of the current member being calculated in the specified dimension. If you specify the optional mbrName, that parent is combined with the specified member.

This member set function can be used as a parameter of another function, where that parameter is a member or list of members.
Syntax

@PARENT (dimName [, mbrName])

Parameters

dimName
Single dimension name specification.

mbrName
Optional. Any valid single member name, or a function that returns a single member.

Notes
• You cannot use this function in a FIX statement.
• You can use this function on both the left and right sides of a formula. If you use this function on the left side of a formula in a calculation script, associate it with a member. For example:

  Sales(@PARENT(Product) = 5);;

• In some cases, @PARENT is equivalent to @PARENTVAL, except in terms of calculation performance. For example, the following two formulas are equivalent:

  Sales = @PARENT(Profit);
  Sales = @PARENTVAL(Profit);

  In this case, using the latter formula results in better calculation performance. In general, use @PARENT as a member rather than as an implied value of a cell. For example:

  Sales = @AVG(SKIPMISSING, @ISIBLINGS(@PARENT("100")));

• The time required for retrieval and calculation may be significantly longer if this function is in a formula attached to a member tagged as Dynamic Calc or Dynamic Calc and Store.
• If you are using @PARENT within @XREF, @XREF requires @NAME to be used around @PARENT. For example:

  COGS=@XREF(Sample, @NAME(@PARENT(Product)),Sales);

Example

In the Sample Basic database:

@PARENT(Market,Sales)

returns Central->Sales, if the current member of Market being calculated is Colorado.

@PARENT(Measures)
returns Profit, if the current member of Measures being calculated is Margin.

See Also

- @ANCEST
- @ANCESTORS
- @CHILDREN
- @DESCENDANTS
- @SIBLINGS

@PARENTVAL

Returns the parent values of the member being calculated in the specified dimension.

Syntax

@PARENTVAL (dimName [, mbrName])

Parameters

dimName
Single dimension name specification that defines the focus dimension of parent values.

mbrName
Optional. Any valid single member name, or a function that returns a single member.

Example

This example is based on the Sample Basic database. The formula calculates Market Share for each state by taking each state's Sales value as a percentage of Sales for East (its parent) as a whole. Market Share->East is calculated as East's percentage of its parent, Market.

"Market Share" = Sales % @PARENTVAL(Market,Sales);

This example produces the following report:

<table>
<thead>
<tr>
<th>Cola</th>
<th>Actual</th>
<th>Jan</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sales</td>
<td>Market Share</td>
<td></td>
</tr>
<tr>
<td>=====</td>
<td>===========</td>
<td>-----</td>
</tr>
<tr>
<td>New York</td>
<td>678</td>
<td>37.42</td>
</tr>
<tr>
<td>Massachusetts</td>
<td>494</td>
<td>27.26</td>
</tr>
<tr>
<td>Florida</td>
<td>210</td>
<td>11.59</td>
</tr>
<tr>
<td>Connecticut</td>
<td>310</td>
<td>17.11</td>
</tr>
<tr>
<td>New Hampshire</td>
<td>120</td>
<td>6.62</td>
</tr>
<tr>
<td>East</td>
<td>1812</td>
<td>37.29</td>
</tr>
<tr>
<td>Market</td>
<td>4860</td>
<td>100</td>
</tr>
</tbody>
</table>

Adding the "Market Share" member and formula to the outline would produce the same result as above.
@POWER

Returns the value of the specified member or expression raised to \textit{power}.

\textbf{Syntax}

\texttt{@POWER (expression, power)}

\textbf{Parameters}

\textbf{expression}
Single member specification, variable name, function, or other numeric expression.

\textbf{power}
Single member specification, variable name, function, or other numeric expression.

\textbf{Notes}

\begin{itemize}
  \item If \textit{expression} is negative, and if \textit{power} is not an integer, Essbase returns \texttt{#MISSING}.
  \item If the value calculated by @POWER is an infinite number, Essbase returns \texttt{#MISSING}.
\end{itemize}

\textbf{Example}

\textbf{Table 2-32  \texttt{@POWER} Results}

<table>
<thead>
<tr>
<th>Usage</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>@POWER(14,3)</td>
<td>2744</td>
</tr>
<tr>
<td>@POWER(2,8)</td>
<td>256</td>
</tr>
</tbody>
</table>

\textbf{See Also}

@FACTORIAL
@PREVSIBLING

Returns the previous sibling (the sibling to the immediate left) of the specified member. This function excludes the specified member. If the specified member is the first sibling, Essbase returns an empty string.

This function returns the next sibling as a string. To pass this function as a parameter of another function, where the function requires a list of members, you must nest the @PREVSIBLING function call within a @MEMBER function call.

Syntax

@PREVSIBLING(mbrName)

Parameters

mbrName
Any valid single member name, or a function that returns a single member.

Example

All examples are from the Sample.Basic database.

@PREVSIBLING("100-20")

Returns 100-10 (the previous sibling of 100-20). The @PREVSIBLING("100-20") function and the @SHIFTSIBLING("100-20",-1) function return the same results.

Returns 100 (the previous sibling of 200).

@PREVSIBLING("100-10")

Returns an empty list, as 100-10 does not have a previous sibling.

@CHILDREN(@MEMBER(@PREVSIBLING("East")))

Returns an empty list, as there is no previous sibling of East at the same level.

See Also

- @NEXTSIBLING
- @SHIFTSIBLING

@PRIOR

Returns the nth previous cell member from mbrName, in the sequence XrangeList. All other dimensions assume the same members as the current member. @PRIOR works only within the designated range, and with level 0 members.
Syntax

@PRIOR (mbrName [, n, XrangeList])

Parameters

mbrName
Any valid single member name, or a function that returns a single member.

n
Optional signed integer. Using a negative value for n has the same effect as using the matching positive value in @NEXT. If you do not specify n, then a default value of 1 is assumed, which returns the next prior member from the lowest level of the dimension tagged as Time in the database outline.

XrangeList
Optional parameter specifying a sequential range of members. If a range is not specified, Essbase uses the level 0 members from the dimension tagged as Time. Can be a valid member name, a comma-delimited list of member names, cross dimensional members, or a return value from a member set function or range function (including @XRANGE).
For more information about rangeList and XrangeList, see Range List Parameters.

Example

In this example, Prev Inventory for each month is derived by taking the Inventory value from the previous month. Since n is not specified, the default is 1, which provides the next prior member in the range. Since XrangeList is not specified, the level 0 members from the dimension tagged as Time are used (Jan, Feb, Mar, ...).

"Prev Inventory" = @PRIOR(Inventory);

This example produces the following report:

<table>
<thead>
<tr>
<th></th>
<th>Jan</th>
<th>Feb</th>
<th>Mar</th>
<th>Apr</th>
<th>May</th>
<th>Jun</th>
</tr>
</thead>
<tbody>
<tr>
<td>======</td>
<td>===</td>
<td>===</td>
<td>===</td>
<td>===</td>
<td>===</td>
<td>===</td>
</tr>
<tr>
<td>Inventory</td>
<td>1100</td>
<td>1200</td>
<td>1000</td>
<td>1300</td>
<td>1300</td>
<td>1400</td>
</tr>
<tr>
<td>Prev Inventory</td>
<td>#MI</td>
<td>1100</td>
<td>1200</td>
<td>1000</td>
<td>1300</td>
<td>1300</td>
</tr>
</tbody>
</table>

The following example assumes a Year dimension is added to Sample Basic.

"Prev Sales" = @PRIOR(Sales, 2, @XRANGE("2011"->"Sep", "2012"->"Mar"));

The above calculation is performed across the following multidimensional range specified by XrangeList:

2011->Sep
2011->Oct
2011->Nov
2011->Dec
@PRIORS

Returns the \textit{n}th previous cell member from \textit{mbrName}, in the sequence \textit{XrangeList}. Provides options to skip \#MISSING, zero, or both \#MISSING and zero values. All other dimensions assume the same members as the current member. \@PRIORS works within the designated range.

**Syntax**

\[
@PRIORS(\text{SKIPNONE} \mid \text{SKIPMISSING} \mid \text{SKIPZERO} \mid \text{SKIPBOTH} \ mbrName[,n,\ XrangeList])
\]

**Parameters**

- **SKIPNONE**
  Includes all cells specified in the sequence, regardless of their content.

- **SKIPMISSING**
  Ignores all \#MISSING values in the sequence.

- **SKIPZERO**
  Ignores all zero values in the sequence.

- **SKIPBOTH**
  Ignores all \#MISSING and zero values in the sequence.

- **mbrName**
  Any valid single member name, or a function that returns a single member.

- **n**
  Optional signed integer. Using a negative value for \textit{n} has the same effect as using the matching positive value in the \@NEXTS function. If you do not specify \textit{n}, then a default value of 1 is assumed, which returns the next prior member from the lowest level of the dimension set as Time in the database outline.

- **XrangeList**
  Optional parameter specifying a sequential range of members. If a range is not specified, Essbase uses the level 0 members from the dimension tagged as Time. Can be a valid member name, a comma-delimited list of member names, cross dimensional members, or a return value from a member set function or range function (including \@XRANGE).
For more information about rangeList and XrangeList, see Range List Parameters.

Example

In this example, Prev Inventory for each month is derived by taking the Inventory value from the previous month and ignoring #MISSING and zero values. Because \( n \) is not specified, the default is 1, which provides the next prior member in the range. Also, because XrangeList is not specified, the level 0 members from the dimension are set as Time used as (Jan, Feb, Mar,...).

"Prev Inventory" = @PRIORS(SKIPBOTH, Inventory);

The following report illustrates this example:

<table>
<thead>
<tr>
<th></th>
<th>Jan</th>
<th>Feb</th>
<th>Mar</th>
<th>Apr</th>
<th>May</th>
<th>Jun</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inventory</td>
<td>1100</td>
<td>#MI</td>
<td>1000</td>
<td>1300</td>
<td>0</td>
<td>1400</td>
</tr>
<tr>
<td>Prev Inventory</td>
<td>#MI</td>
<td>1100</td>
<td>1100</td>
<td>1000</td>
<td>1300</td>
<td>1300</td>
</tr>
</tbody>
</table>

The following example assumes a Year dimension is added to Sample Basic.

FIX(East)
"Prev Sales" = @PRIORS(SKIPBOTH, Sales, 1, @XRANGE("2011"->"Sep", "2012"->"Mar"));
ENDFIX;

The above calculation is performed across the following multidimensional range specified by XrangeList:

2011->Sep
2011->Oct
2011->Nov
2011->Dec
2012->Jan
2012->Feb
2012->Mar

See Also

@PRIOR

@PTD

Calculates the period-to-date values of members in the dimension tagged as Time. By default, data is summed unless Accounts are tagged as "First" or "Last".

Syntax

@PTD (XrangeList)
### Parameters

**XrangeList**
World of members from the dimension tagged as Time.
Can be a valid member name, a comma-delimited list of member names, cross
dimensional members, or a return value from a member set function or range function
(including `@XRANGE`).
For more information about `XrangeList`, see Range List Parameters.

### Notes

- Financial functions never return a value; rather, they calculate a series of values
  internally based on the range specified.
- You can use `@PTD` only if the outline contains a dimension tagged as Accounts.

### Example

In this example, assume that the Year dimension in the Sample Basic database outline
contains two additional members, YTD and QTD. Using a calculation script, the YTD
and QTD members are calculated as follows:

```
YTD = @PTD(Jan:May);
QTD = @PTD(Apr:May);
```

In this example Opening Inventory is tagged with a time balance of First, and Ending
Inventory is tagged with a time balance of Last.

This example produces the following report:

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Product</th>
<th>Market</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jan</td>
<td>31538</td>
<td>117405</td>
</tr>
<tr>
<td>Feb</td>
<td>32069</td>
<td>116434</td>
</tr>
<tr>
<td>Mar</td>
<td>32213</td>
<td>115558</td>
</tr>
<tr>
<td>Qtr1</td>
<td>95820</td>
<td>117405</td>
</tr>
<tr>
<td>Apr</td>
<td>32917</td>
<td>119143</td>
</tr>
<tr>
<td>May</td>
<td>33674</td>
<td>125883</td>
</tr>
<tr>
<td>Jun</td>
<td>35088</td>
<td>136145</td>
</tr>
<tr>
<td>Qtr2</td>
<td>101679</td>
<td>119143</td>
</tr>
<tr>
<td>QTD</td>
<td>66591</td>
<td>245026</td>
</tr>
<tr>
<td>YTD</td>
<td>162411</td>
<td>117405</td>
</tr>
</tbody>
</table>

The following example assumes a Year dimension is added to Sample Basic. It
calculates YTD using a multidimensional range.

```
FIX("100-10", "New York")
YTD = @PTD(@XRANGE("2011"->"Sep", "2012"->"Mar"));
ENDFIX
```
The above calculation is performed across the following multidimensional range specified by `XrangeList`:

2011->Sep
2011->Oct
2011->Nov
2011->Dec
2012->Jan
2012->Feb
2012->Mar

**See Also**

[@NPV](#)

---

**@RANGE**

Returns a member list that crosses the specified member from one dimension (`mbrName`) with the specified member range from another dimension (`rangeList`). This function can be combined with non-range functions, such as [@AVG](#), which replaces an existing range function, such as [@AVGRANGE](#).

**Syntax**

```
@RANGE (mbrName [, rangeList])
```

**Parameters**

**mbrName**

Any valid single member name, or a function that returns a single member.

**rangeList**

Optional. A valid member name, a comma-delimited list of member names, member set functions, and range functions. If `rangeList` is not specified, Essbase uses the level 0 members from the dimension tagged as Time.

**Notes**

This function combined with the cross-dimensional operator (->) cannot be used inside a `FIX` statement.

**Example**

**Example 1**

The following example is based on the Sample Basic database. `@RANGE` is used with [@AVG](#) to determine the average sales for Colas in the West.

```
FIX(Sales)
West=@AVG(SKIPBOTH,@RANGE(Sales,@CHILDREN(West)));
ENDFIX
```
Since the calculation script fixes on Sales, only the Sales value for West are the average of the values for western states; COGS values for West are the sum of the western states. This example produces the following report:

<table>
<thead>
<tr>
<th></th>
<th>Sales Actual</th>
<th>COGS Actual</th>
</tr>
</thead>
<tbody>
<tr>
<td>Qtr3</td>
<td>Qtr4</td>
<td>Qtr3</td>
</tr>
<tr>
<td>California</td>
<td>3401</td>
<td>2070</td>
</tr>
<tr>
<td>Oregon</td>
<td>932</td>
<td>382</td>
</tr>
<tr>
<td>Washington</td>
<td>1426</td>
<td>590</td>
</tr>
<tr>
<td>Utah</td>
<td>1168</td>
<td>520</td>
</tr>
<tr>
<td>Nevada</td>
<td>496</td>
<td>222</td>
</tr>
<tr>
<td>West</td>
<td>1484.6</td>
<td>3784</td>
</tr>
</tbody>
</table>

**Example 2**

The following example is based on the Sample Basic database. Assume that the Measures dimension contains an additional member, Prod Count. @RANGE is used with @COUNT to calculate the count of all products for which a data value exists:

```
"Prod Count" = @COUNT(SKIPMISSING, @RANGE(Sales, @CHILDREN(Product)));
```

This example produces the following report. Since SKIPMISSING is specified in the formula, the #MI value for Sales->Diet Drinks is not counted as a data value:

<table>
<thead>
<tr>
<th></th>
<th>Jan Sales</th>
<th>New York Prod Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Colas</td>
<td>678</td>
<td>#MI</td>
</tr>
<tr>
<td>Root Beer</td>
<td>551</td>
<td>#MI</td>
</tr>
<tr>
<td>Cream Soda</td>
<td>663</td>
<td>#MI</td>
</tr>
<tr>
<td>Fruit Soda</td>
<td>587</td>
<td>#MI</td>
</tr>
<tr>
<td>Diet Drinks</td>
<td>#MI</td>
<td>#MI</td>
</tr>
<tr>
<td>Product</td>
<td>2479</td>
<td>4</td>
</tr>
</tbody>
</table>

**See Also**

- @LIST
- @MERGE
- @REMOVE

**@RANGEFIRSTVAL**

Returns the first value, in a range of the specified mbrList, that satisfies the criterion specified in the first function parameter.

**Syntax**

```
@RANGEFIRSTVAL(SKIPNONE | SKIPMISSING | SKIPZERO | SKIPBOTH, mbrList)
```
Parameters

**SKIPNONE**
Every cell value is considered as data.

**SKIPMISSING**
#MISSING values are not considered as data.

**SKIPZERO**
Zero (0) values are not considered as data.

**SKIPBOTH**
Zero (0) and #MISSING values are not considered as data.

**mbrList**
A valid member name, a comma-delimited list of member names, cross dimension members, or a member set function that returns a list of members from the same dimension. If you use the range operator or a function, the order of mbrList is dictated by the database outline order.

Notes
The function returns #MISSING when mbrList does not contain any value matching the criterion specified in the first argument.

Example

**Example 1**
The following examples use the Sample.Basic database.

@RANGEFIRSTVAL(SKIPMISSING, @CHILDREN("Qtr1"));

or

@RANGEFIRSTVAL(SKIPMISSING, "Jan":"Mar");

or

@RANGEFIRSTVAL(SKIPMISSING, ("Jan","Feb","Mar"))

The previous statements return the first non-#MISSING value found when sequentially looking up the values of members Jan, Feb, and Mar.

**Example 2**

@RANGEFIRSTVAL(SKIPZERO, @CHILDREN("Jan"));

Because member Jan does not have children, returns #MISSING.
Example 3

@RANGEFIRSTVAL(SKIPBOTH, @CHILDREN("Qtr1"));

Returns the first non-#MISSING and nonzero Actual value from Qtr1, using the outline order. All months have data, so the value for Jan is returned.

Example 4

@RANGEFIRSTVAL (SKIPBOTH, (Actual->Feb, Actual->Mar, Actual->Jan ))

Returns the first non-#MISSING and nonzero Actual value from the given list of months, using the order given in mbrList. All months have data, so the value for Feb is returned.

See Also

@RANGELASTVAL

@RANGELASTVAL

Returns the last value, in a range of the specified mbrList, that satisfies the criterion specified in the first function parameter.

Syntax

@RANGELASTVAL(SKIPNONE | SKIPMISSING | SKIPZERO | SKIPBOTH, mbrList)

Parameters

SKIPNONE
Every cell value is considered as data.

SKIPMISSING
#MISSING values are not considered as data.

SKIPZERO
Zero (0) values are not considered as data.

SKIPBOTH
Zero (0) and #MISSING values are not considered as data.

mbrList
A valid member name, a comma-delimited list of member names, cross dimension members, or a member set function that returns a list of members from the same dimension. If you use the range operator or a function, the order of mbrList is dictated by the database outline order.

Notes
The function returns #MISSING when mbrList does not contain any value matching the criterion specified in the first argument.
Example

Example 1

In the following example, @RANGELASTVAL sets Jan's budget sales of Diet Cola to the last actual sales of Qtr1.

FIX("100-10", "New York", "Sales", "Jan")
"Budget" = @RANGELASTVAL(SKIPBOTH, @CHILDREN(Qtr1)->"Actual");
ENDFIX

As indicated by the SKIPBOTH parameter, @RANGELASTVAL skips zero and #MISSING. The mbrList parameter is provided by the @CHILDREN expression.

The following examples use the Sample.Basic database.

Example 2

@RANGELASTVAL(SKIPMISSING, @CHILDREN("Qtr1"));

or

@RANGELASTVAL(SKIPMISSING, "Jan":"Mar");

or

@RANGELASTVAL(SKIPMISSING, ("Jan","Feb","Mar"))

The previous statements return the last non-#MISSING value found when sequentially looking up the values of members Jan, Feb, and Mar.

Example 3

@RANGELASTVAL(SKIPZERO, @CHILDREN("Jan"));

Because member Jan does not have any children, it returns #MISSING.

See Also

@RANGEFIRSTVAL

@RANK

Returns the rank of the specified members or the specified value among the values in the specified data set. The rank of a value is equivalent to its position (its rank) in the sorted data set.

Syntax

@RANK (SKIPNONE | SKIPMISSING | SKIPZERO | SKIPBOTH, value, XrangeList)
Parameters

**SKIPNONE**
Includes all cells specified in the data set, regardless of their content, during calculation of the rank.

**SKIPMISSING**
Excludes all #MISSING values from the data set during calculation of the rank.

**SKIPZERO**
Excludes all zero (0) values from the data set during calculation of the rank.

**SKIPBOTH**
Excludes all zero (0) values and #MISSING values from the data set during calculation of the rank.

**value**
(1) The member or member combination for which the rank is calculated, or (2) a constant value for which the rank is calculated.

**XrangeList**
A list of numeric values across which the rank is calculated. Referred to generically throughout this topic as “the data set.”
Can be a valid member name, a comma-delimited list of member names, cross dimensional members, or a return value from a member set function or range function (including @XRANGE).
For more information about XrangeList, see Range List Parameters.

Notes

- After SKIP processing, @RANK sorts the data set in descending order (for example, 15341, 9650, 6556, 4255, 1989). The rank of a value identifies its position in the sorted data set (for example, 15341 is ranked 1; 1989 is ranked 5).
- An input value of #MISSING returns #MISSING. #MISSING is also returned if, after SKIP processing, there are no values to compare.
- @RANK assigns the same rank to duplicate values; however, the presence of duplicate values affects the rank numbers. For example, if a list of values contains [2,2,4,5], Essbase first sorts the list [5,4,2,2] and then ranks: [5] has a rank of 1, [4] has a rank of 2, and [2] has a rank of 3. In this case, no value has a rank of 4.
- If value is a constant value and that value is not included in the data set (XrangeList), Essbase inserts the constant value in the list and then ranks it accordingly. For example, if a list of values contains [2,4,6,13], and you want to rank a value of [3] in this list, Essbase:
  1. Sorts the list in descending order [13,6,4,2]
  2. Inserts [3] in the list [13,6,4,3,2]
- When you use @RANK in a calculation script, use it within a FIX statement. Although using FIX is not required, it may improve calculation performance.
- When you use @RANK across a large range in a sparse dimension, you may need to increase the size of the calculator cache.
Example

The following example is based on the Sample Basic database. Assume that the Measures dimension contains an additional member, Sales Rank. Essbase ranks the sales values for a set of products:

"Sales Rank" = @RANK(SKIPBOTH,Sales, @RANGE(Sales,@LEVMBRS(Product,1)));

This example produces the following report. Since SKIPBOTH is specified in the formula, the #MI value for Sales->Diet Drinks is not included in the ranked list:

<table>
<thead>
<tr>
<th>New York</th>
<th>Actual</th>
<th>Jan</th>
</tr>
</thead>
<tbody>
<tr>
<td>Colas</td>
<td>678</td>
<td>1</td>
</tr>
<tr>
<td>Root Beer</td>
<td>551</td>
<td>4</td>
</tr>
<tr>
<td>Cream Soda</td>
<td>663</td>
<td>2</td>
</tr>
<tr>
<td>Fruit Soda</td>
<td>587</td>
<td>3</td>
</tr>
<tr>
<td>Diet Drinks</td>
<td>#MI</td>
<td>#MI</td>
</tr>
</tbody>
</table>

The following example assumes a Year dimension is added to Sample Basic. It ranks values using cross-dimensional members in the data set.

FIX(Product)
"Sales Rank" = @RANK(SKIPBOTH,Sales, @XRANGE("2011"->"Sep", "2012"->"Mar"));
ENDFIX

The above calculation is performed across the following multidimensional range specified by XrangeList:

2011->Sep
2011->Oct
2011->Nov
2011->Dec
2012->Jan
2012->Feb
2012->Mar

See Also

• @RANGE
• @XRANGE

@RDESCENDANTS

Returns all descendants of the specified member, or those down to the specified generation or level, including shared members, but excluding the specified member.
You can use this function as a parameter of another function, where that parameter is a list of members.

In the absence of shared members, @RDESCENDANTS and @DESCENDANTS return the same result.

Syntax

@RDESCENDANTS (mbrName [, genLevNum | genLevName])

Parameters

mbrName
Any valid single member name, or a function that returns a single member.

genLevNum
Optional. An integer value that defines the absolute generation or level number down to which to select the members. A positive integer defines a generation number. A value of 0 or a negative integer defines a level number.

genLevName
Optional. Level name or generation name down to which to select the members.

Notes

• The order of members in the result list is important to consider when you use this function with certain forecasting and statistical functions. Essbase generates the list of members in the following sequence: If a shared member is encountered, the above steps are repeated on the member being shared.

  1. The specified member
  2. The nearest descendant of the member
  3. The next nearest descendant of the member, and so on.

• You can use @IRDESCENDANTS to include the specified member in the member list.

Example

Example 1

Assume a variation of the Sample Basic database such that the Product dimension includes the following members:

Product
  100
    100-10
    100-20
    100-30
  200
    200-10
    200-20
    200-30
    200-40
  Diet
Diet has two children "100" and "200". The members "100" and "200" are shared members.

@RDESCENDANTS(Diet)

returns the members: 100, 100-10, 100-20, 100-30, 200, 200-10, 200-20, 200-30, 200-40 (in that order).

Example 2

@RDESCENDANTS(Profit)

returns Margin, Sales, COGS, Total Expenses, Marketing, Payroll, and Misc (in that order) and is identical to @DESCENDANTS(Profit).

See Also

- @DESCENDANTS
- @IDESCENDANTS
- @IRDESCENDANTS
- @LDESCENDANTS

@RELATIVE

Returns all members at the specified generation or level relative to the specified member in the database outline.

Syntax

@RELATIVE (mbrName, genLevNum | genLevName)

Parameters

mbrName
Any valid single member name, or a function that returns a single member.

genLevNum
An integer value that defines the number of a generation or level. A positive integer defines a generation number. A value of 0 or a negative integer defines a level number.

genLevName
Generation or level name specification.

Notes

This function returns all members at the specified generation or level relative to the specified member in the database outline.
Essbase sorts the generated list of members in ascending order. Using Sample Basic as an example, @RELATIVE(200,0), returns 200-10, 200-20, 200-30, 200-40 (in that order). This order is important to consider when you use this function with certain forecasting and statistical functions.

If the specified parameters to @RELATIVE are used with the specified level or generation describing the specified member, then the specified member is included. For example, @RELATIVE(("100-10",0)) includes 100-10 in the results, because 100-10 is a level 0 member. @RELATIVE(("100",1)) includes 100 in the results, because 100 is a level 1 member.

Example

In the Sample Basic database:

@RELATIVE(Qtr1,3)
@RELATIVE(Qtr1,0)

both return the three members that are at generation 3 (or level 0) and that are below Qtr1 in the Sample Basic outline: Jan, Feb, and Mar (in that order).

@RELATIVE(Profit,-1)

returns the two members that are at level 1 and that are below Profit: Margin and Total Expenses (in that order).

@RELXRANGE

Generates a cross-dimensional list for each cell in the predefined cross-dimensional list (XrangeList), based on the relative position of the cell that is currently being calculated and the offsets, using the predefined cross-dimensional list (XrangeList) as the limit.

Syntax

@RELXRANGE (startOffset, endOffset, XrangeList)

Parameters

**startOffset**
Defines the first tuple in the cross dimensional list to be returned.

- An integer value returns a cross-dimensional member relative to the current cell being calculated, in the predefined cross-dimensional list (XrangeList).
- A negative value specifies a prior cross-dimensional member to the current cell being calculated, in XrangeList.
- A value of 0 returns the cross-dimensional member or cell currently being calculated.
- A positive value specifies a subsequent cross-dimensional member to the current cell being calculated, in XrangeList.
**endOffset**
Defines the last tuple in the cross-dimensional list to be returned. The value types are the same as for `startOffset`.

**XrangeList**
A cross-dimensional list to be used as the limit. Can be a valid member name, a comma-delimited list of member names, cross-dimensional members, or a return value from a member set function or range function (including @XRANGE). For more information about `rangeList` and `XrangeList`, see Range List Parameters.

**Notes**
- `startOffset` must be equal to or lesser than `endOffset`.
- The order of dimensions in `XrangeList` drives the sequence of the tuples in the resulting tuples list. The right-most dimension in an `XrangeList` is the most frequently incremented dimension. The increment of members in a dimension goes in outline order, or in the order of the `XrangeList` used as an argument.
- If the cell that is currently being calculated is out of the bounds of `XrangeList`, this function returns an empty cross-dimensional list.
- If `startOffset` is out of the bounds of `XrangeList`, this function returns a cross-dimensional list starting from the first member of `XrangeList`.
- If `endOffset` is out of the bounds of `XrangeList`, this function returns a cross-dimensional list ending on the last member of `XrangeList`.
- Within `XrangeList`, in the parameter list for @XRANGE, you cannot pass members from the anchor dimension, meaning the dimension of the member on which the formula is set. See the Example for a correct way to use members from the anchor dimension.

**Example**
In the parameter list for @XRANGE, you cannot pass members from the anchor dimension. This example demonstrates a correct and an incorrect usage of @XRANGE.

**Correct**
```
mbrCount=@COUNT(SKIPNONE,@RELXRANGE(-1,3,@XRANGE(Jan->Actual,May->Actual))->Sales);
```

Where `mbrCount` and `Sales` are both in the Measures dimension. Measures is the anchor dimension, meaning the dimension of the member on which the formula is set. The `XrangeList` is represented by @XRANGE(Jan->Actual,May->Actual), and returns the following:

- Jan->Actual
- Jan->Budget
- Feb->Actual
- Feb->Budget
- Mar->Actual
- Mar->Budget
@RELXRANGE operates on the XrangeList, returning lists of cross dimensional members within the defined offsets of -1 and 3.

If the current member being calculated is Jan->Actual, the count returned is 4 (offset of -1 is empty):

Jan->Actual (offset 0)
Jan->Budget (offset 1)
Feb->Actual (offset 2)
Feb->Budget (offset 3)

If the current member being calculated is Jan->Budget, the count returned is 5:

Jan->Actual (offset -1)
Jan->Budget (offset 0)
Feb->Actual (offset 1)
Feb->Budget (offset 2)
Mar->Actual (offset 3)

If the current member being calculated is Apr->Budget, the count returned is 3 (offsets of 2 and 3 are empty):

Apr->Actual (offset -1)
Apr->Budget (offset 0)
May->Actual (offset 1)

Incorrect

mbrCount=@COUNT(SKIPNONE,@RELXRANGE(0,0,@XRANGE(Sales->Jan->Actual,Sales->May->Actual)));

You cannot use Sales in the arguments for @XRANGE, because it is from the anchor dimension for mbrCount. Instead, reference a cross dimensional member with Sales and the @XRANGE function call, as shown in the correct example.

@REMAINDER

Returns the remainder value of expression.

Syntax

@REMAINDER (expression)
**Parameters**

**expression**
Single member specification, variable name, or other numeric expression.

**Example**

```
Margin = @REMAINDER("Margin %");
```

This example produces the following report:

```
Product   Market
Margin %                  Margin
Jan      Feb      Mar      Jan      Feb      Mar
===      ===      ===      ===      ===      ===
Scenario  55.10   55.39   55.27     0.10     0.39     0.27
```

**See Also**

[@TRUNCATE](#)

[@REMOVE](#)

**@REMOVE**

Removes values or members in one list from another list.

**Syntax**

```
@REMOVE (list1, list2)
```

**Parameters**

**list1**
A list of member specifications, from which the members specified in **list2** are removed.

**list2**
A list of member specifications to be removed from **list1**.

**Example**

**Example 1**

In the Sample Basic database,

```
@REMOVE(@CHILDREN(East),@LIST("New York",Connecticut))
```

returns Massachusetts, Florida, New Hampshire.

**Example 2**

The following example is based on the Sample Basic database. Assume that the Market dimension contains an additional member, Non-West.
A special analysis requires a sum of the actual sales values of a particular product family for non-western states. In this example, `@REMOVE` is called within `@SUMRANGE` to perform this analysis. `@LIST` groups the last two arguments passed to `@REMOVE` (the children of West, plus two additional members, Texas and New Mexico).

```plaintext
FIX(Sales)
"Non-West"=@SUMRANGE(Sales,@REMOVE(@LEVMBRS(Market,0),
    @LIST(@CHILDREN(West),Texas,"New Mexico")));
ENDFIX
```

This example produces the following report:

<table>
<thead>
<tr>
<th></th>
<th>Jan</th>
<th>Colas</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actual</td>
<td></td>
<td>Sales</td>
</tr>
</tbody>
</table>
|           |     | ======
| Non-West  | 5114|       |
| New York  | 678 |       |
| Massachusetts | 494 |       |
| Florida   | 410 |       |
| Connecticut | 310 |       |
| New Hampshire | 213 |       |
| East      | 2105|       |
| California| 941 |       |
| Oregon    | 450 |       |
| Washington| 320 |       |
| Utah      | 490 |       |
| Nevada    | 138 |       |
| West      | 2339|       |
| Texas     | 642 |       |
| Oklahoma  | 180 |       |
| Louisiana | 166 |       |
| New Mexico| 219 |       |
| South     | 1207|       |
| Illinois  | 579 |       |
| Ohio      | 430 |       |
| Wisconsin | 490 |       |
| Missouri  | 360 |       |
| Iowa      | 161 |       |
| Colorado  | 643 |       |
| Central   | 2663|       |

See Also
- `@INTERSECT`
- `@LIST`
- `@MERGE`
- `@RANGE`
@RETURN

Exits the calculation immediately under specified logical conditions. You can use an IF...ELSEIF command block to specify the error conditions, and use @RETURN to exit the calculation with customized error messages and levels.

Syntax

@RETURN ("ErrorMessage", [,INFO|ERROR|WARNING])

Parameters

ErrorMessage
An error message string, or any expression that returns a string.

INFO|ERROR|WARNING
An error message priority setting, where INFO, ERROR, and WARNING are priority levels:

• INFO—The message indicated in the ErrorMessage string is sent back to the client and the application log as an informational type message. This is the default.

• ERROR—The message indicated in the ErrorMessage string is sent back to the client and the application log as an error type message.

• WARNING—The message indicated in the ErrorMessage string is sent back to the client and the application log as a warning type message.

Notes

• The calculation script will stop executing when this function is called.

• This function can only be used in calculation scripts; it cannot be used in member formulas.

Example

The following example stops the calculation and returns a custom warning message if maximum values specified in the IF statement are empty:

```plaintext
FIX("Actual")
  .  "Profit"(    IF ("Marketing" < 0) OR ("Payroll" < 0) OR ("Misc" < 0) )
      @RETURN( @CONCATENATE(
          @CONCATENATE("The violation of data integrity : Market[", @NAME(@CURRMBR("Market"))),
          "] has a negative expenses. Calculations are interrupted")
        , WARNING);
ELSE
  "Profit" = ("Margin" - "Total Expenses")*0.9;
```
@ROUND

Rounds expression to numDigits.

Syntax

@ROUND (expression, numDigits [, compatibility])

Parameters

expression
Single member specification, variable name, or other numeric expression.

numDigits
Single member specification, variable name, or other numeric expression that provides an integer value. If numDigits is 0 or a positive number, expression is rounded to the number of decimal places specified by numDigits. If numDigits is a negative value, expression is rounded to the nearest 10 to the power of the absolute value of numDigits. For example:

@ROUND(1234, -2) = 1200

The default value for numDigits is 0.

compatibility
Optional backward-compatibility setting to select which algorithm you want to use for rounding margin of error.

Possible keyword values:

- COMPATPREV11121—Original rounding algorithm, in use up until Release 11.1.2.1. The integer part of the number is used to generate the rounding margin of error. Limitation: aggregate values are only accurate up to the 15th decimal place.

  Only some decimal numbers can be represented perfectly in binary. For example, if the value 1234.725 is loaded, it may be represented in binary as 1234.72499999999991. Using the COMPATPREV11121 algorithm to round this number to two decimal places returns 1234.72, though you may prefer 1234.73.

- COMPATPREV11123—Alternate rounding algorithm, in use between Release 11.1.2.1 and 11.1.2.3, to negate the representational error discussed above. The rounding margin of error was changed for better precision, which in some cases returned different results than the original algorithm.

If unspecified, the default rounding algorithm now matches the standard used by the C-language Round function. The C Round function is a common rounding algorithm, used widely across platforms. It uses a built-in construct of floor and ceiling functions to map a real number to the largest previous or the smallest subsequent integer, respectively, depending on numDigits.
Example

The following example is based on the Sample Basic database:

```
SET UPDATECALC OFF;
Profit = @ROUND("Profit_%", 1);
```

This example produces the following report:

<table>
<thead>
<tr>
<th>Market</th>
<th>Product</th>
<th>Profit_%</th>
<th>Profit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jan</td>
<td>Feb</td>
<td>Mar</td>
<td>Jan</td>
</tr>
<tr>
<td>===</td>
<td>===</td>
<td>===</td>
<td>===</td>
</tr>
<tr>
<td>Scenario</td>
<td>21.37</td>
<td>19.09</td>
<td>18.46</td>
</tr>
</tbody>
</table>

See Also

- @ABS
- @INT
- @REMAINDER
- @TRUNCATE

@RSIBLINGS

Returns the right siblings of the specified member.

Syntax

```
@RSIBLINGS (mbrName)
```

Parameters

- **mbrName**
  Any valid single member name, or a function that returns a single member.

Notes

This function returns all of the right siblings of the specified member. Right siblings are children that share the same parent as the member and that follow the member in the database outline. This function excludes the specified member.

This function can be used as a parameter of another function, where that parameter is a list of members.

Essbase sorts the right siblings in ascending order. Using Sample Basic as an example, if you specify 200-10 for `mbrName`, Essbase returns 200-20, 200-30, 200-40 (in that order). This order is important to consider when you use @RSIBLINGS with certain forecasting and statistical functions.
Example
In the Sample Basic database:

@RSIBLINGS(Florida)

returns Connecticut and New Hampshire (in that order). These members appear below Florida in the Sample Basic outline.

@RSIBLINGS(Sales)

returns COGS because this member appears below Sales in the Sample Basic outline.

See Also
- @IRSIBLINGS
- @LSIBLINGS
- @NEXTSIBLING
- @PREVSIBLING
- @SHIFTSIBLING

@sancestval

Returns ancestor-level data based on the shared ancestor value of the current member being calculated.

Syntax

@sancestval (rootMbr, genLevNum [, mbrName])

Parameters

rootMbr
Defines a member that is used to search for the nearest occurrence of an ancestor of a shared member.

genLevNum
Integer value that defines the absolute generation or level number from which the ancestor values are to be returned. A positive integer defines a generation reference. A negative number or value of 0 defines a level reference.
To use this function or any other ancestor value function in a ragged hierarchy, use generation references instead of level references to avoid unexpected results.

mbrName
Optional. Any valid single member name, or a function that returns a single member.

Notes
- You cannot use this function in a FIX statement.
• The time required for retrieval and calculation may be significantly longer if this function is in a formula attached to a member tagged as Dynamic Calc or Dynamic Calc and Store.

Example
Marketing expenses are captured at the Product Category levels in a product planning application. The Product categories are defined as ancestors that contain shared members as children. The Marketing Expense data must be allocated down to each Product code based on Sales contribution.

The following Product hierarchy is defined:

```
Product
  100
    100-10
    100-20
  200
    200-10
    200-20
  Diet ~
    100-10  SHARED
    200-10  SHARED
  Caffeine Free ~
    100-20  SHARED
    200-20  SHARED
```

<table>
<thead>
<tr>
<th>Sales</th>
<th>Marketing</th>
</tr>
</thead>
<tbody>
<tr>
<td>100-10</td>
<td>300</td>
</tr>
<tr>
<td>100-20</td>
<td>200</td>
</tr>
<tr>
<td>100</td>
<td>500</td>
</tr>
<tr>
<td>200-10</td>
<td>100</td>
</tr>
<tr>
<td>200-30</td>
<td>400</td>
</tr>
<tr>
<td>200</td>
<td>900</td>
</tr>
<tr>
<td>100-10</td>
<td>300</td>
</tr>
<tr>
<td>200-10</td>
<td>100</td>
</tr>
<tr>
<td>Diet</td>
<td>400</td>
</tr>
<tr>
<td>100-20</td>
<td>200</td>
</tr>
<tr>
<td>200-30</td>
<td>400</td>
</tr>
<tr>
<td>Caffeine Free</td>
<td>600</td>
</tr>
</tbody>
</table>

The Marketing Expense value is allocated down to each Product code with the following formula:

```
Marketing = (Sales / @SANCESTVAL(Product, 2, Sales)) * @SANCESTVAL(Product, 2, Marketing);
```

which produces the following result:

<table>
<thead>
<tr>
<th>Sales</th>
<th>Marketing</th>
</tr>
</thead>
<tbody>
<tr>
<td>100-10</td>
<td>300</td>
</tr>
<tr>
<td></td>
<td>37.5</td>
</tr>
</tbody>
</table>
The Marketing expenses can then be reconsolidated across Products and Markets.

See Also

- @ANCESTVAL
- @MDPARENTVAL
- @PARENTVAL

@SHARE

Checks each member from rangeList to see if it has a shared member and returns a list of the shared members it has found.

Syntax

@SHARE (rangeList)

Parameters

rangeList
A comma-delimited list of members, functions that return members, and ranges of members. All the members in rangeList must be from the same dimension.

Notes

Other member-set functions return the referenced members, not the shared members. You can use @SHARE within the memberList, rangeList, expList or list parameters of other functions to provide shared members instead.

Example

The following examples are based on Sample Basic.

To remove all shared members from the Product dimension:

@REMOVE(@DESCENDANT(Product),@SHARE(@DESCENDANT((Product)))

To remove a specific member from the Product dimension, you can use @SHARE specifying the shared member to be removed:

@REMOVE(@DESCENDANT(Product),@SHARE("100-20"))
See Also
@REMOVE

@SHIFT

Returns either the prior or next n\textsuperscript{th} cell value from \textit{mbrName}, in the sequence \textit{XrangeList}, retaining all other members identical to the current member.

The direction of \texttt{@SHIFT} is wholly based on \textit{n}, with positive \textit{n} values producing an effect equivalent to \texttt{@NEXT} and negative values of \textit{n} producing an equivalent effect to \texttt{@PRIOR}.

Syntax

\texttt{@SHIFT (mbrName [,n, XrangeList])}

Parameters

\textbf{mbrName}
Any valid single member name, or a function that returns a single member.

\textbf{n}
Optional signed integer. Using a negative value for \textit{n} has the same effect as using a positive value in the \texttt{@PRIOR} function. \textit{n} must be a numeric value, not a reference, such as a member name.

\textbf{XrangeList}
Optional parameter specifying a sequential range of members. If a range is not specified, Essbase uses the level 0 members from the dimension tagged as Time. Can be a valid member name, a comma-delimited list of member names, cross dimensional members, or a return value from a member set function or range function (including \texttt{@XRANGE}). For more information about \textit{rangeList} and \textit{XrangeList}, see Range List Parameters in the topic Range and Financial Functions.

Notes

\texttt{@SHIFT} is provided as a more appropriate, self-documenting name than \texttt{@NEXT} or \texttt{@PRIOR} when the value for \textit{n} is a variable and may change from positive to negative, depending on the database state when the call occurs (that is, when the usage is likely to be NEXT and/or PRIOR).

Example

In this example, Prev Asset for each month is derived by taking the Asset value from the previous month because -1 is specified as the \textit{n} parameter. Next Avl Asset for each month is derived by taking the Asset value from two months following the current month because 2 is specified as the \textit{n} parameter. Since the range sequence is not specified for either formula, the level 0 members from the dimension tagged as Time are used.

\texttt{"Prev Asset" = @SHIFT(Asset,-1);}
\texttt{"Next Avl Asset" = @SHIFT(Asset,2);}
This example produces the following report:

<table>
<thead>
<tr>
<th></th>
<th>Jan</th>
<th>Feb</th>
<th>Mar</th>
<th>Apr</th>
<th>May</th>
<th>Jun</th>
</tr>
</thead>
<tbody>
<tr>
<td>Asset</td>
<td>100</td>
<td>110</td>
<td>105</td>
<td>120</td>
<td>115</td>
<td>125</td>
</tr>
<tr>
<td>Prev Asset</td>
<td>#MI</td>
<td>100</td>
<td>110</td>
<td>105</td>
<td>120</td>
<td>115</td>
</tr>
<tr>
<td>Next Avl Asset</td>
<td>105</td>
<td>120</td>
<td>115</td>
<td>125</td>
<td>#MI</td>
<td>#MI</td>
</tr>
</tbody>
</table>

The following example assumes a Year dimension is added to Sample Basic.

```plaintext
FIX("West")
"Prev Sales" = @SHIFT(Sales, -1, @XRANGE("2011"->"Sep", "2012"->"Mar"));
ENDFIX;
```

The above calculation is performed across the following multidimensional range specified by `XrangeList`:

- 2011->Sep
- 2011->Oct
- 2011->Nov
- 2011->Dec
- 2012->Jan
- 2012->Feb
- 2012->Mar

**See Also**

- @MDSHIFT
- @NEXT
- @PRIOR
- @SHIFTPLUS
- @SHIFTMINUS

**@SHIFTMINUS**

Can be used in place of @SHIFT, @PRIOR, or @NEXT to improve performance if the formula meets the following criteria:

- The formula is being executed in CELL mode.
- The formula has one of the following patterns:

  \[ X = Y - \text{@SHIFT}(mbrName [,n, XrangeList]) \]

  or:

  \[ X = Y - \text{@PRIOR}(mbrName [,n, XrangeList]) \]
or:

\[ X = Y - @\text{NEXT}(mbrName[,n, XrangeList]) \]

If these criteria are met, consider rewriting your formula using \@SHIFTMINUS, which runs the formula in block mode to improve performance.

Syntax

\@SHIFTMINUS (mbrName1, mbrName2 [,n, XrangeList])

Parameters

\textbf{mbrName1}

Any valid single member name, or a function that returns a single member.

\textbf{mbrName2}

Any valid single member name, or a function that returns a single member.

\textbf{n}

Optional signed integer. \( n \) must be a numeric value, not a reference, such as a member name. If you are using \@SHIFTPLUS to replace the \@NEXT function, use 1 as the value for \( n \). If you are using \@SHIFTPLUS to replace the \@PRIOR function, use \(-1\) as the value for \( n \). Default value is \(+1\).

\textbf{XrangeList}

Optional parameter specifying a sequential range of members. If a range is not specified, Essbase uses the level 0 members from the dimension tagged as Time. Can be a valid member name, a comma-delimited list of member names, cross dimensional members, or a return value from a member set function or range function (including \@XRANGE).

For more information about rangeList and XrangeList, see Range List Parameters in the topic Range and Financial Functions.

Example

The following example shows a formula using \@SHIFT().

\[
\text{Sales} = \text{Loss} - \@\text{SHIFT}(\text{Sales}, 1);
\]

Here is the formula using \@SHIFTMINUS() to improve performance:

\[
\@\text{SHIFTMINUS} (\text{Loss}, \text{Sales}, 1)
\]

The following example assumes a Year dimension is added to Sample Basic.

\[
\text{FIX("South", "East")}
\]

Sales = \@\text{SHIFTMINUS} (\text{COGS, Sales, 1, @XRANGE("2011"->"Sep", "2012"->"Mar"))};
\]

ENDFIX;
The above calculation is performed across the following multidimensional range specified by XrangeList:

2011->Sep
2011->Oct
2011->Nov
2011->Dec
2012->Jan
2012->Feb
2012->Mar

See Also
@SHIFTPLUS

@SHIFTPLUS

Can be used in place of @SHIFT, @PRIOR, or @NEXT to improve performance if the formula meets the following criteria:

• The formula is being executed in CELL mode.
• The formula has one of the following patterns:

\[
X = Y + @SHIFT(mbrName [,n, XrangeList])
\]

or:

\[
X = Y + @PRIOR(mbrName [,n, XrangeList])
\]

or:

\[
X = Y + @NEXT(mbrName [,n, XrangeList])
\]

If these criteria are met, consider rewriting your formula using @SHIFTPLUS, which runs the formula in block mode to improve performance.

Syntax

@SHIFTPLUS (mbrName1, mbrName2 [,n, XrangeList])

Parameters

mbrName1
Any valid single member name, or a function that returns a single member.

mbrName2
Any valid single member name, or a function that returns a single member.
n
Optional signed integer. \( n \) must be a numeric value, not a reference, such as a member name. If you are using @SHIFTPLUS to replace the @NEXT function, use 1 as the value for \( n \). If you are using @SHIFTPLUS to replace the @PRIOR function, use -1 as the value for \( n \). Default value is +1.

\textbf{XrangeList}

Optional parameter specifying a sequential range of members. If a range is not specified, Essbase uses the level 0 members from the dimension tagged as Time. Can be a valid member name, a comma-delimited list of member names, cross dimensional members, or a return value from a member set function or range function (including @XRANGE).

For more information about rangeList and XrangeList, see Range List Parameters in the topic Range and Financial Functions.

\textbf{Example}

The following example shows a formula using @SHIFT().

\begin{verbatim}
Sales = Loss + @SHIFT(Sales, 1);
\end{verbatim}

Here is the formula using @SHIFTPLUS() to improve performance:

\begin{verbatim}
@SHIFTPLUS (Loss, Sales, 1);
\end{verbatim}

The following example assumes a Year dimension is added to Sample Basic.

\begin{verbatim}
FIX("North")
Sales = @SHIFTPLUS (COGS, Sales, 1, @XRANGE("2011"->"Sep", "2012"->"Mar"));
ENDFIX;
\end{verbatim}

The above calculation is performed across the following multidimensional range specified by XrangeList:

\begin{verbatim}
2011->Sep
2011->Oct
2011->Nov
2011->Dec
2012->Jan
2012->Feb
2012->Mar
\end{verbatim}

\textbf{See Also}

@SHIFTMINUS

\textbf{@SHIFTSIBLING}

Returns the specified member or the \( n \)th sibling of the member. This function traverses members that are at the same level and of the same parent. If the specified relative position moves beyond the first or last sibling, Essbase returns an empty string.
This function returns the next sibling as a string. To pass the @SHIFTSIBLING function as a parameter of another function, where the function requires a list of members, you must nest the @SHIFTSIBLING function call within a @MEMBER function call.

Syntax

@SHIFTSIBLING (mbrName [, relativePosition])

Parameters

mbrName
Any valid single member name, or a function that returns a single member.

relativePosition
Optional. The integer that defines the position relative to the specified member. Valid values:
- 0 (Default) Returns the specified member.
- < 0 (negative integer): Returns the previous sibling.
- > 0 (positive integer): Returns the next sibling.

Example

All examples are from the Sample.Basic database.

@SHIFTSIBLING("100-20", 0)

Returns 100-20 (the specified member).

@SHIFTSIBLING("200", 1)

Returns 300 (the next sibling of 200). The @SHIFTSIBLING("200", 1) function and the @NEXTSIBLING("200") function return the same results.

Returns 400 (the second-next sibling of 200).

@SHIFTSIBLING("100-20", -1)

Returns 100-10 (the previous sibling of 100-20). The @SHIFTSIBLING("100-20", -1) function and the @PREVSIBLING("100-20") function return the same results.

@SHIFTSIBLING("100-10", 9)

Returns an empty string, as 100-10 does not have a ninth sibling.

@CHILDREN(@MEMBER(@SHIFTSIBLING("East")))

Returns all children of East. Because no shift position is specified, the default shift position is 0, which means the current member.
@SIBLINGS

Returns all siblings of the specified member.

Syntax

@SIBLINGS (mbrName)

Parameters

mbrName
Any valid single member name, or a function that returns a single member.

Notes

This function returns all siblings of the specified member. This function excludes the specified member.

This function can be used as a parameter of another function, where that parameter is a list of members.

Essbase sorts the generated list of members as follows:

1. Left siblings of the member (siblings appearing above the member in the database outline) in descending order
2. Right siblings of the member (siblings appearing below the member in the database outline) in ascending order

Using Sample Basic as an example, if you specify 200-30 for mbrName, Essbase returns 200-20, 200-10, 200-40 (in that order). This order is important to consider when you use this function with certain forecasting and statistical functions.

Example

In the Sample Basic database:

@SIBLINGS (Washington)

Returns Oregon, California, Utah, and Nevada (in that order).

@SIBLINGS(East)

Returns West, South, and Central (in that order).
@SLN

Calculates the periodic amount that an asset in the current period may be depreciated, calculated across a range of periods. The depreciation method used is straight-line depreciation:

\[ \text{cost} - \frac{\text{salvage value}}{\text{life}} \]

The SLN method assumes that the asset depreciates by the same amount each period.

More than one asset may be depreciated over the range. The value is depreciated from its entry period to the last period in the range. The resulting value represents the sum of all the per-period depreciation values of each asset being depreciated.

Syntax

@SLN (costMbr, salvageMbrConst, lifeMbrConst [, XrangeList])

Parameters

**costMbr**
Single member specification representing an input asset for the current period.

**salvageMbrConst**
Single member specification, variable name, or numeric expression, providing a constant numeric value. This value represents the value of the asset in the current period at the end of the useful life of the asset.

**lifeMbrConst**
Single member specification, variable name, or numeric expression representing the useful life of the asset.

**XrangeList**
Optional parameter specifying the range over which the function accepts input and returns depreciation values. If a range is not specified, Essbase uses the level 0 members from the dimension tagged as Time. Can be a valid member name, a comma-delimited list of member names, cross dimensional members, or a return value from a member set function or range function (including @X RANGE). For more information about rangeList and XrangeList, see Range List Parameters.

Notes

Financial functions never return a value; rather, they calculate a series of values internally based on the range specified.
Example

In this example, the depreciation for each year is calculated by taking into account the initial asset (Asset), the salvage value of the asset (Residual), and the life of the asset (Life).

"SLN Dep" = @SLN(Asset,Residual,Life,FY1991:FY1995);

This example produces the following report:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Asset</td>
<td>9,000</td>
<td>0</td>
<td>1,000</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Residual</td>
<td>750.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Life</td>
<td>5.00</td>
<td>#MI</td>
<td>5.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
</tr>
<tr>
<td>SLN Dep</td>
<td>1650</td>
<td>1650</td>
<td>1850</td>
<td>1850</td>
<td>1850</td>
<td>0</td>
</tr>
</tbody>
</table>

The following example assumes a Year dimension is added to Sample Basic. It calculates depreciation using a multidimensional range.

FIX ("100-10", "New York")
"SLN Dep" = @SLN(Asset,Residual,Life,@XRANGE("2011"->"Sep", "2012"->"Mar");
ENDFIX

The above calculation is performed across the following multidimensional range specified by XrangeList:

2011->Sep
2011->Oct
2011->Nov
2011->Dec
2012->Jan
2012->Feb
2012->Mar

See Also
• @DECLINE
• @SYD

@SPARENTVAL

Returns parent-level data based on the shared parent value of the current member being calculated.

Syntax

@SPARENTVAL (RootMbr [, mbrName])
Parameters

**RootMbr**
Defines a member that is used to search for the nearest occurrence of a parent of a shared member.

**mbrName**
Optional. Any valid single member name, or a function that returns a single member.

Notes

- You cannot use this function in a **FIX** statement.
- The time required for retrieval and calculation may be significantly longer if this function is in a formula attached to a member tagged as Dynamic Calc or Dynamic Calc and Store.

Example

Marketing expenses are captured at the Product Category levels in a product planning application. The Product categories are defined as parents that contain shared members as children. The Marketing Expense data must be allocated down to each Product code based on Sales contribution.

The following Product hierarchy is defined:

<table>
<thead>
<tr>
<th>Product</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
</tr>
<tr>
<td>100-10</td>
</tr>
<tr>
<td>100-20</td>
</tr>
<tr>
<td>200</td>
</tr>
<tr>
<td>200-10</td>
</tr>
<tr>
<td>200-20</td>
</tr>
<tr>
<td>Diet</td>
</tr>
<tr>
<td>~</td>
</tr>
<tr>
<td>100-10</td>
</tr>
<tr>
<td>200-10</td>
</tr>
<tr>
<td>Caffeine Free</td>
</tr>
<tr>
<td>100-20</td>
</tr>
<tr>
<td>200-20</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Sales</th>
<th>Marketing</th>
</tr>
</thead>
<tbody>
<tr>
<td>100-10</td>
<td>300</td>
<td>0</td>
</tr>
<tr>
<td>100-20</td>
<td>200</td>
<td>0</td>
</tr>
<tr>
<td>100</td>
<td>500</td>
<td>0</td>
</tr>
<tr>
<td>200-10</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>200-30</td>
<td>400</td>
<td>0</td>
</tr>
<tr>
<td>200</td>
<td>900</td>
<td>0</td>
</tr>
<tr>
<td>100-10</td>
<td>300</td>
<td>0</td>
</tr>
<tr>
<td>200-10</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>Diet</td>
<td>400</td>
<td>50</td>
</tr>
<tr>
<td>100-20</td>
<td>200</td>
<td>0</td>
</tr>
<tr>
<td>200-30</td>
<td>400</td>
<td>0</td>
</tr>
<tr>
<td>Caffeine Free</td>
<td>600</td>
<td>40</td>
</tr>
</tbody>
</table>
The Marketing Expense value is allocated down to each Product code with the following formula:

\[
\text{Marketing} = \left( \frac{\text{Sales}}{\text{SPARENTVAL(Product, Sales)}} \right) \times \text{SPARENTVAL(Product, Marketing)};
\]

which produces the following result:

<table>
<thead>
<tr>
<th></th>
<th>Sales</th>
<th>Marketing</th>
</tr>
</thead>
<tbody>
<tr>
<td>100-10</td>
<td>300</td>
<td>37.5</td>
</tr>
<tr>
<td>100-20</td>
<td>200</td>
<td>13.3</td>
</tr>
<tr>
<td>100</td>
<td>500</td>
<td>#Missing</td>
</tr>
<tr>
<td>200-10</td>
<td>100</td>
<td>12.5</td>
</tr>
<tr>
<td>200-30</td>
<td>400</td>
<td>26.7</td>
</tr>
<tr>
<td>200</td>
<td>900</td>
<td>#Missing</td>
</tr>
<tr>
<td>100-10</td>
<td>300</td>
<td>37.5</td>
</tr>
<tr>
<td>200-10</td>
<td>100</td>
<td>12.5</td>
</tr>
<tr>
<td>Diet</td>
<td>400</td>
<td>#Missing</td>
</tr>
<tr>
<td>100-20</td>
<td>200</td>
<td>13.3</td>
</tr>
<tr>
<td>200-30</td>
<td>400</td>
<td>26.7</td>
</tr>
<tr>
<td>Caffeine Free</td>
<td>600</td>
<td>#Missing</td>
</tr>
</tbody>
</table>

The Marketing expenses can then be reconsolidated across Products and Markets.

See Also

- @ANCESTVAL
- @MDPARENTVAL
- @PARENTVAL

@SPLINE

Applies a smoothing spline to a set of data points. A spline is a mathematical curve that smoothes or interpolates data.

Syntax

@SPLINE (YmbrName [, s [, XmbrName [, XrangeList]]])

Parameters

YmbrName
A valid single member name that contains the dependent variable values used (when crossed with rangeList) to construct the spline.

s
Optional. A zero (0) or positive value that determines the smoothness parameter. The default value is 1.0.
XmbrName
Optional. A valid single member name that contains the independent variable values used (when crossed with rangeList) to construct the spline. The default independent variable values are 0,1,2,3, and so on.

XrangeList
Optional. A valid member name, a comma-delimited list of member names, cross dimension members, or a member set function or range function (including @XRANGE) that returns a list of members from the same dimension. If XrangeList is not specified, Essbase uses the level 0 members from the dimension tagged as Time.

Notes
• XrangeList must contain at least two values.
• If XrangeList contains gaps in the data (for example: Jan, Feb, Mar, Jun, Jul), be sure to specify XmbrName (for example: 0,1,2,5,6) so that correct results are returned.
• This function skips #MISSING values in YmbrName and XmbrName; in the result, Essbase replaces the #MISSING values of YmbrName with the spline values.
• This function calculates a smoothing cubic spline for (n > 0).
• Setting the smoothness parameter (s) to 0 produces an interpolating spline, that is, a spline that fits the initial data exactly. Increasing s results in a smoother spline but a less exact approximation of the initial data.
• @SPLINE can be used with @TREND to forecast future values that are based on the values smoothed with @SPLINE.
• If you use an Essbase member set function to generate a member list for the XrangeList parameter (for example, @SIBLINGS), to ensure correct results, consider the order in which Essbase sorts the generated member list. For more information, see the help topic for the member set function you are using.
• When you use @SPLINE in a calculation script, use it within a FIX statement. Although using FIX is not required, it may improve calculation performance.
• When you use @SPLINE across a large range in a sparse dimension, you may need to increase the size of the calculator cache.
• View the Algorithm for the smoothing spline.

Algorithm
\((x_i, y_i), \ i = 0, 1, \ldots, N\)

A function \(S(x)\) defined on grid \(X = \{x_i\}\) is called a smoothing cubic spline function if the function

1) is a cubic polynomial
\[ S(x) = S_i(x) = a_0^i + a_1^i (x - x_i) + a_2^i (x - x_i)^2 + a_3^i (x - x_i)^3 \]

on each partial segment \([x_i, x_{i+1}], \ i = 0, 1, \ldots, N - 1\),

2) has the continuous second derivatives on segment \([x_0, x_N]\), that is, the function is of class \(C^2[x_0, x_N]\),

3) minimizes the functional
\[ J(f) = s \int_0^1 |f''(x)|^2 \, dx + \sum_{i=0}^{N-1} (f(x_i) - y_i)^2, \]

where \(y_i\) are given numbers and \(s \geq 0\), where \(s\) is the smoothness parameter, and

4) satisfies the boundary condition:
\[ S'(x_0) = 0, \quad S''(x_N) = 0 \]

In each segment \([x_i, x_{i+1}], \ i = 0, 1, \ldots, N - 1\), the smoothing spline function is sought in the following modified form:
\[ S(x) = S_i(x) = a_0^i + a_1^i (x - x_i) + a_2^i (x - x_i)^2 + a_3^i (x - x_i)^3 \]

where
\[ h_i = x_{i+1} - x_i, \quad \ell = \frac{x - x_i}{h_i} \]

and numbers \(x_i\) and \(h_i\), \(i = 0, 1, \ldots, N\) are a solution of a linear algebraic system.
The numbers \( n_i \) are solutions to the system:

\[
\begin{align*}
    a_0 n_0 + b_0 n_1 + c_0 n_2 &= g_0, \\
    b_0 n_0 + a_1 n_1 + b_1 n_1 + c_1 n_2 &= g_1, \\
    c_{i-2} n_{i-2} + b_{i-1} n_{i-1} + a_i n_i + b_i n_{i+1} + c_i n_{i+2} &= g_i, \quad i = 2, 3, \ldots, N-2, \\
    c_{N-3} n_{N-3} + b_{N-2} n_{N-2} + a_{N-1} n_{N-1} + b_{N-1} n_N &= g_{N-1}, \\
    c_{N-2} n_{N-2} + b_{N-1} n_{N-1} + a_N n_N &= g_N,
\end{align*}
\]

where

\[
\begin{align*}
    a_i &= \frac{1}{3} (b_{i-1} + h_i) + \frac{1}{h_i} s + \left( \frac{1}{h_{i-1}} + \frac{1}{h_i} \right) s + \frac{1}{h_i^2} s, \\
    b_i &= \frac{1}{6} h_i - \frac{s}{h_i} \left[ \left( \frac{1}{h_{i-1}} + \frac{1}{h_i} \right) + \left( \frac{1}{h_i} + \frac{1}{h_{i+1}} \right) \right], \\
    c_i &= \frac{s}{h_i h_{i+1}}, \quad i = 1, 2, \ldots, N-3 \\
    g_i &= \frac{y_{i+1} - y_i}{h_i} - \frac{y_i - y_{i-1}}{h_{i-1}}, \quad i = 1, 2, \ldots, N-1
\end{align*}
\]

The end conditions are:

\[
\begin{align*}
    a_0 &= 1, \quad b_0 = 0, \quad c_0 = 0, \quad g_0 = 0, \\
    a_N &= 1, \quad b_{N-1} = 0, \quad c_{N-2} = 0, \quad g_N = 0.
\end{align*}
\]

When numbers \( m_i \) are found, the magnitudes \( z_i \) are easily determined by formulas

\[
z_i = y_i - s D_i, \quad i = 0, 1, 2, \ldots, N.
\]
The following example is based on the Sample Basic database. Assume that the Measures dimension contains an additional member, Sales Spline. The formula calculates the spline of Sales values for Jan through Jun, based on a smoothness parameter of 2.

"Sales Spline" = @SPLINE(Sales,2,,Jan:Jun);

This example produces the following report:

<table>
<thead>
<tr>
<th>Colas</th>
<th>Actual</th>
<th>New York</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sales</td>
<td>Sales Spline</td>
<td></td>
</tr>
<tr>
<td>======</td>
<td>===========</td>
<td></td>
</tr>
<tr>
<td>Jan</td>
<td>645</td>
<td>632.8941564</td>
</tr>
<tr>
<td>Feb</td>
<td>675</td>
<td>675.8247101</td>
</tr>
<tr>
<td>Mar</td>
<td>712</td>
<td>724.7394598</td>
</tr>
<tr>
<td>Apr</td>
<td>756</td>
<td>784.2860765</td>
</tr>
<tr>
<td>May</td>
<td>890</td>
<td>852.4398456</td>
</tr>
<tr>
<td>Jun</td>
<td>912</td>
<td>919.8157517</td>
</tr>
</tbody>
</table>

See Also

@TREND

@STDEV

Calculates the standard deviation of the specified data set (expList). The calculation is based upon a sample of a population. Standard deviation is a measure of how widely values are dispersed from their mean (average).

This function assumes that expList represents a sample of a population. If you want expList to represent the entire population, use @STDEVP. For large samples, the functions return similar values.

@STDEV is calculated using the "nonbiased" or "n-1" method.

@STDEV uses the following formula:

$$
D_0 = \frac{1}{h_0} (x_1 - n_0), \quad D_N = -\frac{1}{h_{N-1}} (n_N - n_{N-1}),
$$

$$
D_i = \frac{1}{h_i} (x_{i+1} - n_i) - \frac{1}{h_{i-1}} (n_i - n_{i-1}), \quad i = 1, 2, ..., N - 1.
$$

And now given any x, use (*^*) and (**^*) from above to calculate S(x).
Syntax

@STDEV (SKIPNONE | SKIPMISSING | SKIPZERO | SKIPBOTH, expList)

Parameters

SKIPNONE
Includes all cells specified in expList, regardless of their content, during calculation of the standard deviation.

SKIPMISSING
Excludes all #MISSING values from expList during calculation of the standard deviation.

SKIPZERO
Excludes all zero (0) values from expList during calculation of the standard deviation.

SKIPBOTH
Excludes all zero (0) values and #MISSING values from expList during calculation of the standard deviation.

expList
Comma-delimited list of member specifications, variable names, functions, or numeric expressions. expList provides a list of numeric values across which the standard deviation is calculated.

Example

The following example is based on the Sample Basic database. Assume that the Measures dimension contains an additional member, Std Deviation. This example calculates the standard deviation (based on a sample of a population) of the sales values for all products and uses @RANGE to generate expList.

```
FIX (Product)
"Std Deviation" = @STDEV(SKIPBOTH,@RANGE(Sales,@CHILDREN(Product)));
ENDFIX
```

This example produces the following report:

<table>
<thead>
<tr>
<th></th>
<th>Jan Actual</th>
<th>New York Budget</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sales</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Colas</td>
<td>678</td>
<td>640</td>
</tr>
<tr>
<td>Root Beer</td>
<td>551</td>
<td>530</td>
</tr>
<tr>
<td>Cream Soda</td>
<td>663</td>
<td>510</td>
</tr>
<tr>
<td>Fruit Soda</td>
<td>587</td>
<td>620</td>
</tr>
<tr>
<td>Diet Drinks</td>
<td>#MI</td>
<td>#MI</td>
</tr>
<tr>
<td>Product</td>
<td>2479</td>
<td>2300</td>
</tr>
</tbody>
</table>
See Also

- @RANGE
- @STDEV
- @STDEVRANGE

@STDEVP

Calculates the standard deviation of the specified data set \((\text{expList})\).

This function assumes that \(\text{expList}\) represents the entire population. If you want \(\text{expList}\) to represent a sample of a population, use @STDEV. For large samples, the functions return similar values.

Syntax

@STDEVP (SKIPNONE | SKIPMISSING | SKIPZERO | SKIPBOTH, expList)

Parameters

**SKIPNONE**
Includes all cells specified in \(\text{expList}\), regardless of their content, during calculation of the standard deviation.

**SKIPMISSING**
Excludes all #MISSING values from \(\text{expList}\) during calculation of the standard deviation.

**SKIPZERO**
Excludes all zero (0) values from \(\text{expList}\) during calculation of the standard deviation.

**SKIPBOTH**
Excludes all zero (0) values and #MISSING values from \(\text{expList}\) during calculation of the standard deviation.

**expList**
Comma-delimited list of member specifications, variable names, functions, or numeric expressions. \(\text{expList}\) provides a list of numeric values across which the standard deviation is calculated.

Notes

@STDEVP calculates the standard deviation of the specified data set \((\text{expList})\). The calculation is based upon the entire population. Standard deviation is a measure of how widely values are dispersed from their mean (average).

@STDEVP is calculated using the "biased" or "n" method.

@STDEVP uses the following formula:
Example

The following example is based on the Sample Basic database. Assume that the Measures dimension contains an additional member, Std Deviation. This example calculates the standard deviation (based on the entire population) of the sales values for all products and uses @RANGE to generate expList.

```
FIX (Product)
"Std Deviation" = @STDEVP(SKIPBOTH,@RANGE(Sales,@CHILDREN(Product)));
ENDFIX
```

This example produces the following report:

<table>
<thead>
<tr>
<th></th>
<th>Jan</th>
<th>New York</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actual</td>
<td></td>
<td></td>
</tr>
<tr>
<td>=======</td>
<td>-----</td>
<td>----------</td>
</tr>
<tr>
<td>Sales</td>
<td>Colas</td>
<td>678</td>
</tr>
<tr>
<td></td>
<td>Root Beer</td>
<td>551</td>
</tr>
<tr>
<td></td>
<td>Cream Soda</td>
<td>663</td>
</tr>
<tr>
<td></td>
<td>Fruit Soda</td>
<td>587</td>
</tr>
<tr>
<td></td>
<td>Diet Drinks</td>
<td>#MI</td>
</tr>
<tr>
<td></td>
<td>Product</td>
<td>2479</td>
</tr>
<tr>
<td>Std Deviation</td>
<td>Product</td>
<td>52.59</td>
</tr>
</tbody>
</table>

See Also

- @RANGE
- @STDEV
- @STDEVRANGE

@STDEVRANGE

Calculates the standard deviation of all values of the specified member \( mbrName \) across the specified data set \( XrangeList \). The calculation is based upon a sample of a population. Standard deviation is a measure of how widely values are dispersed from their mean (average).

This function is calculated using the "unbiased" or "n-1" method. See @STDEV for the formula used.

Syntax

@STDEVRANGE (SKIPNONE | SKIPMISSING | SKIPZERO | SKIPBOTH, mbrName [, XrangeList])
Parameters

**SKIPNONE**
Includes all cells specified in `expList`, regardless of their content, during calculation of the standard deviation.

**SKIPMISSING**
Excludes all #MISSING values from `expList` during calculation of the standard deviation.

**SKIPZERO**
Excludes all zero (0) values from `expList` during calculation of the standard deviation.

**SKIPBOTH**
Excludes all zero (0) values and #MISSING values from `expList` during calculation of the standard deviation.

**mbrName**
Any valid single member name, or a function that returns a single member.

**XrangeList**
Optional. A valid member name, a comma-delimited list of member names, cross dimension members, or a member set function or range function (including @XRANGE) that returns a list of members from the same dimension. If `XrangeList` is not specified, Essbase uses the level 0 members from the dimension tagged as Time.

Example
The following example is based on the Sample Basic database. Assume that the Measures dimension contains an additional member, Std Deviation. This example calculates the standard deviation (based on a sample of a population) of the sales values for all products.

```sql
FIX (Product)
"Std Deviation" = @STDEVRANGE(SKIPBOTH,Sales,@CHILDREN(Product));
ENDFIX
```

This example produces the following report:

<table>
<thead>
<tr>
<th></th>
<th>Jan Actual</th>
<th>New York Budget</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sales</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Colas</td>
<td>678</td>
<td>640</td>
</tr>
<tr>
<td>Root Beer</td>
<td>551</td>
<td>530</td>
</tr>
<tr>
<td>Cream Soda</td>
<td>663</td>
<td>510</td>
</tr>
<tr>
<td>Fruit Soda</td>
<td>587</td>
<td>620</td>
</tr>
<tr>
<td>Diet Drinks</td>
<td>#MI</td>
<td>#MI</td>
</tr>
<tr>
<td>Product</td>
<td>2479</td>
<td>2300</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Std Deviation</th>
<th>Product</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>60.73</td>
</tr>
<tr>
<td></td>
<td></td>
<td>64.55</td>
</tr>
</tbody>
</table>

See Also
- @STDEV
@STDEV

@SUBSTRING

Returns the requested string of characters from an existing source string. The source string can be a text string or a member name, or it can result from a specified function that returns a text string or a single member name.

Syntax

@SUBSTRING (String, StartPosition [, EndPosition])

Parameters

String
A string or a function that returns a string or a single member name (For example, @ATTRIBUTESVAL, @CONCATENATE, and @NAME return strings.)

StartPosition
Beginning character position within String to include in the substring. An integer greater than or equal to 0, where 0 corresponds to the first character in String, 1 corresponds to the second character, and so on.

EndPosition
Optional. An integer greater than or equal to 1, where 1 corresponds to the first character in String, 2 corresponds to the second character, and so on. If EndPosition is not specified or is less than StartPosition, Essbase returns all remaining characters from the source string. Note that this is a different numbering scheme that the start position uses.

Example

The following examples are based on the Sample Basic database:

Table 2-33  @SUBSTRING Examples and Results

<table>
<thead>
<tr>
<th>Function Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>@SUBSTRING (&quot;100-10&quot;,1)</td>
<td>&quot;00-10&quot;</td>
</tr>
<tr>
<td>@SUBSTRING (&quot;200-21&quot;,2,2)</td>
<td>&quot;20&quot;</td>
</tr>
<tr>
<td>@SUBSTRING (@Name(@Parent(Jan)),3) (The parent of Jan is Qtr1.)</td>
<td>&quot;1&quot;</td>
</tr>
</tbody>
</table>

See Also

- @CONCATENATE
- @MEMBER

@SUM

Returns the summation of all the values in expList.
Syntax

@SUM (expList)

Parameters

expList
Comma-delimited list of member specifications, variable names, or numeric expressions, all of which provide numeric values.

Example

In the Sample Basic database:

FIX("Total Expenses")
West=@SUM(West,East);
ENDFIX

Since the calculation script fixes on Total Expenses, the value for Total Expenses->West is equal to the sum of the value for East and the values for the states making up the West. For Sales, West and East are simply the sum of the states making up each region (that is, Sales->West is not equal to the sum of East and West). This example produces the following report:

<table>
<thead>
<tr>
<th>Product</th>
<th>Qtr1</th>
<th>Actual</th>
</tr>
</thead>
<tbody>
<tr>
<td>New York</td>
<td>7705</td>
<td>2068</td>
</tr>
<tr>
<td>Massachusetts</td>
<td>3660</td>
<td>892</td>
</tr>
<tr>
<td>Florida</td>
<td>4132</td>
<td>1313</td>
</tr>
<tr>
<td>Connecticut</td>
<td>3472</td>
<td>1087</td>
</tr>
<tr>
<td>New Hampshire</td>
<td>1652</td>
<td>801</td>
</tr>
<tr>
<td>East</td>
<td>20621</td>
<td>6161</td>
</tr>
<tr>
<td>California</td>
<td>11056</td>
<td>2742</td>
</tr>
<tr>
<td>Oregon</td>
<td>5058</td>
<td>1587</td>
</tr>
<tr>
<td>Washington</td>
<td>4835</td>
<td>1621</td>
</tr>
<tr>
<td>Utah</td>
<td>4209</td>
<td>1544</td>
</tr>
<tr>
<td>Nevada</td>
<td>6516</td>
<td>2193</td>
</tr>
<tr>
<td>West</td>
<td>31674</td>
<td>15848</td>
</tr>
</tbody>
</table>

See Also

@SUMRANGE

@SUMRANGE

Returns the summation of all the values of the specified member (mbrName) across the specified range (XrangeList).

Syntax

@SUMRANGE (mbrName [,XrangeList])
Parameters

mbrName
Any valid single member name, or a function that returns a single member.

XrangeList
Optional. A valid member name, a comma-delimited list of member names, cross dimension members, or a member set function or range function (including @XRANGE) that returns a list of members from the same dimension. If XrangeList is not specified, Essbase uses the level 0 members from the dimension tagged as Time.

Example

The following example is based on the Sample Basic database. Assume that the Year dimension contains an additional member, Partial Year. The formula for Partial Year sums the values for New York across the range of Jan through Jun. The calculation script fixes on Sales, so this formula is applied only to Sales values.

FIX(Sales)
"Partial Year"=@SUMRANGE("New York", Jan:Jun);
ENDFIX
This example produces the following report:

<table>
<thead>
<tr>
<th></th>
<th>New York</th>
<th>Colas</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actual</td>
<td>Sales</td>
<td></td>
</tr>
<tr>
<td></td>
<td>=====</td>
<td></td>
</tr>
<tr>
<td>Jan</td>
<td>678</td>
<td></td>
</tr>
<tr>
<td>Feb</td>
<td>645</td>
<td></td>
</tr>
<tr>
<td>Mar</td>
<td>675</td>
<td></td>
</tr>
<tr>
<td>Apr</td>
<td>712</td>
<td></td>
</tr>
<tr>
<td>May</td>
<td>756</td>
<td></td>
</tr>
<tr>
<td>Jun</td>
<td>890</td>
<td></td>
</tr>
<tr>
<td>Partial Year</td>
<td>4356</td>
<td></td>
</tr>
</tbody>
</table>

See Also

@SUM

@SYD

Calculates the periodic amount (usually annual) that an asset in the current period may be depreciated, across a range of periods. The depreciation method used is sum of the year’s digits.

The SYD method assumes that depreciation amounts are higher at the earlier stages of the asset's life. Thus, XrangeList can be used to specify a period to calculate.

More than one asset may be depreciated over the range. The value is depreciated from its entry period to the last period in the range. The resulting value represents the sum of all per-period depreciation values of each asset.
Syntax

@SYD (costMbr, salvageMbrConst, lifeMbrConst [, XrangeList])

Parameters

costMbr
Single member specification representing an input asset for the current period.

salvageMbrConst
Single member specification, variable name, or numeric expression, providing a constant numeric value. This value is the value of the asset in the current period after the useful life of the asset.

lifeMbrConst
Single member specification, variable name, or numeric expression representing the useful life of the asset.

XrangeList
Optional parameter specifying the range over which the function accepts input and returns depreciation values. If a range is not specified, Essbase uses the level 0 members from the dimension tagged as Time. Can be a valid member name, a comma-delimited list of member names, cross dimensional members, or a return value from a member set function or range function (including @XRANGE). For more information about rangeList and XrangeList, see Range List Parameters.

Notes

Financial functions never return a value; rather, they calculate a series of values internally based on the range specified.

Example

In this example, the depreciation for each year is calculated by taking into account the initial asset (Asset), the salvage value of the asset (Residual), and the life of the asset (Life).

"SYD Dep"=@SYD(Asset,Residual,Life,FY1999:FY2002,FY2003);

This example produces the following report:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Asset</td>
<td>9,000</td>
<td>0</td>
<td>1,000</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Residual</td>
<td>750.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
</tr>
<tr>
<td>Life</td>
<td>5.00</td>
<td>#MISSING</td>
<td>3.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>SYD Dep</td>
<td>2750</td>
<td>2200</td>
<td>2150</td>
<td>1433</td>
<td>717</td>
</tr>
</tbody>
</table>
The following example assumes a Year dimension is added to Sample Basic. It calculates depreciation using a multidimensional range.

```plaintext
FIX ("100-10", "New York")
"SYD Dep" = @SYD(Asset,Residual,Life,@XRANGE("2011"->"Sep", "2012"->"Mar"));
ENDFIX
```

The above calculation is performed across the following multidimensional range specified by `XrangeList`:

- 2011->Sep
- 2011->Oct
- 2011->Nov
- 2011->Dec
- 2012->Jan
- 2012->Feb
- 2012->Mar

**See Also**

- @DECLINE
- @SLN

**@TODATE**

Converts date strings to numbers that can be used in calculation formulas. @TODATE converts date strings into the number of seconds elapsed since midnight, January 1, 1970.

**Syntax**

```plaintext
@TODATE (formatString, dateString)
```

**Parameters**

- **formatString**
  The format of the date string, either "mm-dd-yyyy" or "dd-mm-yyyy" (must be in lower case).

- **dateString**
  The date string.

**Notes**

- If you specify a date that is earlier than 01-01-1970, this function returns an error.
- The latest date supported by this function is 12-31-2037.
Example

The following example is based on the Sample Basic database.

Marketing

(IF (@ATTRIBUTEVAL("Intro Date") >
     @TODATE("mm-dd-yyyy","06-30-1996"))
Marketing - (Marketing * .1);
ENDIF;);

This formula searches for members with an Intro Date attribute member that is later than 6-30-96 and decreases Marketing for those members by 10 percent. In order to process the formula, Essbase converts the date strings to numbers before it calculates.

This example produces the following report:

<table>
<thead>
<tr>
<th>Actual</th>
<th>Jan</th>
<th>Massachusetts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Marketing</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Intro Date_12-10-1996</td>
<td>200-30</td>
<td>9</td>
</tr>
<tr>
<td>200-40</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>Intro Date_10-01-1996</td>
<td>400-10</td>
<td>9</td>
</tr>
<tr>
<td>400-20</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>Intro Date_07-26-1996</td>
<td>200-20</td>
<td>9</td>
</tr>
<tr>
<td>Intro Date_06-26-1996</td>
<td>300-10</td>
<td>9</td>
</tr>
<tr>
<td>300-20</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>300-30</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>Intro Date_04-01-1996</td>
<td>100-20</td>
<td>10</td>
</tr>
<tr>
<td>100-30</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>Intro Date_03-25-1996</td>
<td>100-10</td>
<td>10</td>
</tr>
<tr>
<td>Intro Date_09-27-1995</td>
<td>200-10</td>
<td>10</td>
</tr>
</tbody>
</table>

See Also

- @ATTRIBUTE
- @ATTRIBUTEVAL
- @WITHATTR

@TODATEEX

Returns the numeric date value from input date-string according to the date-format specified. The date returned is the number of seconds elapsed since midnight, January 1, 1970.

If the date or the date format strings are invalid, an error is returned.

Syntax

@TODATEEX(date_format_string, string)
Parameters

date_format_string
One of the following literal strings (excluding ordered-list numbers and parenthetical examples) indicating a supported date format.

1. "mon dd yyyy" (Example: mon = Aug)
2. "Month dd yyyy" (Example: Month = August)
3. "mm/dd/yy"
4. "mm/dd/yyyy"
5. "yy.mm.dd"
6. "dd/mm/yy"
7. "dd.mm.yy"
8. "dd-mm-yy"
9. "dd Month yy"
10. "dd mon yy"
11. "Month dd, yy"
12. "mon dd, yy"
13. "mm-dd-yy"
14. "yy/mm/dd"
15. "yymmdd"
16. "dd Month yyyy"
17. "dd mon yyyy"
18. "yyyy-mm-dd"
19. "yyyy/mm/dd"
20. Long format (Example: WeekDay, Mon dd, yyyy)
21. Short format (Example: m/d/yy)

string
A date string following the rules of internal-date-format. The following examples correspond to the above listed internal date formats.

1. Jan 15 2006
2. January 15 2006
3. 01/15/06
4. 01/15/2006
5. 06.01.06
6. 15/01/06
7. 15.01.06
Notes

- This function is case-sensitive. For example, using `apr` instead of `Apr` returns an error.
- Using extra whitespace not included in the internal format strings returns an error.
- Trailing characters after the date format has been satisfied are ignored. If you erroneously use a date string of `06/20/2006` with date format `mm/dd/yy`, the trailing `06` is ignored and the date is interpreted as June 20, 2020.
- Long Format (Weekday, Mon dd, yyyy) is not verified for a day-of-week match to the given date.
  
  For example: For date string `Sunday, March 13, 2007` with date format Long Format, the input date string is parsed correctly for March 13, 2007, although March 13, 2007 does not fall on Sunday.
- If you specify a date that is earlier than 01-01-1970, this function returns an error.
- The latest date supported by this function is 12-31-2037.
- When the `yy` format is used, this function interprets years in the range 1970 to 2029.

See Also

- `@DATEDIFF`
- `@DATEPART`
- `@DATEROLL`
- `@FORMATDATE`
- `@TODAY`
@TODAY

Returns a number representing the current date on the Essbase computer. The number is the number of seconds elapsed since midnight, January 1, 1970.

Syntax

@TODAY()

Notes

• The date returned can be used as input to other functions listed in the See Also section.
• As this function is a run-time formula, you cannot use it in a FIX statement.

Example

If today's date is 15-Jul-2014, the following expression returns 15:

@DATEPART(@TODAY(), DP_DAY)

See also the example for @FORMATDATE.

See Also

• @DATEDIFF
• @DATEPART
• @DATEROLL
• @FORMATDATE
• @TODATEEX

@TREND

Calculates future values based on curve-fitting to historical values. The @TREND procedure considers a number of observations; constructs a mathematical model of the process based on these observations (that is, fits a curve); and predicts values for a future observation. You can use weights to assign credibility coefficients to particular observations, report errors of the curve fitting, choose the forecasting method to be used (for example, linear regression), and specify certain data filters.

Syntax

@TREND (Ylist, [Xlist], [weightList], [errorList], [XforecastList], YforecastList, method[, method parameters] [, Xfilter1 [, parameters]] [, XfilterN [, parameters]] [, Yfilter1 [, parameters]] [, YfilterN [, parameters]])
Parameters

Ylist
An expression list that contains known observations; for example, sales figures over a period of time.

Xlist
Optional. An expression list that contains underlying variable values. For example, for each sales figure in Ylist, Xlist may contain a value for associated time periods. If you do not specify Xlist, the default variable values are 1,2,3, and so on, up to the number of values in Ylist.

weightList
Optional. An expression list that contains weights for the data points in Ylist, for the linear regression method only. If values in weightList are #MISSING, the default is 1. Weights for methods other than linear regression are ignored. Negative weights are replaced with their absolute values.

errorList
Optional. Member list that represents the differences between the data points in Ylist and the data points on the line or curve (as specified for method).

XforecastList
Optional. Expression list that contains the underlying variable values for which the forecasting is sought. If you do not specify XforecastList, the values are assumed to be as follows: {(last value in Xlist + 1), (last value in Xlist + 2), ...} up to (last value in Xlist + the number of values in YforecastList)
If you forecast consecutively from where Ylist stops, you do not need to specify XforecastList. If you want to move the forecasting period forward, specify the new period with XforecastList.

YforecastList
A member list into which the forecast values are placed.

method
A choice among LR (linear regression), SES (single exponential smoothing), DES (double exponential smoothing), and TES (triple exponential smoothing). Method parameters must be numeric values, not member names. Method parameters may be any of the following:

- \textit{LR}[,\textit{t}]: standard linear regression with possible weights assigned to each data point and an optional seasonal adjustment period \([t]\), where \([t]\) is the length of the period. In general, the weights are equal to 1 by default. You might want to increase the weight if the corresponding observation is important, or decrease the weight if the corresponding observation is an outlier or is unreliable.

- \textit{SES}[,\textit{c}]: single exponential smoothing with parameter \textit{c} (default \(c=0.2\)). This method uses its own weight system, using the single parameter \textit{c}. Increasing this parameter gives more weight to early observations than to later ones.

- \textit{DES}[,\textit{c1},\textit{c2}]: double exponential smoothing (Holt’s method) with optional parameters \textit{c1}, \textit{c2} (default \(c1=0.2, c2=0.3\)). This is a two-parameter weight system and a linear subsequent approximation scheme. The first parameter controls weight distribution for the intercept; the second parameter controls weight distribution for the slope of the line fit.
- $\text{TES}[[T,c1,c2,c3]]$: triple exponential smoothing (Holt-Winters method) with optional parameters $c_1$, $c_2$, $c_3$, $T$ (default $c_1=0.2$, $c_2=0.05$, $c_3=0.1$, $T=1$). This is a three-parameter weight system and a linear model with a multiplicative seasonal component.

**Xfilter1 ... XfilterN**
Optional. Use one or more of the following filter methods to scale $Xlist$:
- $\text{XLOG}[c]$: logarithmic change with shift $c$ ($x' = \log(x+c)$) (default $c=1$)
- $\text{XEXP}[c]$: exponential change with shift $c$ ($x' = \exp(x+c)$) (default $c=0$).
- $\text{XPOW}[c]$: power change with power $c$ ($x' = x^c$) (default $c=2$).

**Yfilter1 ... YfilterN**
Optional. Use one or more of the following filter methods to scale $Ylist$:
- $\text{YLOG}[c]$: logarithmic change with shift $c$ ($y' = \log(y+c)$) (default $c=1$)
- $\text{YEXP}[c]$: exponential change with shift $c$ ($y' = \exp(y+c)$) (default $c=0$).
- $\text{YPOW}[c]$: power change with power $c$ ($y' = y^c$) (default $c=2$).

**Notes**
- @TREND can be used only in calculation scripts, not in outline formulas.
- You must associate the @TREND formula with a member.
- $Ylist$, $Xlist$, $weightList$, and $errorList$ should contain the same number of values.
- $XforecastList$ and $YforecastList$ should contain the same number of values.
- The method and filter parameters must be numbers only; functions and member names are not allowed.
- @TREND ignores #MISSING values during calculation of the trend.
- When you use the LR method with seasonal adjustments or when you use the TES method, Essbase places strict requirements on the input data. With these methods, input data cannot contain #MISSING values. Also, if you specify $Xlist$, the data must be equidistant, with the interval (step) being a whole fraction of the period, $T$ (for example, $T/5$, $T/2$). The $XforecastList$ parameters should also contain multiples of the interval.
- For another example using @TREND with more options, see Forecasting Future Values in Designing and Maintaining Essbase Cubes.
- If you use a member set function to generate a member list for this function, (for example, @SIBLINGS), to ensure correct results, consider the order in which Essbase sorts the generated member list. For more information, see the help topic for the member set function you are using.
- The following algorithms are used to calculate @TREND:

**Algorithm for Linear Regression**
$Y_{list} = y_1, y_2, \ldots, y_K$

$X_{list} = x_1, x_2, \ldots, x_K$

$weightList = w_1, w_2, \ldots, w_K$

**Linear Regression (LR)**

(if $w_i$ is \#MISSING or the whole $weightList$ is missing as an argument, $w_i = 1$)

\[
\begin{align*}
S &= \sum_{i=1}^{K} (w_i)^2 \\
S_x &= \sum_{i=1}^{K} x_i (w_i)^2 \\
S_y &= \sum_{i=1}^{K} y_i (w_i)^2 \\
S_{xx} &= \sum_{i=1}^{K} (x_i)^2 (w_i)^2 \\
S_{xy} &= \sum_{i=1}^{K} (x_i y_i)(w_i)^2 \\
\Delta &= SSS_{xx} - (S_x)^2 \\
\end{align*}
\]

\[
\begin{align*}
a &= \frac{S_{xx} S_y - S_x S_{xy}}{\Delta} \\
b &= \frac{SS_{xy} - S_x S_y}{\Delta} \\
\text{the equation of the line is}: \\
\text{line} &= Y_{list}(x) = a + bx
\end{align*}
\]

**Algorithm for Linear Regression with Seasonal Adjustment**
\[ Y_1 \quad Y_2 \quad Y_3 \quad Y_4 \quad Y_5 \quad Y_6 \]

In linear regressions, the intervals between \( x \) values must be the same. The value of that interval is \( \Delta \). In this case, \( \Delta = 1 \).

Step 1. Centered moving average of \( y \)'s, where \( n = 3 \) (moving centered average with 3 members at a time)

\[
\begin{align*}
Y_1 & \quad Y_2 & \quad Y_3 & \quad Y_4 & \quad Y_5 & \quad Y_6 \\
3 \times \frac{Y_1 + Y_2 + Y_3}{3} & \quad \frac{Y_2 + Y_3 + Y_4}{3} & \quad \frac{Y_3 + Y_4 + Y_5}{3} & \quad \frac{Y_4 + Y_5 + Y_6}{3}
\end{align*}
\]

\[ = \bar{Y}_2 \quad = \bar{Y}_3 \quad = \bar{Y}_4 \quad = \bar{Y}_5 \quad \text{centered moving average} \]

\[ Ylist \quad Y_1, Y_2, \ldots, Y_K \]

\[ Xlist \quad x_1, x_2, \ldots, x_K \]

\[ \text{weightList } w_1, w_2, \ldots, w_K \]

\[ \text{@TREND}(Ylist, \ldots, LR, t) \]

Linear regression with seasonal adjustment example:

There are 6 data points and a seasonal adjustment parameter, \( t=3 \)

Input data:

\[ x_1 = 1 \quad x_2 = 2 \quad x_3 = 3 \quad x_4 = 4 \quad x_5 = 5 \quad x_6 = 6 \]
Chapter 2
Calculation Function List

\[ y_1, y_2, y_3, y_4, y_5, y_6 \]

In linear regressions with seasonal adjustments, the intervals between \( x \) values must be the same. \( \Delta \) is equal to that interval. In this case, \( \Delta = 1 \).

Step 1. Centered moving average of \( y \)'s, where \( \kappa = 3 \) (moving centered average with 3 members at a time)

\[
\begin{align*}
\bar{y}_2 &= \frac{y_1 + y_3 + y_5}{3} \\
\bar{y}_3 &= \frac{y_2 + y_4 + y_6}{3} \\
\bar{y}_4 &= \frac{y_3 + y_5 + y_7}{3} \\
\end{align*}
\]

Step 2. Subtract \( \bar{y} \)'s from \( y \)'s:

\[
\begin{vmatrix}
\bar{y}_2 & \bar{y}_3 & \bar{y}_4 & \bar{y}_5 \\
\end{vmatrix}
\]

Step 3. Arrange \( \hat{y} \)'s into \( \kappa (\kappa = 3) \) columns to derive \( P \)'s and average values along columns:

\[
\begin{align*}
* & \hat{y}_2 & \hat{y}_3 \\
\hat{y}_4 & \hat{y}_5 & * \\
\hat{y}_6 & \hat{y}_7 & \hat{y}_8 \\
\end{align*}
\]

\[
\begin{align*}
1 & 2 & 1 \\
\end{align*}
\]

\[
= P_6 = P_1 = P_2 \quad \text{adjustment list}
\]
Algorithm for Single Exponential Smoothing (SES)

Step 4. Subtract $P$’s from original $Ylist$:

\[
\begin{array}{cccccccc}
Y_1 & Y_2 & Y_3 & Y_4 & Y_5 & Y_6 \\
\hline
P_0 & P_1 & P_2 & P_0 & P_1 & P_2 \\
Y'_1 & Y'_2 & Y'_3 & Y'_4 & Y'_5 & Y'_6 \\
\end{array}
\]

Step 5. Linear Regression (LR) with

\[
x_1 = 1 \quad x_2 = 2 \quad x_3 = 3 \quad x_4 = 4 \quad x_5 = 5 \quad x_6 = 6
\]

\[
Y'_1 \quad Y'_2 \quad Y'_3 \quad Y'_4 \quad Y'_5 \quad Y'_6
\]

as shown in **Linear Regression (LR) section**, deriving $a, b$ such that $y = bx + a$ is the trending line.

Step 6. To get future trend value for $x$:

\[
x: \quad Y_{\text{forecast}} = b \times x + a + P_i, \quad \text{where} \quad P_i: \quad i = \frac{(x - x_1) \mod t}{\Delta}
\]

\[
= \frac{(x - 1) \mod 3}{1}
\]

\[
= (x - 1) \mod 3
\]

Algorithm for Single Exponential Smoothing (SES)

\[
\begin{align*}
Ylist & \quad y_1, y_2, \ldots, y_K \\
Xlist & \quad x_1, x_2, \ldots, x_K
\end{align*}
\]
Algorithm for Double Exponential Smoothing (DES)

\[ c = 0.2 \text{ default, or else } c \text{ is input into the trend} \]

Find \( S_1, S_2, \ldots, S_K \):

\[ S_1 = y_1 \]

\[ S_{i+1} = a_i \times S_i + (1 - a_i) \times y_i \quad \text{for } i = 1, \ldots, K - 1 \]

Then \( Y_{\text{forecast}}(x) = a \times S_K + (1 - a) \times y_K \)

where \( a_i = (1 - c)^{y_{i+1} - y_i} \)

\[ a = (1 - c)^{x - x_k} \]

**Note:** When \( X_{\text{list}} \) is missing, \( x_i - x_{i-1} = l \) and the correspondent coefficients

\[ a_i = (l - c) \quad \text{for } i = 1, \ldots, K - 1 \]

Algorithm for Double Exponential Smoothing (DES)

\( Y_{\text{list}} \quad y_1, y_2, \ldots, y_K \)

\( X_{\text{list}} \quad x_1, x_2, \ldots, x_K \)
\( c_1 = .2, \quad c_2 = .3 \) default, or else they are input into the trend

find \( S_1, S_2, \ldots, S_K \)
\( b_1, b_2, \ldots, b_K \)

\[
S_1 = y_1
\]
\[
b_1 = \frac{(y_2 - y_1)}{(x_2 - x_1)}
\]

\[
S_{i+1} = a_i \times (S_i + b_i (x_{i+1} - x_i)) + (1 - a_i) \times (y_{i+1})
\]

\[
b_{i+1} = d_i \times b_i + (1 - d_i) \times \left[ \frac{(S_{i+1} - S_i)}{(x_{i+1} - x_i)} \right]
\]

where \( a_i = (1 - c_1) \times (x_{i+1} - x_i) \)
\( d_i = (1 - c_2) \times (x_{i+1} - x_i) \)

then \( Y_{\text{forecast}}(x) = S_K + (x - x_K) b_K \)

Note: When \( X_{\text{list}} \) is missing, \( x_{i+1} - x_i = 1 \) and the correspondent coefficients

\( a_i = (1 - c_1) \) for \( i = 1, \ldots, K - 1 \)
\( b_i = (1 - c_2) \)

Algorithm for Triple Exponential Smoothing (TES)

\( Y_{\text{list}} \quad y_1, y_2, \ldots, y_K \)

\( X_{\text{list}} \quad x_1, x_2, \ldots, x_K \)
TES with period $T$ (if $T$ is not given, it is assumed to be $T = 1$)

$x_1, x_2, \ldots, x_T, y_1, y_2, \ldots, y_T$ are input to TES, $x$ is forecast value.

\[ a_i = (1 - c)^{\frac{i}{T}} x \quad d_i = (1 - d)^{\frac{i}{T}} x \quad e_i = (1 - e)^{\frac{i}{T}} x \]

Note: When $Xlist$ is missing, $x_{i-1} - x_i = 1$ and the correspondent coefficients

\[ a_i = (1 - c) \quad \text{for } i = 1, \ldots, K - 1 \]

\[ d_i = (1 - d) \]

\[ e_i = (1 - e) \]

Default

\[ c = .2 \]
\[ d = .05 \]
\[ e = .1 \]

Step 1,

\[ S_1 = y_1 \]
\[ b_1 = \frac{y_2 - y_1}{x_2 - x_1} \]
\[ I_1 = 1 \]

Step 2, For $i = 1, \ldots, T - 1$

\[ S_{i+1} = a_i \ast (S_i + b_i \ast (x_{i+1} - x_i)) + (1 - a_i) \ast \frac{y_i}{I_i} \]

\[ I_{i+1} = \frac{y_i}{S_i} \]

\[ b_{i+1} = d_i b_i + (1 - d_i) \ast \frac{S_{i+1} - S_i}{x_{i+1} - x_i} \]
Step 3. For $i = 1, \ldots, K$

$$S_{i+1} = a_i \times (S_i + b_i (x_{i+1} - x_i)) + (1 - a_i) \frac{x_{i+1}}{I_{i+1-r}}$$

$$I_{i+1} = e_i I_{i+1-r} + (1 - e_i) \frac{x_{i+1}}{S_{i+1}}$$

$$b_{i+1} = d_i \beta_i + (1 - d_i) \frac{S_{i+1} - S_i}{x_{i+1} - x_i}$$

Forecast for $x$ is $(S_x + b_x (x - x_x))^* (I_x)^n$

where $j$ is determined by finding the maximum $j$, such that $x_j < x$ and then

$$m = \frac{x - x_j}{T}$$

Example

The following example is based on the Sample Basic database. It forecasts sales data for May through December, based on the trend of the same sales data from January through April. The method used is linear regression with no seasonal adjustment.

Sales(@TREND(Jan:Apr,,,May:Dec,LR));

This example produces the following report:

<table>
<thead>
<tr>
<th>Actual</th>
<th>Sales</th>
<th>West</th>
</tr>
</thead>
<tbody>
<tr>
<td>Colas</td>
<td></td>
<td></td>
</tr>
<tr>
<td>=====</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jan</td>
<td>2339</td>
<td></td>
</tr>
<tr>
<td>Feb</td>
<td>2298</td>
<td></td>
</tr>
<tr>
<td>Mar</td>
<td>2313</td>
<td></td>
</tr>
<tr>
<td>Apr</td>
<td>2332</td>
<td></td>
</tr>
<tr>
<td>May</td>
<td>2319</td>
<td></td>
</tr>
<tr>
<td>Jun</td>
<td>2318.4</td>
<td></td>
</tr>
<tr>
<td>Jul</td>
<td>2317.8</td>
<td></td>
</tr>
<tr>
<td>Aug</td>
<td>2317.2</td>
<td></td>
</tr>
<tr>
<td>Sep</td>
<td>2316.6</td>
<td></td>
</tr>
<tr>
<td>Oct</td>
<td>2316</td>
<td></td>
</tr>
<tr>
<td>Nov</td>
<td>2315.4</td>
<td></td>
</tr>
<tr>
<td>Dec</td>
<td>2314.8</td>
<td></td>
</tr>
<tr>
<td>Year</td>
<td>27817.2</td>
<td></td>
</tr>
</tbody>
</table>

See Also

@LIST
@TRUNCATE

Removes the fractional part of expression, returning the integer.

Syntax

@TRUNCATE (expression)

Parameters

expression
Single member specification, function, variable name, or other numeric expression, which returns a numeric value.

Example

In the following example, Total Sales is calculated by (1) taking the sum of the values for Direct Sales and Other Sales and (2) truncating the summed values.

"Total Sales" = @TRUNCATE(@SUM("Direct Sales":"Other Sales"));

This example produces the following report:

<table>
<thead>
<tr>
<th></th>
<th>Colas</th>
<th>New York</th>
<th>Actual</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Jan</td>
<td>Feb</td>
<td>Mar</td>
</tr>
<tr>
<td>Direct Sales</td>
<td>678.557</td>
<td>645.874</td>
<td>675.299</td>
</tr>
<tr>
<td>Other Sales</td>
<td>411.299</td>
<td>389.554</td>
<td>423.547</td>
</tr>
<tr>
<td>Total Sales</td>
<td>1089</td>
<td>1035</td>
<td>1098</td>
</tr>
</tbody>
</table>

See Also

- @REMAINDER
- @ROUND

@UDA

Returns members based on a common attribute, which you have defined as a user-defined attribute (UDA) on the Essbase Server.

Syntax

@UDA (dimName, uda)

Parameters

dimName
Name of the dimension with which the uda is associated.
uda
Name of the user-defined attribute as it appears in the database outline.

Notes
You must type the UDA string exactly as it appears in the database outline.

Example
In the Sample Basic database:

@UDA(Market, "New Mkt")

Returns a list of members with the UDA of New Mkt.

See Also
• @ISUDA
• @ISMBRUDA

@VAR
Calculates the variance (difference) between two members. The variance calculation recognizes the difference between accounts that are tagged in the database outline as expense and those that are non-expense (the default), and calculates the variance accordingly.

Syntax
@VAR (mbrName1, mbrName2)

Parameters
mbrName1 and mbrName2
Members from the same dimension whose variance results are to be calculated. The variance is derived by subtracting mbrName2 values from mbrName1, unless an account is tagged as expense, in which case mbrName1 values are subtracted from mbrName2.

Example
The following example is based on the Sample Basic database. The variance between Actual and Budget is calculated as follows:

Variance = @VAR(Actual,Budget);

Sales is non-expense, whereas COGS is expense. This example produces the following report:

<table>
<thead>
<tr>
<th>Year</th>
<th>Product</th>
<th>Market</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sales</td>
<td>COGS</td>
<td>=====</td>
</tr>
<tr>
<td></td>
<td></td>
<td>=====</td>
</tr>
<tr>
<td>Actual</td>
<td>400855</td>
<td>179336</td>
</tr>
</tbody>
</table>
Budget            373080     158940
Variance           27775    (20396)

See Also
• @VARPER
• @VARIANCE
• @VARIANCEP

@VARPER

Calculates the percent variance (difference) between two members. The variance calculation recognizes the difference between accounts that are tagged in the database outline as expense and those that are non-expense, and calculates the variance accordingly.

Syntax
@VARPER (mbrName1, mbrName2)

Parameters

mbrName1 and mbrName2
Members from the same dimension whose variance results are to be calculated. The percent variance is derived by taking the percent variance of mbrName2 values from mbrName1, unless an account is tagged as expense, in which case mbrName1 values are taken as a percent variance of mbrName2.

Example

The following example is based on the Sample Basic database. The percent variance between Actual and Budget is calculated as follows:

Variance % = @VARPER(Actual,Budget);

In this example Sales is non-expense, whereas COGS is expense. This example produces the following report:

<table>
<thead>
<tr>
<th>Year</th>
<th>Product</th>
<th>Market</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Sales</td>
<td>COGS</td>
</tr>
<tr>
<td>==========</td>
<td>========</td>
<td>=======</td>
</tr>
<tr>
<td>Actual</td>
<td>400855</td>
<td>179336</td>
</tr>
<tr>
<td>Budget</td>
<td>373080</td>
<td>158940</td>
</tr>
<tr>
<td>Variance %</td>
<td>7.4</td>
<td>(12.8)</td>
</tr>
</tbody>
</table>

See Also
• @VAR
• @VARIANCE
• @VARIANCEP
@VARIANCE

Calculates the statistical variance of the specified data set. The calculation is based upon a sample of a population. Variance is a measure of the dispersion of a set of data points around their mean (average) value.

Syntax

@VARIANCE (SKIPNONE | SKIPMISSING | SKIPZERO | SKIPBOTH, XrangeList)

Parameters

SKIPNONE
Includes all cells specified in the data set, regardless of their content, during calculation of the variance.

SKIPMISSING
Excludes all #MISSING values from the data set during calculation of the variance.

SKIPZERO
Excludes all zero (0) values from the data set during calculation of the variance.

SKIPBOTH
Excludes all zero (0) values and #MISSING values from the data set during calculation of the variance.

XrangeList
A list of numeric values across which the variance is calculated. Referred to generically throughout this topic as "the data set."

Can be a valid member name, a comma-delimited list of member names, cross dimensional members, or a return value from a member set function or range function (including @XRANGE).

For more information about XrangeList, see Range List Parameters.

Notes

- @VARIANCE is different from @VAR, which calculates the variance (difference) between two members.
- @VARIANCE assumes that the data set (XrangeList) represents a sample of the population. If you want the data set to represent the entire population, use @VARIANCEP.
- @VARIANCE is calculated with the "unbiased" or "n-1" method.
- @VARIANCE uses the following formula:

\[ \frac{n \sum x^2 - \left( \sum x \right)^2}{n(n - 1)} \]
Example

The following example is based on the Sample Basic database. Assume that the Measures dimension contains an additional member, Sales Var. This example uses the @RANGE function to generate the data set, and calculates the variance of the sales values for a product family.

```
FIX (Product)
"Sales Var" = @VARIANCE(SKIPBOTH, @RANGE(Sales, @CHILDREN(Product)));
ENDFIX
```

This example produces the following report:

<table>
<thead>
<tr>
<th></th>
<th>Jan</th>
<th>New</th>
</tr>
</thead>
<tbody>
<tr>
<td>New</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Actual</td>
<td></td>
<td></td>
</tr>
<tr>
<td>======</td>
<td>====</td>
<td>====</td>
</tr>
<tr>
<td>Sales</td>
<td>678</td>
<td>640</td>
</tr>
<tr>
<td>Root Beer</td>
<td>551</td>
<td>530</td>
</tr>
<tr>
<td>Cream Soda</td>
<td>663</td>
<td>510</td>
</tr>
<tr>
<td>Fruit Soda</td>
<td>587</td>
<td>620</td>
</tr>
<tr>
<td>Diet Drinks</td>
<td>#MI</td>
<td>#MI</td>
</tr>
<tr>
<td>Product</td>
<td>2479</td>
<td>2300</td>
</tr>
</tbody>
</table>

Sales Var Product 3687.58 4166.67

The following example assumes a Year dimension is added to Sample Basic. It calculates variance using cross-dimensional members in the data set.

```
FIX(Product)
"Sales Var" = @VARIANCE(SKIPBOTH, @XRANGE("2011"->"Sep", "2012"->"Mar"));
ENDFIX
```

The above calculation is performed across the following multidimensional range specified by XrangeList:

2011->Sep
2011->Oct
2011->Nov
2011->Dec
2012->Jan
2012->Feb
2012->Mar

See Also

@VARIANCEP
@VARIANCEP

Calculates the statistical variance of the specified data set. The calculation is based upon the entire population. Variance is a measure of the dispersion of a set of data points around their mean (average) value.

Syntax

@VARIANCEP (SKIPNONE | SKIPMISSING | SKIPZERO | SKIPBOTH, XrangeList)

Parameters

SKIPNONE
Includes all cells specified in the data set, regardless of their content, during calculation of the variance.

SKIPMISSING
Excludes all #MISSING values from the data set during calculation of the variance.

SKIPZERO
Excludes all zero (0) values from the data set during calculation of the variance.

SKIPBOTH
Excludes all zero (0) values and #MISSING values from the data set during calculation of the variance.

XrangeList
A list of numeric values across which the variance is calculated. Referred to generically throughout this topic as "the data set."
Can be a valid member name, a comma-delimited list of member names, cross dimensional members, or a return value from a member set function or range function (including @XRANGE).
For more information about XrangeList, see Range List Parameters.

Notes

- @VARIANCEP is different from @VARPER, which calculates the percent variance (difference) between two members.
- @VARIANCEP assumes that the data set (XrangeList) represents the entire population. If you want the data set to represent a sample of the population, use @VARIANCE.
- @VARIANCEP is calculated using the "biased" or "n" method.
- @VARIANCEP uses the following formula:

\[
\frac{n \sum x^2 - (\sum x)^2}{n^2}
\]
Example

The following example is based on the Sample Basic database. Assume that the Measures dimension contains an additional member, Sales Var. This example uses the @RANGE function to generate the data set, and calculates the variance of the sales values for a product family.

```
FIX (Product)
"Sales Var" = @VARIANCEP(SKIPBOTH,@RANGE(Sales,@CHILDREN(Product)));
ENDFIX
```

This example produces the following report:

<table>
<thead>
<tr>
<th></th>
<th>Jan</th>
<th>New York</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Actual</td>
<td>Budget</td>
</tr>
<tr>
<td>Sales</td>
<td>======</td>
<td>======</td>
</tr>
<tr>
<td>Colas</td>
<td>678</td>
<td>640</td>
</tr>
<tr>
<td>Root Beer</td>
<td>551</td>
<td>530</td>
</tr>
<tr>
<td>Cream Soda</td>
<td>663</td>
<td>510</td>
</tr>
<tr>
<td>Fruit Soda</td>
<td>587</td>
<td>620</td>
</tr>
<tr>
<td>Diet Drinks</td>
<td>#MI</td>
<td>#MI</td>
</tr>
<tr>
<td>Product</td>
<td>2479</td>
<td>2300</td>
</tr>
</tbody>
</table>

Sales Var | Product | 2765.69 | 3125

The following example assumes a Year dimension is added to Sample Basic. It calculates variance using cross-dimensional members in the data set.

```
FIX(Product)
"Sales Var" = @VARIANCEP(SKIPBOTH,@XRANGE("2011"->"Sep", "2012"->"Mar"));
ENDFIX
```

The above calculation is performed across the following multidimensional range specified by XrangeList:

- 2011->Sep
- 2011->Oct
- 2011->Nov
- 2011->Dec
- 2012->Jan
- 2012->Feb
- 2012->Mar

See Also

@VARIANCE
@WEIGHTEDSUMX

Aggregates all members in a member list, depending on the unit weight of each member, which is fetched from a remote data source. @WEIGHTEDSUMX improves the performance of aggregating currency databases by calling the calculation framework only once.

The following terminology is used to describe this function:

• Data target: the database on which the current calculation is running (that is, the database on which the @WEIGHTEDSUMX call originates).
• Data source: the database that is queried by @WEIGHTEDSUMX. This database may be remote (that is, on a different machine than the data target).
• Point of view: the member combination currently being calculated on the data target (that is, the member combination that identifies the left hand side of a calculation).

Syntax

There are multiple ways to call this function, depending on your goal.

To incorporate values from a remote cube, use this syntax:

@WEIGHTEDSUMX (mbrList, locationAlias [, CurrencyType, CurrencyRate, Period])

The mbrList and locationAlias parameters are required. If the other parameters are not provided, they are taken from the POV.

To incorporate values from another application and database on the same Essbase server instance, use this syntax:

@WEIGHTEDSUMX (mbrList, appname, dbname [, CurrencyType, CurrencyRate, Period])

Parameters

mbrList

Required. Specifies the list of members to be aggregated according to the unit weight of the individual members. The mbrList can be a calculation function that returns a member list or a comma-separated list of member names. The member list cannot contain functions that return more than one member.

Examples of functions that return a member list: @CHILDREN, @DESCENDANTS, and @RANGE.

A comma-separated list of member names must be expressed as a single argument. For example, a list of currencies such as "USD", "ARS", "AUD", "BRL" can be used with a member list function, as in @LIST ("USD", "ARS", "AUD", "BRL"), or expressed as a range if the members are at the same level, as in "USD::BRL", or enclosed in parentheses, as in ("USD", "ARS", "AUD", "BRL").

The members you specify for mbrList are sent to the data source in addition to the members in the current point of view in the data target. The data source then constructs a member combination, using in order of precedence:
• The members specified in mbrList
• The members in the current point of view
• The top member in any unspecified dimensions in the data source

The following formula modifies the point of view on the data target. Assume that the cube on the data source (sourceDB) contains data only from 2002. This formula sets Inventory for Jan 2003 to the Inventory value for Dec 2002:

$$2003(2003\rightarrow Jan\rightarrow Inventory = @WEIGHTEDSUMX (mbrList, locationAlias, Dec))$$

The following formula defines a specific point of view on the data target. Assume that the data target contains the member Jan and the data source (locationAlias) contains the member January. This formula maps the member in the data target (Jan) with its corresponding member in the data source (January), and pulls January from data source:

$$Jan = @WEIGHTEDSUMX (mbrList, locationAlias, January);$$

The following formula is an example of using @RANGE with a comma-separated list of members, which includes a range of members at the same level:

$$@WEIGHTEDSUMX(@RANGE("Entered","USD":"ZAR"), _FCCS_Rates_, "Rate.Average", "Rate_USD");$$

**locationAlias**
Required. A location alias for the data source. A location alias is a descriptor that identifies the data source. The location alias must be set on the database on which the calculation script will be run. The location alias is set by the database administrator and specifies a server, application, database, user name, and password for the data source.

**appname, dbname**
Application and cube name. Use only for deployments with only one Essbase server instance.

**CurrencyType**
Optional. A member in a dimension that contains currency types, with members such as Average, Closing, or Historical.

**CurrencyRate**
Optional. A member in a dimension that contains currency rates, with members depicting the global currency rates.

**Period**
Optional. A member from a time dimension.

**Notes**
• You must be signed in on the data target, and also provisioned on the data source.
• An error is returned if the members supplied in mbrList do not exist in the data source.
• The number of data cells queried on the data source must match the number of data cells expected on the data target.
The member list cannot contain functions that return more than one member.
The member list cannot contain functions that return more than one member.

Only one parameter can be provided per dimension.

@WITHATTR

Returns all base members that are associated with an attribute or varying attribute that satisfies the conditions you specify. You can use operators such as >, <, =, and IN to specify conditions that must be met. This function can be used as a parameter of another function, where that parameter is a list of members.

Syntax

@WITHATTR (dimName, "operator", value)

Parameters

dimName
Single attribute dimension name or varying attribute dimension name.

operator
Operator specification, which must be enclosed in quotation marks ("").

value
A value that, in combination with the operator, defines the condition that must be met. The value can be an attribute member specification, a constant, or a date-format function (@TODATE).

Notes

• A varying attribute cannot be included in a FIX command if no perspective is specified in the calculation script.

• @WITHATTR is a superset of @ATTRIBUTE. The following two formulas return the same member set:

@ATTRIBUTE(Bottle)
@WITHATTR("Pkg Type","==",Bottle)

However, the following formula can be performed only with @WITHATTR (not with @ATTRIBUTE) because you specify a condition:

@WITHATTR(Ounces,">","16")

• If you specify a date attribute with @WITHATTR, you must use @TODATE in the string parameter to convert the date string to a number.

The following operators are supported:

Table 2-34  Supported Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than or equal to</td>
</tr>
</tbody>
</table>
Table 2-34  (Cont.) Supported Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equal to</td>
</tr>
<tr>
<td>= =</td>
<td>Equal to</td>
</tr>
<tr>
<td>&lt;&gt; or !=</td>
<td>Not equal to</td>
</tr>
<tr>
<td>IN</td>
<td>In</td>
</tr>
</tbody>
</table>

- The IN operator returns the base members that are associated with a subcategory of attributes in the attribute dimension. For example, in the Sample Basic database, @WITHATTR(Population,"IN",Medium) returns the base members that are associated with all attributes under the Medium parent member in the Population dimension.

- When using Boolean attributes with @WITHATTR, use only the actual Boolean attribute member name, or use 1 (for True or Yes) or 0 (for False or No). You cannot use True/Yes and False/No interchangeably.

- An operator may work differently with different attribute types. For example:
  - **Text**—@WITHATTR(Flavors,"<",Orange) returns base members with attributes that precede Orange in the alphabet; for example, Apple, Cranberry, Mango, and Oat, but not Peach or Strawberry.
  - **Boolean**—@WITHATTR(Caffeinated,"<",True) returns all base members that have Caffeinated set to False (or No). It does not return base members that do not have Caffeinated set to True (or Yes) or do not have a Caffeinated attribute at all. The behavior is similar for a formula like @WITHATTR(Caffeinated,"<>",True), which returns only base members with Caffeinated set to False.
  - **Date**—@WITHATTR("Intro Date","<",@TODATE("mm-dd-yyyy","07-26-2002")) returns all base members with date attributes that are before July 26, 2002.

**Example**

The following table shows examples, based on the Sample Basic database, for each type of operator:

Table 2-35  Operator Results

<table>
<thead>
<tr>
<th>Operator</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;</td>
<td>@WITHATTR(Population,&quot;&gt;&quot;,&quot;180000000&quot;)</td>
<td>Returns New York, California, and Texas</td>
</tr>
<tr>
<td>&gt;=</td>
<td>@WITHATTR(Population,&quot;&gt;=&quot;,10000000) where 10,000,000 is not a numeric attribute member, but a constant</td>
<td>Returns New York, Florida, California, Texas, Illinois, and Ohio</td>
</tr>
<tr>
<td>&lt;</td>
<td>@WITHATTR(Ounces,&quot;&lt;&quot;,&quot;16&quot;)</td>
<td>Returns Cola, Diet Cola, Old Fashioned, Sasparilla, and Diet Cream</td>
</tr>
<tr>
<td>&lt;=</td>
<td>@WITHATTR(&quot;Intro Date&quot;,&quot;&lt;=&quot;,@TODATE(&quot;mm-dd-yyyy&quot;,&quot;04-01-2002&quot;))</td>
<td>Returns Cola, Diet Cola, Caffeine Free Cola, and Old Fashioned</td>
</tr>
</tbody>
</table>
### Table 2-35  (Cont.) Operator Results

<table>
<thead>
<tr>
<th>Operator</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>= =</td>
<td>@WITHATTR(&quot;Pkg Type&quot;,&quot;= &quot;,Can)</td>
<td>Returns Cola, Diet Cola, and Diet Cream</td>
</tr>
<tr>
<td>&lt;&gt; or !=</td>
<td>@WITHATTR(Caffeinated,&quot;&lt;&gt;&quot;,True)</td>
<td>Returns Caffeine Free Cola, Sasparilla, Birch Beer, Grape, Orange Strawberry</td>
</tr>
<tr>
<td>IN</td>
<td>@WITHATTR(&quot;Population&quot;,&quot;IN&quot;,Medium)</td>
<td>Returns Massachusetts, Florida, Illinois, and Ohio</td>
</tr>
</tbody>
</table>

The following two examples show @WITHATTR used in a calculation script, based on the Sample Basic database:

/* To increase by 10% the price of products that are greater than or equal to 20 ounces */

```plaintext
FIX (@WITHATTR(Ounces,">","20"));
Price = Price * 1.1;
ENDFIX
```

/* To increase by 10% the marketing budget for products brought to market after a certain date */

```plaintext
FIX (@WITHATTR("Intro Date",">",
@TODATE("mm-dd-yyyy","06-26-1996");)
Marketing = Marketing * 1.1;
ENDFIX
```

**See Also**
- @ATTRIBUTE
- @ATTRIBUTEVAL
- SET SCAPERSPECTIVE
- @TODATE

**@XRANGE**

Returns the range of members between (and inclusive of) two specified single or cross-dimensional members at the same level.

For example, when you work with the Time and Scenario dimensions, you can use @XRANGE to return a member set combination of Time and Scenario instead of creating a dimension that combines the two (which creates many more individual members than necessary).

@XRANGE is a member set function. Member set functions return a list of members. @XRANGE can appear anywhere in a formula where a range can normally appear.
Syntax

@XRANGE (mbrName1, mbrName2)

Parameters

mbrName1
Any valid member name, member combination, or function that returns a single member.

mbrName2
Any valid member name, member combination, or function that returns a single member. If mbrName1 is a cross-dimensional member (such as Actual->Jan), then mbrName2 must be also, and the dimension order must match the order used in mbrName1.

Notes

• The two arguments to @XRANGE can be either both single members or both cross-dimensional members. For example, @XRANGE (Actual->Jan, Budget) is invalid because a single member and a cross dimensional member are used together. Both @XRANGE(Actual->Jan, Budget->Feb) and @XRANGE(Jan, Mar) are valid.

• The dimension order of members must match for both arguments. For example, @XRANGE(Actual->Jun, Jul->Budget) is invalid because the two member components are in different orders. @XRANGE(Actual->Jun, Budget->Jul) is valid.

• Although the syntax is correct, a function such as @XRANGE (Dec, Mar) is meaningless because it results in an empty set.

• The member components of each argument must be from the same level. For example, @XRANGE (Actual->Jun, Budget->Qtr1) is invalid because Jun and Qtr1 are not from the same level.

Example

The following examples are based on the Sample Basic database.

Example 1

Here is a very simple example using simple members to return the range between Jan and Mar.

@XRANGE (Jan, Mar)

This example returns the following members:

Jan
Feb
Mar
Example 2

Here is a very simple example using cross dimensional members to return the range between Actual, Jan and Budget, Mar:

@XRANGE (Actual->Jan, Budget->Mar)

This example returns the following members:

- Actual, Jan
- Actual, Feb
- Actual, Mar
- Actual, Apr
- Actual, May
- Actual, Jun
- Actual, Jul
- Actual, Aug
- Actual, Sep
- Actual, Oct
- Actual, Nov
- Actual, Dec
- Budget, Jan
- Budget, Feb
- Budget, Mar

Example 3

This example is not based on the Sample Basic database. It is based on database that contains a dimension called Year that contains members for each year, from 2001 to 2003.

The following formula computes the average sales for all months between Mar of 2000 and Jan of 2001.

SalesAvg = @MOVAVG(Sales, 3, @XRANGE("2000"->Mar, "2001"->Jan));

This example returns the following members:

<table>
<thead>
<tr>
<th>Colas</th>
<th>New York Sales</th>
<th>Actual SalesAvg</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mar</td>
<td>678</td>
<td>678</td>
</tr>
<tr>
<td>Apr</td>
<td>645</td>
<td>645</td>
</tr>
<tr>
<td>May</td>
<td>675</td>
<td>666</td>
</tr>
<tr>
<td>Jun</td>
<td>712</td>
<td>677.3</td>
</tr>
<tr>
<td>Jul</td>
<td>756</td>
<td>714.3</td>
</tr>
<tr>
<td>Aug</td>
<td>890</td>
<td>786</td>
</tr>
<tr>
<td>Sep</td>
<td>924</td>
<td>856.7</td>
</tr>
<tr>
<td>Oct</td>
<td>914</td>
<td>909.3</td>
</tr>
<tr>
<td>Nov</td>
<td>912</td>
<td>916.7</td>
</tr>
<tr>
<td>Dec</td>
<td>723</td>
<td>849.7</td>
</tr>
</tbody>
</table>
@XREF

Enables a database calculation to incorporate values from another Essbase database.

The following terminology is used to describe @XREF:

- **Data target**: the database on which the current calculation is running (that is, the database on which the @XREF call originates).
- **Data source**: the database that is queried by @XREF. This database may be remote (that is, on a different machine than the data target).
- **Point of view**: the member combination currently being calculated on the data target (that is, the member combination that identifies the left hand side of a calculation).

The @XREF function retrieves values from a data source to be used in a calculation on a data target. @XREF does not impose member and dimension mapping restrictions, which means that the data source and data target outlines can be different.

**Syntax**

There are multiple ways to call this function, depending on your goal.

To incorporate values from a remote cube, use:

```plaintext
@XREF (locationAlias [, mbrList])
```
To incorporate values from another application and database on the same Essbase server instance, use:

@XREF (appname, dbname [, mbrList])

Parameters

**locationAlias**

A location alias for the data source. A location alias is a descriptor that identifies the data source. A location alias is not needed if the source and target are on the same Essbase server instance.

If used, the location alias must be set on the database on which the calculation script will be run. The location alias is set by the database administrator and specifies a server, application, database, user name, and password for the data source.

**mbrList**

Optional. A comma-delimited list of member names that qualify the @XREF query. The members you specify for mbrList are sent to the data source in addition to the members in the current point of view in the data target. The data source then constructs a member combination, using in order of precedence:

- The members specified in mbrList
- The members in the current point of view
- The top member in any unspecified dimensions in the data source

The mbrList parameter (1) modifies the point of view on the data target or (2) defines a specific point of view on the data source. For example, the following formula modifies the point of view on the data target:

```
2003(2003->Jan->Inventory = @XREF(sourceDB,Dec));
```

If the cube on the data source (sourceDB) contains data only from 2002, this formula sets Inventory for Jan in 2003 to the Inventory value for Dec from 2002. The following formula defines a specific point of view on the data target:

```
Jan = @XREF(sourceDB,January);
```

Assume that the data target contains the member Jan, while the data source (sourceDB) contains the member January. This formula simply maps the member in the data target (Jan) with its corresponding member in the data source (January), and pulls January from sourceDB.

See Notes for more information about the mbrList parameter.

**appname, dbname**

Application and cube name. Use only for deployments with only one Essbase server instance.

**Notes**

- You must be signed in on the data target, and also provisioned on the data source.
- An error is returned if the members supplied in mbrList do not exist in the data source.
• The number of data cells queried on the data source must match the number of data cells expected on the data target.

• The member list cannot contain functions that return more than one member. For example, the following formula is not valid:

\[ \text{West} = \text{@XREF(SourceDb, @LEVMBRS(Market, 0))} \]

• The member list cannot contain ranges. For example, the following formula is not valid:

\[ \text{West} = \text{@XREF(SourceDb, Jan:Mar)} \]

• \( mbrList \) can contain attribute members. For example, if the data source classifies products based on a color attribute, the following formula would calculate the sum of the sales of all red products and would assign the result to member RedThings:

\[ \text{RedThings} = \text{@XREF(SourceDb, Sales, Red)} \]

• \( mbrList \) can contain attribute operators. For example, the following formula calculates RedThings as the average sales of all red products:

\[ \text{RedThings} = \text{@XREF(SourceDb, Sales, Red, Average)} \]

• \text{@XREF} can query all types of members. For example, members retrieved from a data source can be Dynamic Calc members as well as attribute members. Keep in mind that all performance considerations that apply to dynamic and attribute calculations also apply to \text{@XREF} queries that depend on dynamic and attribute members.

• Over the course of an \text{@XREF} calculation, data in the source database may change. \text{@XREF} does not incorporate changes made after the beginning of the calculation.

• \text{@XREF} is a top-down formula. For more information on top-down formulas, see Bottom-Up and Top-Down Calculation in Designing and Maintaining Essbase Cubes.

• For a member that does not exist in either the data source or the data target, \text{@XREF} returns the value of the top dimension, not the value #M1.

• If you are using \text{@PARENT} within \text{@XREF}, it must be within \text{@NAME}. For example:

\[ \text{COGS} = \text{@XREF(Sample, \text{@NAME(@PARENT(Product))}, Sales)} \]

• When running a parallel calculation that includes \text{@XREF}, the application times out if the number of threads you specify to use is higher than the number of \text{SERVERTHREADS}. For example, the default value of \text{SERVERTHREADS} is 20. If you set \text{CALCPARALLEL} to 25, an application timeout error is generated.

Example

For this example, consider the following two databases:
Main Database

Year
  Qtr1
  Qtr2
Measures
  Sales
  Units
Product
  100
    100-10
    100-20
Market
  East
  West
Scenario
  Budget
  Forecast

Inflation Rates Database

Year
  Qtr1
  Qtr2
Assumptions
  Inflation
    Deflation = Inflation * .5 (Dynamic Calc)
Country
  US
  Canada
  Europe

The following formula is associated with the Main Database:

Units = Units * @XREF(InflatDB,Inflation,US);

Where InflatDB is the location alias for the Inflation Rates Database and Inflation is the member for which a data value is retrieved from InflatDB.

In this example, Essbase calculates the following member combinations:

Units->Qtr1->100-10->East->Budget = Units->Qtr1->100-10->East->Budget * Inflation->Qtr1->US

Units->Qtr2->100-10->East->Budget = Units->Qtr2->100-10->East->Budget * Inflation->Qtr2->US and so on.

See Also

• SERVERTHREADS
• @XWRITE
@XWRITE

Enables a database calculation to write values to another Essbase database, or to the same database.

The following terminology is used to describe the @XWRITE function:

- **Data source**: the database on which the current calculation is running (that is, the database on which the @XWRITE call originates).
- **Data target**: the database that is updated by @XWRITE. This database may be remote (that is, on a different machine than the data source).
- **Point of view**: the member combination currently being calculated on the data source.

This function writes to data blocks, either in the same database or in a remote database, while calculating a block in the current database. @XWRITE does not impose member and dimension mapping restrictions, which means that the data source and data target outlines can be different.

As arguments, this function takes a location alias, an implied list of members that represents the current point of view, and an optional list of members to qualify @XWRITE on the data target. The second argument (the members making up the current point of view) is implied; that is, these members are not specified as an @XWRITE parameter. An @XWRITE that omits the third argument indicates that a given data point in the data source will be set to the same data point in the data target.

Syntax

There are multiple ways to call this function, depending on your goal.

To incorporate values from a remote cube, use:

```
@XWRITE (expression, locationAlias [, mbrList])
```

To incorporate values from another application and database on the same Essbase server instance, use:

```
@XWRITE (expression, appname, dbname [, mbrList])
```

Parameters

**expression**

A single member specification, variable name, or other numeric expression corresponding to the value to be stored.

**locationAlias**

A location alias for the data target. A location alias is not needed if the source and target are on the same Essbase server instance.

If used, the location alias must be set on the database on which the calculation script will be run. The location alias is set by the database administrator and specifies a server, application, database, username, and password for the data target. The same location alias can be used by both @XREF and @XWRITE. For @XREF, it represents the data source, and for @XWRITE it represents the data target.
For @XWRITE only, a reserved keyword @LOOPBACK can be used to write to the same database.

**mbrList**
Optional. A comma-delimited list of member names that qualify the @XWRITE operation. The members you specify for mbrList, in addition to the members in the current point of view in the data source, determine what is written to the data target. The data target is written to using the following calculation logic (in order of precedence):

- The members specified in mbrList
- The members in the current point of view
- The top member in any unspecified dimensions in the data target

Therefore, the remote member list is calculated and written using members from current point of view, overridden with members from the mbrList specified to @XWRITE, and if some dimensions are still absent at the data target, the top most dimension of the data target is used. See Notes for more information about the mbrList parameter.

**appname, dbname**
Application and cube name. Use only for deployments with only one Essbase server instance.

**Notes**
- You must be signed in on the data target, and also provisioned on the data source.
- This function is applicable only to block storage databases.
- An error is returned if the members supplied in mbrList do not exist in the data target.
- The member list cannot contain functions that return more than one member. For example @LEVMBRS(Market,0).
- The member list cannot contain ranges.
- The member list cannot contain attribute members or attribute operators.
- @XWRITE is a top-down formula. For more information on top-down formulas, see Bottom-Up and Top-Down Calculation in Designing and Maintaining Essbase Cubes.
- @XWRITE to dynamic calc cells is not recommended; the data is calculated in memory, but not written.
- @XWRITE can be used in calculation scripts as well as outline member formulas.

**Example**
The following Sample Basic formula writes the 100-30 values into 100-20 on the same database.

```plaintext
FIX (East, Actual, Budget, Sales)
"100-30" { @XWRITE("100-30", @loopback, "100-20");
```
The following Sample Basic formula writes the 100-30 values into 100-20 on a remote database, Sample2 Basic, using the location alias "sam2basic" defined from Sample Basic to Sample2 Basic.

```plaintext
FIX (East, Actual, Budget, Sales)
"100-30" {
@XWRITE("100-30", sam2basic, "100-20");
}
ENDFIX
```

The following example shows how to call another function within the @XWRITE function call.

```plaintext
FIX (East, Actual, Budget, Sales)
"100"  {
@XWRITE(@PARENT("100-30"), @loopback, "100-20");
}
ENDFIX
```

See Also
@XREF

Functions Supported in Hybrid Aggregation Mode

The Essbase configuration setting ASODYNAMICAGGINBSO controls whether block storage databases use hybrid aggregation mode. Hybrid aggregation for block storage databases means that wherever possible, block storage data calculation executes with efficiency similar to that of aggregate storage databases. If enabled, hybrid aggregation is supported for member formulas using any of functions in this group.

- @ABS
- @ACCUM
- @ALLANCESTORS
- @ALIAS
- @ANCEST
- @ANCESTORS
- @ANCESTVAL
- @ATTRIBUTE
- @ATTRIBUTEBVAL
- @ATTRIBUTESVAL
- @ATTRIBUTEVAL
- @AVG
• @AVGRANGE
• @BETWEEN
• @CALCMODE
• @CHILDREN
• @COMPOUND
• @COMPOUNDGROWTH
• @CONCATENATE
• @CORRELATION
• @COUNT
• @CURGEN
• @CURLEV
• @CURRMBR
• @CURRMBRRANGE
• @DATEDIFF
• @DATEPART
• @DATEROLL
• @DECLINE
• @DESCENDANTS
• @DISCOUNT
• @ENUMVALUE
• @EQUAL
• @EXP
• @EXPAND
• @FACTORIAL
• @FORMATDATE
• @GEN
• @GENMBRS
• @GROWTH
• @IALLANCESTORS
• @IANCESTORS
• @ICHOILDREN
• @IDESENDANTS
• @ILANCESTORS
• @ILDSENDANTS
• @ILSIBLINGS
• @INT
• @INTEREST
• @INTERSECT
• @IRDESCENDANTS
• @IRR
• @IRSIBLINGS
• @ISACCTYPE
• @ISANCEST
• @ISATTRIBUTE
• @ISCHILD
• @ISDESC
• @ISGEN
• @ISANCEST
• @ISANCEST
• @ISIBLINGS
• @ISICHILD
• @ISIDESC
• @ISIPARENT
• @ISISIBLING
• @ISLEV
• @ISMBR
• @ISMBRUDA
• @ISMBRWITHATTR
• @ISPARENT
• @ISRANGENONEMPTY
• @ISSAMEGEN
• @ISSAMELEV
• @ISSIBLING
• @ISUDA
• @LANCESTORS
• @LDESCENDANTS
• @LEV
• @LEVMBRS
• @LIKE
• @LIST
• @LN
• @LOG
• @LOG10
• @LSIBLINGS
• @MATCH
• @MAX
• @MAXRANGE
• @MAXS
• @MAXSRANGE
• @MBRCOMPARE
• @MBRPARENT
• @MDANCESTVAL
• @MDPARENTVAL
• @MEDIAN
• @MEMBER
• @MEMBERAT
• @MERGE
• @MIN
• @MINRANGE
• @MINS
• @MINSRANGE
• @MOD
• @MODE
• @MOVAVG
• @MOVMAX
• @MOVMED
• @MOVMIN
• @MOVSUM
• @NAME
• @NEXT
• @NEXTS
• @NEXTSIBLING
• @NOTEQUAL
• @NPV
• @PARENT
• @PARENTVAL
• @POWER
• @PREVSIBLING
• @PRIOR
• @PRIORS
• @RANGE
• @RANGEFIRSTVAL
The following functions are not supported for hybrid aggregation mode. If encountered, Essbase defaults to block storage execution for these functions.
• @ALLOCATE
• @CREATEBLOCK
• @IRREX
• @MDALLOCATE
• @MDSHIFT
• @MOVSUMX
• @PTD
• @SANCESTVAL
• @STDEV
• @STDEVP
• @STDEVRANGE
• @SYD
• @TREND
• @XWRITE
Calculation Commands

Calculation scripts enable you to develop custom operations to supplement the built-in calculation of the database outline.

- Calculation Commands Overview
- Calculation Operators
- Calculation Command Groups
- Calculation Command List

Calculation Commands Overview

You use calculation scripts to create calculations that differ from those defined in the database outline. Calculation scripts enable development of custom operations to supplement the built-in calculation of the database outline.

Calculation commands are the elements of calculation scripts that instruct Essbase in the calculation rules to be used.

When a database is created, a default calculation script is set to "calculate all", which means that it will calculate all dimensions based on the database outline's hierarchical relationships and formulas.

You can override this default script by using a custom script. You can use the custom script(s) temporarily or permanently, without altering the default script. In the custom script, you can refer to calculation rules defined in the database outline or you can specify custom formulas, calculation formats, and calculation orders.

A calculation script contains a series of calculation commands. The order of the commands defines the execution order of the calculation.

Calculation Operators

Calculation operators (mathematical, conditional and logical, and cross-dimensional) define equations for member formulas and calc scripts.

- Mathematical Operators
- Conditional and Logical Operators
- Cross-Dimensional Operator

Mathematical Operators

Mathematical operators perform common arithmetic operations.
Table 3-1  Mathematical Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Adds</td>
</tr>
<tr>
<td>-</td>
<td>Subtracts</td>
</tr>
<tr>
<td>*</td>
<td>Multiplies</td>
</tr>
<tr>
<td>/</td>
<td>Divides</td>
</tr>
<tr>
<td>%</td>
<td>Evaluates percentage, for example: Member1%Member2 evaluates Member1 as a percentage of Member2.</td>
</tr>
<tr>
<td>()</td>
<td>Controls the order of calculations and nests equations and formulas</td>
</tr>
</tbody>
</table>

Conditional and Logical Operators

Conditional operators build logical condition into calculations.

Table 3-2  Conditional and Logical Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF</td>
<td>ELSE</td>
</tr>
<tr>
<td>&gt;</td>
<td>Data value is greater than</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Data value is greater than or equal to</td>
</tr>
<tr>
<td>&lt;</td>
<td>Data value is less than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Data value is less than or equal to</td>
</tr>
<tr>
<td>=</td>
<td>If data value is equal to</td>
</tr>
<tr>
<td>&lt; &gt; or !=</td>
<td>Data value is not equal to</td>
</tr>
<tr>
<td>AND</td>
<td>Logical AND linking operator for multiple value tests. Result is TRUE if both conditions are TRUE. Otherwise the result is FALSE.</td>
</tr>
<tr>
<td>OR</td>
<td>Logical OR linking operator for multiple value tests. Result is TRUE if either condition is TRUE. Otherwise the result is FALSE.</td>
</tr>
<tr>
<td>NOT</td>
<td>Logical NOT operator. Result is TRUE if condition is FALSE. Result is FALSE if condition is TRUE.</td>
</tr>
</tbody>
</table>

1 The logical constants TRUE and FALSE are interpreted as 1 (TRUE) and 0 (FALSE) where appropriate.
2 The logical constants TRUE and FALSE are interpreted as 1 (TRUE) and 0 (FALSE) where appropriate.
3 The logical constants TRUE and FALSE are interpreted as 1 (TRUE) and 0 (FALSE) where appropriate.

Cross-Dimensional Operator

The cross-dimensional operator points to data values of specific member combinations. It is created with a hyphen (-) and a right angle bracket (>), with no space between them: ->
Calculation Command Groups

This section lists calculation commands grouped by type:

• Conditional Commands
• Control Flow Commands
• Data Declaration Commands
• Functional Commands
• Member Formulas

Conditional Commands

Conditional commands control the flow of events in formulas. You can control which formulas are executed within a calculation, test conditions, and calculate a formula based on the result of the test.

• IF
• ENDIF
• ELSE
• ELSEIF

When you use an IF statement as part of a member formula in a calc script, you need to:

• Associate it with a single member
• Enclose it in parentheses

For example:

Profit (IF (Sales > 100)
       Profit = (Sales - COGS) * 2;
ELSE
       Profit = (Sales - COGS) * 1.5;
ENDIF;);

Essbase cycles through the database, performing the following calculations:

1. The IF statement checks to see if the value of Sales for the current member combination is greater than 100.
2. If Sales is greater than 100, Essbase subtracts the value in COGS from the value in Sales, multiplies it by 2, and places the result in Profit.
3. If Sales is less than, or equal to 100, Essbase subtracts the value in COGS from the value in Sales, multiplies it by 1.5, and places the result in Profit.

The entire IF ENDIF statement is enclosed in parentheses and associated with the Profit member, Profit (IF(fixend.htm)fixend.htm).
Control Flow Commands

Control Flow commands are used to iterate a set of commands or to restrict the commands' effect to a subset (partition) database. They control the flow of a calculation script. The FIX...ENDFIX and EXCLUDE...ENDEXCLUDE commands restrict calculations to specified members. The LOOP...ENDLOOP command enables repetition. The FIXPARALLEL...ENDFIXPARALLEL command block enables parallel calculation controls on a subset.

Data Declaration Commands

These commands are used to declare and set the initial values of temporary variables. The values stored in a variable are not returned in queries, because they only exist while the calculation script is being processed. If you want to report these values, you need to create members within the database outline, or assign the values from the variables into existing members.

- ARRAY
- VAR

Functional Commands

Functional commands are used to perform operations such as calculation, data copying, exporting data, and clearing data.

- AGG
- CALC ALL
- CALC AVERAGE
- CALC DIM
- CALC FIRST
- CALC LAST
- CALC TWOPASS
- CLEARBLOCK
- CLEARDATA
- DATACOPY
- DATAEXPORT
- DATAEXPORTCOND
- DATAIMPORTBIN
- SET DATAEXPORTOPTIONS
- SET DATAIMPORTIGNORETIMESTAMP
- SET AGGMISSG
- SET CACHE
- SET CLEARUPDATESTATUS
- SET FRMLBOTTOMUP
Member Formulas

Member formulas are used to calculate the default outline format on a custom formula within the script. As with formulas in the database outline, a formula in a calculation script defines mathematical relationships between database members. For example, the following expressions are valid within a calculation script:

"Profit%";

Specifying a member name with a formula defined in the outline calculates the member using its formula.

Expenses = Payroll + Marketing;

The above formula expresses a simple mathematical relationship, which is used in place of the database outline formula on the Expenses member.

Interdependent Member Formulas

Essbase optimizes calculation performance by calculating formulas for a range of members in the same dimension. However, some formulas require values from members of the same dimension. A good example is that of cash flow, in which the opening inventory is dependent on the closing inventory from the previous month.

For examples of interdependent formulas, see Using Interdependent Values in Designing and Maintaining Essbase Cubes.

When you use an interdependent formula in a calc script, the same rules apply as for the IF statement. You need to:

• Associate the formula with a single member
• Enclose the formula in parentheses

If you place the following interdependent formula in a calc script, you construct it as follows:

"Opening Inventory" (IF(NOT @ISMBR (Jan))"Opening Inventory" = @PRIOR("Ending Inventory"));
ENDIF;
"Ending Inventory" = "Opening Inventory" - Sales + Additions;)

The entire formula is enclosed in parentheses and associated with the Opening Inventory member, "Opening Inventory" (IF(fixend.htm)...).
Calculation Command List

- & (ampersand)
- AGG
- ARRAY
- CALC ALL
- CALC AVERAGE
- CALC DIM
- CALC FIRST
- CALC LAST
- CALC TWOPASS
- CLEARBLOCK
- CLEARDATA
- DATACOPY
- DATAEXPORT
- DATAEXPORTCOND
- DATAIMPORTBIN
- DATAMERGE
- ELSE
- ELSEIF
- ENDIF
- EXCLUDE...ENDEXCLUDE
- FIX...ENDFIX
- FIXPARALLEL...ENDFIXPARALLEL
- IF
- LOOP...ENDLOOP
- POSTFIXPARALLEL
- SET Commands
- SET AGGMISSG
- SET CACHE
- SET CALCDIAGNOSTICS
- SET CALCPARALLEL
- SET CALCTASKDIMS
- SET CLEARUPDATESTATUS
- SET COPYMISSINGBLOCK
- SET CREATENONMISSINGBLK
& (ampersand)

Prefaces a substitution variable in a calculation script.

Syntax

&variableName;

Parameters

variableName
The name of the substitution variable set on the database.

Notes

Essbase treats strings beginning with & as substitution variables, replacing them with values before parsing the calculation script.

Example

&CurQtr;

becomes

Qtr1;

if substitution variable &CurQtr has the value "Qtr1".
AGG

Consolidates database values. This command ignores all member formulas, consolidating only parent/child relationships.

The AGG command performs a limited set of high-speed consolidations. Although AGG is faster than the CALC commands when calculating sparse dimensions, it cannot calculate formulas; it can only perform aggregations based on the database structure. AGG aggregates a list of sparse dimensions based on the hierarchy defined in the database outline. If a member has a formula, it is ignored, and the result does not match the relationship defined by the database outline.

If you want to aggregate a dimension that contains formulas:

1. Calculate any members that are "leaf" members (that is, level 0).
2. Aggregate the dimension, using the AGG command.
3. Calculate all other members with formulas that have not been calculated yet.

Syntax

AGG (dimList);

Parameters

dimList
Name of a dimension or comma-separated list of dimensions.

Notes

• AGG only works with sparse dimensions.
• When a dimension contains fewer than six consolidation levels, AGG is typically faster than CALC. Conversely, the CALC command is usually faster on dimensions with six or more levels.
• AGG follows the rules for any defined FIX command.

Example

AGG(Market);
AGG(Product,Market,Scenario);

Related Topics

• CALC ALL
• CALC DIM
• SET AGGMISSG

ARRAY

Declares one-dimensional array variables.

Syntax

ARRAY arrayVariableName [dimName] = { constList};

Parameters

arrayVariableName
Comma-delimited list of one or more array variable names.

dimName
Dimension whose size determines the size of the array variable. Surround dimName with brackets [

constList
Optional list of data values used to initialize the array variable(s). If no initialization is performed, the array variables are set to #MISSING. The order of the values corresponds to the order of the members in the dimension used to define the array.

Notes

• Typically, arrays are used to temporarily store variables as part of a member formula. The variables cease to exist after the calculation script ends. The size of the array variable is determined by the corresponding dimension (for example, if dimension Period has 12 members, ARRAY Discount[Period] has 12 members).
• To create multiple arrays simultaneously, separate the array declarations in the ARRAY command with commas, as shown in the Example.
• You can calculate data for an array directly as part of a member formula. As the member formula is processed, each value in the array is assigned as its member is evaluated in the calculation.
• Do not use quotation marks (") in variables; for example:

ARRAY "discount"

Example

ARRAY discount[Scenario];

yields an array of 4 entries, with the values 1 through 4 entered in those four entries.

ARRAY discount[Scenario] = {1, 2, 3, 4};
ARRAY discount[Scenario], tmpProduct[Product];

yields two arrays:

1. discount, corresponding to Scenario and containing four members
2. tmpProduct, corresponding to Product and containing nine members

See Also

VAR
CALC ALL

Calculates and aggregates the entire database based on the database outline.

Syntax

CALC ALL [EXCEPT DIM (dimList) | MBR (mbrList)];

Parameters

EXCEPT
Defines an exception list of dimensions or members to be excluded from calculation.

DIM
Single-dimension specification.

dimList
Optional comma-delimited list of dimensions.

MBR
Single-member specification.

mbrList
Optional comma-delimited list of members, member set functions, or range functions.

Notes

The order in which dimensions are processed depends on their characteristics in the outline. For more information, see Defining Calculation Order in Designing and Maintaining Essbase Cubes.

Example

CALC ALL;
CALC ALL EXCEPT DIM(Product);

See Also

• CALC DIM
• SET FRMLBOTTOMUP
• SET UPDATECALC

CALC AVERAGE

Calculates members tagged as time balance Average or Average Non-Missing. All other member calculations are ignored.

Syntax

CALC AVERAGE;
Notes

This command calculates based on the Accounts dimension; it does not do a Time Series calculation on the Time dimension.

Example

CALC AVERAGE;

Related Topics

• CALC FIRST
• CALC LAST

CALC DIM

Calculates formulas and aggregations for each member of the specified dimensions.

Syntax

CALC DIM (dimList);

Parameters

dimList
Dimension or comma-delimited list of dimensions to be calculated.

Notes

The order in which dimensions are calculated depends on whether they are dense or sparse. Dense dimensions are calculated first, in the order of dimList. The sparse dimensions are then calculated in a similar order.

Example

CALC DIM(Accounts);
CALC DIM(Dense1,Sparse1,Sparse2,Dense2);

In the above example, the calculation order is: Dense1, Dense2, Sparse1, Sparse2. If your dimensions need to be calculated in a particular order, use separate CALC DIM commands:

CALC DIM(Dense1);
CALC DIM(Sparse1);
CALC DIM(Sparse2);
CALC DIM(Dense2);

Related Topics

• CALC ALL
• SET UPDATECALC
• SET CLEARUPDATESTATUS

CALC FIRST

Calculates all members tagged in the database outline as time balance First.

Note:

Only members tagged as time balance First are calculated using this command. Other members are ignored.

Syntax

CALC FIRST;

Notes

This command calculates based on the Accounts dimension; it does not do a Time Series calculation on the Time dimension.

Example

CALC FIRST;

Related Topics

• CALC AVERAGE

• CALC LAST

CALC LAST

Calculates all members tagged in the database outline as time balance Last.

Note:

Only members tagged as time balance Last are calculated using this command. Other members are ignored.

Syntax

CALC LAST;

Notes

This command calculates based on the Accounts dimension; it does not do a Time Series calculation on the Time dimension.
Example

CALC LAST;

Related Topics

• CALC AVERAGE
• CALC FIRST

CALC TWOPASS

Calculates all members tagged in the database outline as two-pass. These members must be on a dimension tagged as Accounts.

Syntax

CALC TWOPASS;

Notes

Member formulas are applied at each consolidated level of the database. All non two-pass members are ignored during this process.

Example

CALC TWOPASS;

CLEARBLOCK

Sets cell values to #MISSING, and if all the cells are empty or #MISSING, removes the block. This command is useful when you need to clear old data values across blocks before loading new values.

CLEARBLOCK helps optimize database calculation speed. For example, if an initial calculation creates numerous consolidated level blocks, subsequent recalculations take longer, because Essbase must pass through the additional blocks. CLEARBLOCK clears blocks before a calculation occurs.

Another example: if a database to be copied contains a lot of empty blocks, copying the database also copies the empty blocks, resulting in a many more empty blocks. Using CLEARBLOCK EMPTY first makes the copy process more efficient.

If you use CLEARBLOCK within a FIX statement containing dense dimension members, Essbase clears only the cells within the fixed range, and not the entire block.

Syntax

CLEARBLOCK ALL | UPPER | NONINPUT | DYNAMIC | EMPTY;
Parameters

**ALL**
Clears and removes all blocks.

**UPPER**
Clears consolidated level blocks.

**NONINPUT**
Clears blocks containing derived values. Applies to blocks that are completely created by a calculation operation. Cannot be a block into which any values were loaded.

**DYNAMIC**
Clears blocks containing values derived from Dynamic Calc and Store member combinations.

**EMPTY**
Removes empty blocks (blocks where all values are #MISSING).

Notes

- If you regularly enter data values directly into a consolidated level, the UPPER option overwrites your data. In this case, you should use the NONINPUT option, which only clears blocks containing calculated values.

- If you use CLEARBLOCK EMPTY, the resulting, smaller database can be processed more efficiently; however, the CLEARBLOCK EMPTY process itself can take some time, depending on the size and density of the database.

- If CLEARBLOCK is used within a FIX command on a dense dimension, the FIX statement is ignored and all blocks are scanned for missing cells.

- In a FIX statement, blocks are cleared only if the entire CLEARBLOCK block is selected by the FIX (no dense dimensions in the FIX), and the block is update-able (it is not a replicated-partition target region). If you wish to retain empty blocks, then in the FIX statement, set the blocks to #MISSING, instead of using CLEARBLOCK.

For example, the following command block clears East data and removes the block (because Market is sparse):

```plaintext
FIX("East")
    CLEARBLOCK ALL;
ENDFIX
```

The following command block sets New York data values to #MISSING without removing the blocks:

```plaintext
FIX("East")
    "New York" = #Missing;
ENDFIX
```

- To use this command with parallel calculation, use **FIXPARALLEL...ENDFIXPARALLEL** instead of SET CALCPARALLEL.
Example

CLEARBLOCK ALL;
CLEARBLOCK UPPER;
CLEARBLOCK NONINPUT;
CLEARBLOCK DYNAMIC;
CLEARBLOCK EMPTY;

See Also

CLEARDATA

CLEARDATA

Clears data values from the database and sets them to #MISSING.

This command is useful when you need to clear existing data values before loading new values into a database. CLEARDATA can only clear a section of a database. It cannot clear the entire database. To clear the entire database, use the following MaxL statement:

```maxl
alter database <dbs-name> reset;
```

Syntax

CLEARDATA mbrName;

Parameters

**mbrName**
Any valid single member name or member combination, or a function that returns a single member or member combination.

Notes

- CLEARDATA does not work if placed in an IF statement.
- Use CLEARBLOCK instead of CLEARDATA if you wish to remove blocks from the database, which can improve performance.
- To use this command with parallel calculation, use FIXPARALLEL...ENDFIXPARALLEL instead of SET CALCPARALLEL.

Example

CLEARDATA Budget;

Clears all Budget data.

CLEARDATA Budget->Colas;
Clears only Budget data for the Colas product family.

```plaintext
FIX("Actual")
    CLEARDATA "200-10``
ENDFIX;
```

**Caution:**

Clears data from the 200-10 block, but does not remove the block, as this is not a whole-block fix (a dense dimension is selected in the FIX).

**DATACOPY**

Copies a range of data cells to another range within the database.

This command is useful when you must maintain an original set of data values and perform changes on the copied data set.

DATACOPY is commonly used as part of the currency conversion process.

DATACOPY is useful when you need to define multiple iterations of plan data.

To reduce typing, if any dimension(s) represented by the members in `mbrName1` are not represented in `mbrName2`, then by default the same member or members from `mbrName1` are assumed to exist in `mbrName2` to complete the range. The reverse is not true. Any dimension explicitly represented in `mbrName2` MUST be represented by another member of the same dimension in `mbrName1`.

The ranges specified by both `mbrName1` and `mbrName2` must be of the same size. The same dimensions represented by the members that make up `mbrName1` must also be present in `mbrName2`.

**Syntax**

```
DATACOPY mbrName1 TO mbrName2;
```

**Parameters**

`mbrName1` and `mbrName2`

Any valid single member name or member combination.

**Notes**

- The size of the copied dimensions must be equal to the destination (`TO`) size.
- DATACOPY follows the rules for any defined FIX command.
- To prevent creation of #MISSING blocks, add the following calculation command to your script:

```
SET COPYMISSINGBLOCK OFF;
```
To use this command with parallel calculation, use
FIXPARALLEL...ENDFIXPARALLEL instead of SET CALCPARALLEL.

Example

DATACOPY Plan TO Revised_Plan;

See Also

• SET COPYMISSINGBLOCK
• MDX Insert

DATAEXPORT

Writes data to a text or binary file.

Syntax

For a text output file:

DATAEXPORT "File" "delimiter" "fileName" "missingChar"

For a binary output file (DATAEXPORT to binary files is not supported across Essbase releases, and is only supported between 64-bit operating systems):

DATAEXPORT "Binfile" "fileName"

Parameters

"File" "Binfile"
Required keyword for the type of output file. Specify the appropriate keyword, then use the associated syntax.

"delimiter"
Required for "File" exports
The character that separates fields; for example, ",,"
Do not use with "Binfile" exports

"fileName"
Required for "File" and "Binfile" exports
Full path name for the export file.

"missingChar"
Optional for output type "File"
• A text string to represent missing data values. Maximum length: 128 characters.
• "NULL" to skip the field, resulting in consecutive delimiters (such as ,,).
• Default value: #MI

Do not use with "Binfile" exports, or in combination with the SET DATAEXPORTRELATIONALFILE command.
Notes

• In general, specify SET commands within the calculation script to specify various options, and then use FIX…ENDFIX to refine data to be exported, including the DATAEXPORT command within the FIX…ENDFIX command set. Without FIX…ENDFIX, the entire database is exported.

• If outputting a file, and fileName:
  – Does not include a path, the file is written in the application directory.
  – Includes a path, Essbase interprets the path in context to the server. Export files cannot be written to a client.

• To use this command with parallel calculation, use FIXPARALLEL…ENDFIXPARALLEL instead of SET CALCPARALLEL.

• Use the DATAIMPORTBIN command to import a previously exported binary export file.

• Calculation export locks one block at a time; all other blocks can be updated. For information about handling concurrent calculation, see Understanding Intelligent Calculation in Designing and Maintaining Essbase Cubes.

Description

The DATAEXPORT calculation command writes data into a text or binary output file, or connects directly to an existing relational database wherein the selected exported data is inserted.

Whereas the MaxL Export Data statement can export all, level 0, or input data from the entire database as text data, the DATAEXPORT calculation command also enables you to:

• Use FIX…ENDFIX or EXCLUDE…ENDEXCLUDE calculations to select a slice of the database and use a DATAIMPORTCONDC command to select data based on data values.

• Use parameters to qualify the type and destination of the export data.

• Use options provided by the SET DATAIMPORTOPTIONS command to refine export content, format, or process.

• Use the SET DATAIMPORTIGNORETIMESTAMP command to manage the import requirement for a matching outline timestamp.

Example

Text Output File Example 1

SET DATAIMPORTOPTIONS
  {  
    DataExportLevel "LEVEL0"; 
  };
DATAIMPORTCONDC ("Sales">=1000); 
FIX ("100-10","New York","Actual","Sales"); 
DATAIMPORT "File" "," "jan.txt" ";
ENDFX;
Specifies a level 0 data export level, limits output to data only with 1000 or greater Sales, fixes the data slice, then exports to a text file located in the database directory, using comma (,) delimiters and specifying #MI for missing data values.

**Binary Example 1: Export**

```
SET DATAEXPORTOPTIONS
{
  DataExportLevel "ALL";
};
FIX ("New York");
DATAEXPORT "BinFile" "newyork.bin";
ENDFIX;
```

Exports all New York blocks. Binary exports can be fixed only on sparse dimensions. Essbase uses the same bitmap compression technique to create the file as is used by Essbase Kernel.

**Binary Example 2: Import**

```
SET DATAIMPORTIGNORETIMESTAMP OFF;
DATAIMPORTBIN "newyork.bin"
```

Imports the previously exported file. The timestamp must match. The data is imported to the database on which the calculation script is executed. Because only data was exported, to recreate a database after using DATAIMPORT to read in the data, you must recalculate the data.

**See Also**

- DATAEXPORTCOND
- DATAIMPORTBIN
- FIX...ENDFIX
- SET Commands
- SET DATAEXPORTOPTIONS
- SET DATAIMPORTIGNORETIMESTAMP
- MDX Export

**DATAEXPORTCOND**

Specifies value conditions that select export records to be included or marked as "#NoValue" in the export output file.

**Syntax**

```
DATAEXPORTCOND "conditionExpression" ReplaceAll;
```
Parameters

**conditionExpression**
One or more conditions separated by a logical AND or OR. Each condition specifies a member name the value of which is equal to (=), greater than (>), greater than or equal (>=), less than (<), or less than or equal (<=) to a specified value or the value of another member; for example, "Sales" > 500 AND "Ending Inventory" < 0.
The condition list is processed from left to right. Thus the result of cond1 is calculated first, then the operator (AND or OR) is calculated against cond2, and so on. While processing conditions, if a resultant condition is found to be false, the entire record is omitted from the output file.

**ReplaceAll**
The keyword that indicates whether exported records are to be excluded from the initial export set of records, or included but marked as "#NoValue". The initial export set of records is determined by the region defined by the FIX command and SET commands that apply to the data export.

- When ReplaceAll is not specified, only those records within the initial export set are exported that meet the specified conditions.
- When ReplaceAll is specified, all records within the initial export set are exported, but the AND and OR specifications are ignored. All fields that do not satisfy any of the specified conditions are marked as #NoValue.

Notes

Use DATAEXPORTCOND to specify conditions that identify records to be exported based on field values. Whether a condition can specify a member compared to a numeric value or compared to another member depends the member being a row or column element of the output. In order to represent multidimensional data within a two-dimension file, the members of one dense dimension become columns. The combinations of the members of the other dense dimensions and the sparse dimensions create rows. (You can use the DataExportColHeader option of the SET DATAEXPORTOPTIONS calculation command to specify which dimension defines the columns.)

- If a condition is placed on a column member, the value of the specified member can be compared to a specific value (for example, Sales > 500) or to the value of another member of the same export record (for example, Sales < Cost).
- If a condition is placed on a row member, the value of the specified member can be compared only to a specific value (for example, Cost < 500).

Example

**Not Using ReplaceAll**

```
SET DATAEXPORTOPTIONS {
  DataExportLevel "ALL";
};
DATAEXPORTCOND {Actual >= 2 AND Sales > 2000 OR COGS > 600};
FIX("100-10","East");
DATAEXPORT "File ", "E:\temp\2222.txt";
ENDFIX;
```
Sets the contents of the initial export file through the DataExportLevel option of the `SET DATAEXPORTOPTIONS` command and `FIX...ENDFIX` command. The `DATAEXPORTCOND` command specifies the records to be included when the Actual value is greater than or equal to 2 and Sales are greater than 2000, or when the Actual value is greater than or equal to 2 and COGS is greater than 600. The conditions are specified on the column Actual, the column Sales, and the column COGS. The exported data includes only records that meet the conditions. Sample output:

```
"Sales","COGS","Marketing","Payroll","Misc","Opening Inventory","Additions","Ending Inventory"
"100-10","East"
"Jun","Actual",2205,675,227,177,2,3775,2028,3598
"Jul", "Actual", 2248,684,231,175,2,3598,1643,2993
"Sep", "Actual", 2012,633,212,175,4,2389,1521,1898
"Jun", "Budget", 2070,620,180,120,#Mi,2790,1700,2420
"Jul", "Budget", 2120,620,180,120,#Mi,2420,1400,1700
"Aug", "Budget", 2120,620,180,120,#Mi,1700,1400,980
```

**Using ReplaceAll**

```
SET DATAEXPORTOPTIONS
{
  DataExportLevel "ALL";
};
DATAEXPORTCOND (Actual >= 2 AND Sales > 2000 OR COGS > 600;
  FIX("100-10","East");
  DATAEXPORT "File" "," "E:\temp\2222.txt" ReplaceAll;
ENDFIX;
```

Using the same conditions as the prior example, but including "ReplaceAll" in the `DATAEXPORT` command, the exported data includes all records specified by the `FIX` command. #NoValue is inserted for fields that do not meet the specified conditions. Sample output:

```
"Sales","COGS","Marketing","Payroll","Misc","Opening Inventory","Additions","Ending Inventory"
"100-10","East" "Jan","Actual",#NoValue,#NoValue, 199,175,2,4643,1422,4253
"Feb","Actual",#NoValue,#NoValue,196,175,3,4253,1413,3912
"Mar","Actual",#NoValue,#NoValue,199,175,3,3912,1640,3747
"Apr","Actual",#NoValue,606,204,177,3,3747,1824,3701
"May","Actual",#NoValue,622,210,177,4,3701,2023,3775
"Jun", "Actual", 2205,675,227,177,2,3775,2028,3598
"Jul", "Actual", 2248,684,231,175,2,3598,1643,2993
"Aug", "Actual", 2245,684,231,175,#NoValue,2993,1641,2389
"Sep", "Actual", 2012,633,212,175,4,2389,1521,1898
"Oct", "Actual",#NoValue,#NoValue,196,175,3,1898,1535,1677
"Nov", "Actual",#NoValue,#NoValue,192,175,#NoValue,1677,1584,1553
"Dec", "Actual",#NoValue,#NoValue,200,175,2,1553,1438,1150
"Jan", "Budget",#NoValue,#NoValue,160,120,#Mi,4490,1100,3900
"Feb", "Budget",#NoValue,#NoValue,160,120,#Mi,3900,1200,3460
"Mar", "Budget",#NoValue,#NoValue,160,120,#Mi,3460,1400,3170
"Apr", "Budget",#NoValue,#NoValue,150,120,#Mi,3170,1500,2920
"May", "Budget",#NoValue,#NoValue,160,120,#Mi,2920,1700,2790
```
Related Topics

• DATAEXPORT
• FIX...ENDFIX
• SET Commands
• SET DATAEXPORTOPTIONS

DATAIMPORTBIN

Imports the binary output file previously exported with the DATAEXPORT "Binfile" calculation command.

You can use DATAIMPORTBIN to import previously exported binary files. For example, you can use DATAEXPORT "Binfile" and DATAIMPORTBIN as a method for data backup and recovery.

Note:

DATAIMPORTBIN is not supported across Essbase releases.

Syntax

DATAIMPORTBIN fileName;

Parameters

fileName
Full path name for the binary input file to be imported.

Notes

• The outline timestamp is included with the export file created by DATAEXPORT. By default, the DATAIMPORTBIN process checks the timestamp. Use the SET DATAIMPORTIGNORETIMESTAMP calculation command with DATAIMPORT to bypass checking the timestamp. See SET DATAIMPORTIGNORETIMESTAMP for details.
• Use DATAIMPORTBIN only with files created by DATAEXPORT "Binfile".

Example

DATAIMPORTBIN e:\january\sales.bin;
Specifies the binary file e:\january\sales.bin is to be imported to the database for which the calculation script is being run.

Related Topics
- DATAEXPORT
- SET DATAIMPORTIGNORETIMESTAMP

DATAMERGE

Merges a range of data cells from the current scenario to a baseline target, or to another scenario.

This command is useful when, after working on a scenario, you decide to commit the changes.

Syntax

DATAMERGE sourceMbrName targetMbrName [NOCALC] [SBSOURCE|SBTARGET];

Parameters

sourceMbrName
A single sandbox member name or member combination.

targetMbrName
A single sandbox member name.

NOCALC
Keyword indicating that only cells with cell status of INPUT or LOAD should be merged.

SBSOURCE or SBTARGET
Keyword indicating a merge preference:
- SBSOURCE—The default. If merge source and merge target have different values, apply the source value to the target.
- SBTARGET—If merge source and merge target have different values, apply the target value to the source.

Notes
- Use of this command presumes you have created and provisioned a cube for scenario modeling.
- The merge process iterates over all non-missing cells in the source, and checks whether the cell should be copied into the target. The decision depends on the value of the optional SOURCE | TARGET keyword (SOURCE is default). If a cell needs to be merged, Essbase copies the cell's value, transaction ID, and status from the source to the target, and updates the cell status to MERGED.
Example

Fix(@Relative(Colas,0))
DATAMERGE sb1->2016 sb2 NOCALC SBTARGET;
EndFix

ELSE

The ELSE command designates a conditional action to be performed in an IF statement. All actions placed after the ELSE in an IF statement are performed only if the test in the IF statement generates a value of FALSE.

Syntax

ELSE statement ; [ ...statement; ] ENDIF;

Parameters

statement
Those operations that are to be performed in the event that the IF test including the ELSE command produces a FALSE, or 0, result.

Notes

• The ELSE command can only be used in conjunction with an IF command.
• You do not need to end ELSE statements with ENDIF statements. Only IF statements should be ended with ENDIF statements.

Example

The following example is based on the Sample Basic database. This calculation script tests to see if the current member in the Market dimension is a descendant of West or East. If so, Essbase multiplies the value for Marketing by 1.5. If the current member is not a descendant of West or East, Essbase multiplies the value for Marketing by 1.1.

Marketing
{IF (@ISMBR(@DESCENDANTS(West))
   OR
      (@ISMBR(@DESCENDANTS(East)))
Marketing = Marketing * 1.5;
ELSE
Marketing = Marketing * 1.1;
ENDIF;

Related Topics

• ELSEIF
• ENDIF
• IF
ELSEIF

Designates a conditional test and conditions that are performed if the preceding IF test generates a value of FALSE. For this reason, multiple ELSEIF commands are allowed following a single IF.

Syntax

ELSEIF( condition  ) statement ; [ ...statement ; ]
ELSEIF | ELSE | ENDIF

Parameters

c condition
Formula or function that returns a Boolean value of TRUE (a nonzero value) or FALSE (a zero value).

statement
Those operations that are to be performed in the event that the IF test (including the ELSE command) produces a FALSE, or 0, result.

Notes

• The ELSEIF command must be used in conjunction with an IF command.
• You do not need to end ELSEIF statements with ENDIF statements. Only IF statements should be ended with ENDIF statements. For example:

    IF (condition)
        statement;
    IF (condition)
        statement;
    ELSEIF (condition)
        statement;
    ENDIF;
    statement;
    ENDIF;

Example

The following example is based on the Sample Basic database. This calculation script tests to see if the current member in the Market dimension is a descendant of West or East. If so, Essbase multiplies the value for Marketing by 1.5. The calculation script then tests to see if the current member is a descendant of South. If so, Essbase multiplies the value for Marketing by .9. If the current member is not a descendant of West, East, or South, Essbase multiplies the value for Marketing by 1.1.

    IF (@ISMBR(@DESCENDANTS(West))
        OR
        @ISMBR(@DESCENDANTS(East))
    )
        Marketing = Marketing * 1.5;
    ELSEIF (@ISMBR(@DESCENDANTS(South))
    )
Marketing = Marketing * .9;
ELSE
    Marketing = Marketing * 1.1;
ENDIF;

Related Topics
• ELSE
• ENDIF
• IF

ENDIF

Marks the end of an IF command sequence. The ENDIF command can be used only in conjunction with IF or IF ... ELSEIF statements.

Syntax

ENDIF;

Notes
• You must supply an ENDIF statement for every IF statement in your formula or calculation script. If you do not supply the required ENDIF statements, your formula or calculation script does not verify.
• If you are using an IF statement nested within another IF statement, end each IF with an ENDIF. For example:

  "Opening Inventory"
  (IF (@ISMBR(Budget))
   IF (@ISMBR(Jan))
    "Opening Inventory" = Jan;
   ELSE
    "Opening Inventory" = @PRIOR("Ending Inventory");
   ENDIF;
  ENDIF;

• You do not need to end ELSE or ELSEIF statements with ENDIF statements.
• Although ending ENDIF statements with a semicolon is not required, it is good practice to follow each ENDIF statement in your formula or calculation script with a semicolon.
• IF, ELSE, ELSEIF, and ENDIF must all be used within a database outline formula, or must be associated with a member in the database outline when used in a calculation script.

Example

The following example is based on the Sample Basic database. This calculation script tests to see if the current member in the Market dimension is a descendant of West or East. If so, Essbase multiplies the value for Marketing by 1.5. The calculation script then tests to see if the current member is a descendant of South. If so, Essbase
multiplies the value for Marketing by .9. If the current member is not a descendant of West, East, or South, Essbase multiplies the value for Marketing by 1.1.

```plaintext
IF (@ISMBR(@DESCENDANTS(West))
   OR
   @ISMBR(@DESCENDANTS(East))
) 
   Marketing = Marketing * 1.5;
ELSEIF (@ISMBR(@DESCENDANTS(South))
) 
   Marketing = Marketing * .9;
ELSE
   Marketing = Marketing * 1.1;
ENDIF;
```

Related Topics

- ELSE
- ELSEIF
- IF

**EXCLUDE...ENDEXCLUDE**

The EXCLUDE command allows you to define a fixed range of members which are not affected by the associated commands. The ENDEXCLUDE command ends an EXCLUDE command block.

As shown in the example, you call ENDEXCLUDE after all of the commands in the EXCLUDE command block have been called, and before the next element of the calculation script.

Specifying members that should not be calculated in an EXCLUDE..ENDEXCLUDE command may be simpler than specifying a complex combination of member names in a FIX...ENDFIX command.

**Syntax**

```plaintext
EXCLUDE (Mbrs)
COMMANDS ;
ENDEXCLUDE
```

**Parameters**

- **Mbrs**
  A member name or list of members from any number of database dimensions. *Mbrs* can also contain:
  - AND/OR operators. Use the AND operator when all conditions must be met. Use the OR operator when one condition of several must be met.
  - Member set functions, which are used to build member lists based on other members.
COMMANDS
The commands to be executed for the duration of the EXCLUDE.

Notes

• Use EXCLUDE…ENDEXCLUDE commands only within calculation scripts, not in outline member formulas.

• You can include EXCLUDE commands within FIX command blocks.

• If a FIX command within an EXCLUDE command block specifies cells already specified by the EXCLUDE statement, those cells are not calculated, and a warning message is posted to the application log file.

• An EXCLUDE command block cannot include CALC ALL, CLEARDATA, and DATACOPY commands.

• AND and OR operators have the same precedence and are evaluated from left to right. Use parentheses to group the expressions. For example: A OR B AND C is the same as ((A OR B) AND C). However, subexpressions (for example, (A OR (B AND C))) are evaluated before the whole expression, producing a different result.

• Inside EXCLUDE command blocks, the AND operator represents the intersection of two sets; the OR operator represents the union of two sets. In formulas, these operators are Boolean operators. Using the AND or OR operators on members that are from different dimensions, returns:
  – AND: An empty set. The EXCLUDE statement is ignored and the calculation continues with a warning message.
  – OR: The union of two members sets. EXCLUDE (Jan OR Market) is identical to FIX (Jan, Market).

• NOT operators are not supported in EXCLUDE command blocks. Use the @REMOVE function.

• You do not need to follow ENDEXCLUDE with a semicolon.

• Use the @ATTRIBUTE and @WITHATTR functions to specify attributes within EXCLUDE command blocks; for example EXCLUDE(@ATTRIBUTE(Can)). FIX(Can) is not supported.

• You cannot use EXCLUDE on a dimension if it is a subset of a dimension that you calculate within the EXCLUDE command block. For example you could not use Market "New Mkt" in an EXCLUDE statement if you calculate all of Market within the command block.

• Dynamic Calc members are ignored in an EXCLUDE statement. If the only member in an EXCLUDE statement is a Dynamic Calc member, an error message is displayed stating that the EXCLUDE statement cannot contain a Dynamic Calc member.

• If the EXCLUDE command is issued from a calculation script and produces an empty set, that part of the calculation is ignored, and the calculation continues to the next statement. The application log entry for the calculation shows that the EXCLUDE statement evaluated to an empty set (Calculating […] with fixed members []).
For example, consider the following statement in a Sample Basic calculation script:

```
EXCLUDE (@children(Jan))
CALC DIM (Accounts, Product, Market)
ENDEXCLUDE
```

Since @children(Jan) is empty (Jan is a level 0 member), the EXCLUDE parameter is ignored; the calculation operates on the entire database.

Similarly, if a region defining a partition or a security filter evaluates to an empty set, Essbase behaves as if the region definition or security filter does not exist.

- Calculator function @RANGE and the cross-dimensional operator (->) cannot be used inside an EXCLUDE Mbrs parameter).

Example

The following example excludes calculations on the children of Qtr4, enabling calculation of other quarters in the Year dimension.

```
EXCLUDE (@CHILDREN(Qtr4))
CALC DIM (Year)
ENDEXCLUDE
```

Related Topics

- FIX...ENDFIX
- LOOP...ENDLOOP

**FIX...ENDFIX**

The FIX...ENDFIX command block restricts database calculations to a subset of the database. All commands nested between the FIX and ENDFIX statements are restricted to the specified database subset.

This command is useful because it allows you to calculate separate portions of the database using different formulas, if necessary. It also allows you to calculate the subsection much faster than you would otherwise.

The ENDFIX command ends a FIX command block. As shown in the example, you call ENDFIX after all of the commands in the FIX command block have been called, and before the next element of the calculation script.

The optional syntax within the {set} brackets is for selecting regions you define using calculation tuples. Tuple selection helps you optimize asymmetric grid calculations across dimensions, avoiding over-calculation.

**Syntax**

```
FIX ([{ tupleList | @GRIDTUPLES(dimensionList) },] fixMbrs)
COMMANDS ;
ENDFIX
```
Parameters

fixMbrs
A member name or list of members from any number of database dimensions. 
fixMbrs can also contain:

- AND/OR operators. Use the AND operator when all conditions must be met. Use the OR operator when one condition of several must be met.
- Member set functions, which are used to build member lists based on other members.

COMMANDS
The commands you want to be executed for the duration of the FIX.

tupleList
Optional list of calculation tuples. A calculation tuple is a list of members from two or more sparse dimensions. Tuples can contain different numbers of members. Examples:

("Diet Cola", "Cola", Florida)
(Cola, "New Hampshire")

tupleList must not contain members from dimensions used in fixMbrs. When tuples overlap, the overlapping regions are calculated only once.

@GRIDTUPLES(dimensionList)
Contextual tuple selection based on whichever members are present in a Smart View grid POV at calculation run time. Pass to the @GRIDTUPLES function a list of two or more sparse dimensions whose members from the active Smart View grid will be used to define calculation regions. Example:

@GRIDTUPLES(Product, Market)

Notes

- You can use SET EMPTYMEMBERSETS to stop the calculation within a FIX command if the FIX evaluates to an empty member set.
- FIX commands can be nested within other FIX command blocks. For an example of an incorrect use of nested FIX commands, see Using the FIX Command in Designing and Maintaining Essbase Cubes.
- FIX statements can only be used in calculation scripts, not in outline member formulas. Use an IF command instead of a FIX statement in member formulas. For example:

  Jan(
  IF (Sales)
  Actual=5;
  ENDIF;)

- AND/OR operators have the same precedence; Essbase evaluates them from left to right. Use parentheses to group the expressions. For example: \( A \ OR B \ AND C \)
the same as \((A \lor B) \land C\). However, if you use \((A \lor (B \land C))\), Essbase evaluates the sub-expression in parentheses \((B \land C)\) before the whole expression, producing a different result.

- Inside FIX statements, the AND operator represents the intersection of two sets; the OR operator represents the union of two sets. In formulas, these operators are Boolean operators. Using the AND or OR operators on members that are from different dimensions, returns:
  - AND: An empty set. The FIX statement is ignored and the calculation continues with a warning message.
  - OR: The union of two members sets. FIX (Jan OR Market) is identical to FIX (Jan, Market).

- In FIX statements, members from the same dimension are always acted on as OR unless you specify otherwise.
- NOT operators are not supported in FIX statements. Use @REMOVE with FIX statements.
- You do not need to follow ENDFIX with a semicolon.
- You can specify attributes in FIX statements using @ATTRIBUTE and @WITHATTR; for example FIX(@ATTRIBUTE(Can)). You must use these functions; FIX(Can) is not supported.
- You cannot use a FIX statement on a dimension if it is a subset of a dimension that you calculate within the FIX statement. For example you could not use Market "New Mkt" in a FIX statement if you calculate all of Market within the FIX statement.
- Dynamic Calc members are ignored in a FIX statement. If the only member in a FIX statement is a Dynamic Calc member, an error message is displayed stating that the FIX statement cannot contain a Dynamic Calc member.
- If the FIX command is issued from a calculation script and produces an empty set, that part of the calculation is ignored, and the calculation continues to the next statement. The application log entry for the calculation shows that the FIX statement evaluated to an empty set (Calculating [...] with fixed members []).

For example, using Sample Basic, assume this statement is in a calculation script:

```plaintext
FIX (@children(Jan))
CALC DIM (Accounts, Product, Market)
ENDFIX
```

Since @children(Jan) is empty, the FIX is ignored; the calculation issues a warning and operates on the entire database.

Similarly, if a region defining a partition or a security filter evaluates to an empty set, Essbase issues a warning and behaves as if the region definition or security filter did not exist.

- The @RANGE function and the cross-dimensional operator (->) cannot be used inside a FIX fixMbrs parameter.
- Using an EXCLUDE...ENDEXCLUDE block to specify members that should not be calculated may be simpler than specifying a complex combination of member names in a FIX...ENDFIX block.
The variable (\texttt{varName}) that is defined by a VAR calculation command cannot be used within the FIX member statement. The FIX members are evaluated before the calculation is executed, and variables are evaluated during runtime after the FIX statement is set. Because variables can change during the calculation execution, you cannot use the variable as part of the FIX statement. The following example shows the incorrect use of the variable in the FIX member statement:

\begin{verbatim}
VAR varName=1;
FIX (@relative(@memberat(List("Product1","Product2"),varName),0))
  COMMANDS;
ENDFIX
\end{verbatim}

\textbf{Example}

\begin{verbatim}
FIX (Budget)
  CALC DIM (Year, Measures, Product, Market);
ENDFIX
FIX (Budget, Jan, Feb, Mar, @DESCENDANTS(Profit))
  CALC DIM (Product, Market);
ENDFIX
\end{verbatim}

The following example fixes on the children of East and the Market dimension members with the UDA "New Mkt".

```
FIX (@CHILDREN(East) OR @UDA(Market, "New Mkt"))
```

The following example fixes on the children of East with the UDA "New Mkt" and Market dimension members with the UDA "Big Mkt".

```
FIX((@CHILDREN(East) AND @UDA(Market, "New Mkt")) OR @UDA(Market,"Big Mkt"))
```

\textbf{See Also}

- Calculate Selected Tuples
- EXCLUDE...ENDEXCLUDE
- LOOP...ENDLOOP
- SET EMPTYMEMBERSETS

\textbf{FIXPARALLEL...ENDFIXPARALLEL}

Enables parallel calculation on a block of commands by using up to a specified number of parallel threads.

The ENDFIXPARALLEL command ends a FIXPARALLEL command block.
Syntax

```
FIXPARALLEL (numThreads, mbrList)
COMMANDS ;
[ POSTFIXPARALLEL ( [ varName = ACCUMULATEVAR ( threadVarName ); ]* ); ]
ENDFIXPARALLEL
```

Parameters

**numThreads**
A positive integer specifying the number of threads to be made available for parallel calculation.

**mbrList**
A selection of slices for restricting the calculation. These slices become the task members for the FIXPARALLEL calculation. Can be one of the following:

- A member name or list of members. Note: If `mbrList` is a single member from one or more sparse dimensions, then it only generates one task, and cannot benefit from parallel execution. Multiple members from one or more sparse dimensions generate multiple tasks.
- Member set functions, which are used to build member lists based on other members.

The database regions (slices) you specify must be independent of one another. From `mbrList`, Essbase generates tasks to be calculated in parallel. Essbase uses only non-dynamic, non-shared, sparse members to create the tasks, which in turn determine the blocks to be calculated. Therefore, `mbrList` must contain at least one non-dynamic, non-shared, sparse member. In order to use multiple threads, `mbrList` should contain two or more members from each sparse dimension. `mbrList` should indicate at least as many tasks as the `numThreads` you specify. To avoid setting too many tasks in a FIXPARALLEL calculation, only those member combinations that are to be used for tasks should be in the `mbrList`. All other sparse member combinations belong in an inner or outer FIX.

**COMMANDS**
The commands you want to be executed for the duration of the FIXPARALLEL. These commands are applied to the database regions described by `mbrList`. May include THREADVAR commands.

**POSTFIXPARALLEL**
Optional block of operations to copy THREADVAR variables to VAR variables. Essbase executes POSTFIXPARALLEL block once, before the FIXPARALLEL command finishes. See POSTFIXPARALLEL.

**varName**
Name of a VAR variable.

**threadVarName**
Name of a THREADVAR variable.
ACCUMULATEVAR
* Used within optional POSTFIXPARALLEL. Add up all the thread values of a given THREADVAR variable. The sum is then assigned to a specified VAR variable.

ENDFIXPARALLEL
* Closes the FIXPARALLEL command block.

Notes
* You control thread activity by using:
  - The numThreads parameter
  - The THREADVAR command
  - The ACCUMULATEVAR command (inside POSTFIXPARALLEL)
  - The mbrList parameter. The member list is an important tool for optimizing calculations, because it tells Essbase how to divide the calculation regions into tasks. As mbrList becomes larger, each task becomes smaller. When tasks become too small, calculation memory overhead could slow down performance. However, when tasks are too large, there might not be enough tasks for parallel calculation threads to work on.

Overview of FIXPARALLEL
* Although parallel calculation can be performed using the CALCPARALLEL configuration setting, in certain cases it might be beneficial to use the FIXPARALLEL command block method.

In a FIXPARALLEL command block, you input some commands to be executed, along with a number of threads (numThreads) and a member list (mbrList) specifying the database regions (slices) to be calculated. Essbase creates a list of tasks from the combinations in the member list, and divides the tasks across the threads.

FIXPARALLEL has the following benefits:
  - You can use temporary variables during parallel calculation.
  - You can use the DATACOPY, DATAEXPORT, or CLEARBLOCK commands.
  - You can use it in conjunction with the @XREF or @XWRITE functions.
  - You can export regions of the database in parallel. See the Example in this topic.
  - In cases where CALCPARALLEL is not meeting performance requirements, and your outline generates many empty tasks, or contains many task groupings with fewer tasks than threads made available to the calculation. See also “Task Selection Comparison of FIXPARALLEL and CALCPARALLEL.”

When considering converting FIX statements to FIXPARALLEL within a calculation script, follow these guidelines:
  - Focus on FIX statements that do not meet your performance needs using CALCPARALLEL.
  - Focus on FIX statements that require a substantial amount of work. Parallelizing a FIX statement requires some overhead, so trying to parallelize calculation passes with light workloads may not be beneficial. Heavier workloads, such as AGG and CALC DIM, are good candidates for FIXPARALLEL.
  - First, try parallelism with a single large sparse dimension, or by restricting mbrList to one or more hierarchies with a limited stored member count. You may continue
adding dimensions to the member list to see if the calculation time continues to improve.

Note that when "parallel" calculation of tasks occurs, it means that the tasks are divided and executed concurrently in any order. In other words, there is no guarantee that any task will be executed before any other tasks. This is why the regions you specify must not have any data or calculation dependencies. For example, assume there are two parallel threads, and there is a division of work into tasks A, B, C, and D.

The possible sequence of calculation might be:

- Thread #1 executes A and then C.
- Thread #2 executes B and then D.

Or,

- Thread #1 executes A.
- Thread #2 executes B, then C, then D.

Or,

- Thread #1 executes C and then A.
- Thread #2 executes D and then B.

Task Selection Comparison of FIXPARALLEL and CALCPARALLEL

CALCPARALLEL creates tasks from the last sparse dimension first, then the second from the last, and so on, until it has enough tasks. FIXPARALLEL can choose from any sparse dimension that is not in its COMMANDS block. For example (as is true with FIX), you cannot FIXPARALLEL on (Level 0, Product) and also AGG (Product).

FIXPARALLEL can help you customize task selection, but it also assumes no interdependencies when generating tasks from the selected region. CALCPARALLEL must consider sparsity, outline order, dependencies, and member formulas in generating a task list.

Limitations of FIXPARALLEL Parallel Calculation

- For databases which are the target of transparent partitions, FIXPARALLEL is supported only when remote calculation is disabled (SET REMOTECALC OFF).
- The following calculation commands are not supported in a FIXPARALLEL block:
  - DATAEXPORT with options other than flat files
  - DATAIMPORTBIN
  - EXCLUDE...ENDEXCLUDE
- FIXPARALLEL supports up to 8 threads (more if Essbase is running on Oracle Exalytics In-Memory machine). The data structures created in each thread and the algorithms used for scheduling and executing tasks require significant CPU and memory resources. Executing highly parallelized activities on servers with limited resources might have a negative impact on performance and system stability. Therefore, using FIXPARALLEL with more than 8 threads, when the ORACLEHARDWAREACCELERATION configuration setting is set to FALSE, is not supported.
Example

FIXPARALLEL used with DATAEXPORT enables you to export restricted regions of database in parallel. The following example uses two threads to export data relating to [California], [Oregon], [Washington], [Utah], and [Nevada].

```
FIXPARALLEL (2, @CHILDREN("West"))
   DATAEXPORT "File" "dataOfWest.txt" "#MI";
ENDFIXPARALLEL
```

See also the example for POSTFIXPARALLEL.

Related Topics

- POSTFIXPARALLEL
- THREADVAR

IF

Performs conditional tests within a formula. Using the IF statement, you can define a Boolean test, as well as formulas to be calculated if the test returns either a TRUE or FALSE value.

Syntax

```
IF( condition ) statement ; [ ...statement ; ] [ ELSEIF...statement | ELSE...statement]
ENDIF;
```

Parameters

- **condition**  
  Formula or function that returns a Boolean value of TRUE (a nonzero value) or FALSE (a zero value).

- **statement**  
  Operations to be performed depending on the results of the test.

Notes

- The IF statement block can also use the ELSE and ELSEIF statements as part of its decision syntax.
- For information about using ENDIF statements and semicolons with IF, ELSE, and ELSEIF statements, see ENDIF.
- In calculation scripts, IF statements must be placed within parentheses and associated with a specific database member. They must also be closed with ENDIF statements.
- You can specify attributes in IF statements using @ATTRIBUTE and @WITHATTR; for example: IF (@ISMBR(@ATTRIBUTE(Can))) .... You must use the attribute functions; IF(@ISMBR(Can)) is not supported.
Example

Example 1

IF(
    @ISMBR(@DESCENDANTS(Europe))
    OR @ISMBR(@DESCENDANTS(Asia))
)
  Taxes = "Gross Margin" * "Foreign Tax Rate";
ELSE
  Taxes = "Gross Margin" * "Domestic Tax Rate";
ENDIF;

This test checks to see if the current cell includes a member that is a descendant of either the Europe or Asia members. If it does, the formula calculates the taxes for the member based on the foreign tax rate. If the current cell does not include a member from one of those groups, then the domestic tax rate is used for the tax calculation.

Example 2

When you use an IF statement as part of a member formula in a calculation script, you need to perform both of the following tasks:

• Associate the IF statement with a single member
• Enclose the IF statement in parentheses

A sample IF statement is illustrated in the following example:

Profit
(IF (Sales > 100)
  Profit = (Sales - COGS) * 2;
ELSE
  Profit = (Sales - COGS) * 1.5;
ENDIF;)

Essbase cycles through the database and performs the following calculations:

1. The IF statement checks to see if the value of Sales for the current member combination is greater than 100.
2. If Sales is greater than 100, Essbase subtracts the value in COGS from the value in Sales, multiplies the difference by 2, and places the result in Profit.
3. If Sales is less than or equal to 100, Essbase subtracts the value in COGS from the value in Sales, multiplies the difference by 1.5, and places the result in Profit.

The whole of the IF ... ENDIF statement is enclosed in parentheses and associated with the Profit member, Profit (IF(...)...).

See Also

• ELSE
• ELSEIF
• ENDIF
LOOP...ENDLOOP

The LOOP...ENDLOOP command block specifies the number of times to iterate calculations. All commands between the LOOP and ENDLOOP statements are performed the number of times that you specify.

Syntax

```
LOOP (integer, [break])COMMANDS ;
ENDLOOP
```

Parameters

integer
The integer constant that indicates the number of times to execute the commands contained in the loop block.

break
Optional parameter used to break the iterative process of a loop. break must be the name of a temporary variable (VAR). Setting the value of the variable to 1 during the execution of the loop causes the loop to break at the beginning of its next iteration.

COMMANDS
Those commands that you want to be executed for the duration of the LOOP.

Notes

LOOP is a block command that defines a block of commands for repeated execution. As with the FIX command, you can nest LOOP statements if necessary.

The ENDLOOP command ends a LOOP command block. It terminates the LOOP block and occurs after the commands in the LOOP block, but before any other commands.

Example

In this example, the LOOP command finds a solution for Profit and Commission. This operation is done as a loop because Profit and Commission are interdependent: Profit is needed to evaluate Commission, and Commission is needed to calculate Profit. This example thus provides a model for solving simultaneous formulas.

```
FIX("New York", Camera, Actual, Mar)
LOOP (30)
  Commission = Profit * .15;
  Profit = Margin - "Total Expenses" - Commission;
ENDLOOP;
ENDFIX
```

POSTFIXPARALLEL

The POSTFIXPARALLEL command block is an optional, post-processing block within FIXPARALLEL...ENDFIXPARALLEL. You can use it to copy temporary, thread-level...
THREADVAR values into longer-persisting VAR variables that you can use outside the FIXPARALLEL block.

Syntax

POSTFIXPARALLEL ( [ varName = ACCUMULATEVAR (threadVarName ); ]* );

Parameters

varName
Name of a VAR variable to store the sum of all the thread’s values of a specified THREADVAR variable.

ACCUMULATEVAR
Keyword to add up all the thread values of a specified THREADVAR variable. The sum is then assigned to a specified VAR variable.

threadVarName
Name of a THREADVAR variable.

Notes

To copy temporary THREADVAR values into VAR variables you can use outside FIXPARALLEL, use the following task flow:

1. Declare a VAR variable (outside of FIXPARALLEL block) to store the computed result.
2. Declare a THREADVAR variable that you use within the FIXPARALLEL block.
3. Use a POSTFIXPARALLEL block to copy the THREADVAR to the VAR.

Example

The following example accumulates Sales values from THREADVAR variables to a VAR variable.

/* Store computed result of four tasks */
VAR totalSalesAmnt = 0;
/* Four tasks */
FIXPARALLEL (2, "New York", "California", "Oregon", "Florida")
/* Accumulate results of tasks into threads */
THREADVAR s_entitySalesAmnt;
/* Use for computation in each task */
THREADVAR entitySalesAmnt;
/* Use/change THREADVARS within member formula blocks */
"Sales"
{
    /* Initialize variables for this task */
    entitySalesAmnt = 2;
    /* Use the THREADVARS ... */
    /* Accumulate task-data into thread-data */
    s_entitySalesAmnt = s_entitySalesAmnt + entitySalesAmnt;
}
/* Copy computed data into longer-persisting VAR */
POSTFIXPARALLEL ( totalSalesAmnt = ACCUMULATEVAR ( s_entitySalesAmnt ););
SET Commands

SET commands in a calculation script are procedural. The first occurrence of a SET command in a calculation script stays in effect until the next occurrence of the same SET command.

Example

In the following example, Essbase displays messages at the DETAIL level when calculating the Year dimension. However, when calculating the Measures dimension, Essbase displays messages at the SUMMARY level.

```
SET MSG DETAIL; CALC DIM(Year);
SET MSG SUMMARY; CALC DIM(Measures);
```

In the following example, Essbase calculates member combinations for Qtr1 with the SET AGGMISSG setting turned on. Essbase then does a second calculation pass through the database and calculates member combinations for East with the AGGMISSG setting turned off.

```
SET AGGMISSG ON; Qtr1;
SET AGGMISSG OFF; East;
```

SET AGGMISSG

Specifies whether Essbase consolidates #MISSING values in the database.

The default behavior is determined by the global setting for the database. For more information, see Specifying Global Settings for a Database Calculation in Designing and Maintaining Essbase Cubes.

**Syntax**

```
SET AGGMISSG ON | OFF ;
```

**Example**

```
SET AGGMISSG OFF;
CALC ALL;
```
SET CACHE

Specifies the size of the calculator cache.

Syntax

```
SET CACHE HIGH | DEFAULT | LOW | OFF | ALL;
```

Parameters

**HIGH, DEFAULT, and LOW**
Levels defining the size of the calculator cache. You set the values of HIGH, DEFAULT and LOW in the Essbase configuration settings. If you do not set the value of DEFAULT, Essbase uses a default value of 200,000 bytes. The maximum calculator cache size that you can specify is 200,000,000 bytes.

**OFF**
Essbase does not use the calculator cache.

**ALL**
Essbase uses the calculator cache, even when you do not calculate at least one full sparse dimension.

⚠️ Caution:
Forcing use of the calculator cache inside a **FIXPARALLEL** statement could increase calculation time.

Notes

Essbase uses the calculator cache to create and track data blocks during calculation. Using the calculator cache significantly improves your calculation performance. The size of the performance improvement depends on the configuration of your database.

You can choose one of three levels. The size of the calculator cache at each level is defined using the **CALCCACHE {HIGH | DEFAULT | LOW}** settings in the Essbase configuration.

The level you choose depends on the amount of memory your system has available and the configuration of your database.

You can specify whether, by default, Essbase uses a calculator cache using **CALCCACHE TRUE | FALSE** in the Essbase configuration. By default, **CALCCACHE** is set to **TRUE**.

Essbase uses the calculator cache providing that:

- Your database has at least two sparse dimensions.
- You calculate at least one, full sparse dimension (unless you specify the **CALCCACHE ALL** option).

You can use this command more than once within a calculation script.

You can display the calculator cache setting using the **SET MSG** command.
Example

If the Essbase configuration contains the following settings:

```
CALCCACHEHIGH  1000000
CALCCACHEDEFAULT  300000
CALCCACHELOW  200000
```

Then:

```
SET CACHE HIGH;
```

Sets a calculator cache of up to 1,000,000 bytes for the duration of the calculation script.

```
SET CACHE DEFAULT;
```

Sets a calculator cache of up to 300,000 bytes for the duration of the calculation script.

```
SET CACHE LOW;
```

Sets a calculator cache of up to 200,000 bytes for the duration of the calculation script.

```
SET CACHE ALL;
SET CACHE LOW;
```

Sets a calculator cache of 200,000 bytes to be used even when you do not calculate at least one, full sparse dimension.

```
SET CACHE OFF;
```

Specifies that Essbase does not use a calculator cache.

See Also

- CALCCACHE
- CALCCACHEHIGH
- SET MSG

SET CALCDIAGNOSTICS

Enables diagnostic logging for parallel calculation tasks.

Enabling diagnostic logging instructs Essbase to log the calculation time of the first `numTasks` longest-running parallel tasks.
Syntax

SET CALCDIAGNOSTICS { LOGSIZE numTasks; };

Parameters

LOGSIZE
A required keyword.

numTasks
How many of the top longest-running tasks to log. To disable diagnostic logging in the calculation script, set numTasks to 0.

Notes

• Diagnostics logging is not on by default, because it has performance overhead. After you are finished designing or optimizing your calculation script, you should turn off diagnostic logging.
• When used inside a FIXPARALLEL block, this command only takes effect within that block.

Example

The following example enables diagnostic logging for all parallel calculations in the calculation script.

SET CALCDIAGNOSTICS { LOGSIZE 4; };

FIXPARALLEL (2, @IDESCENDANT("US_Market"))
  AGG ("Product");
ENDFIXPARALLEL

The following example enables diagnostic logging for a specific FIXPARALLEL block.

FIXPARALLEL (2, @IDESCENDANT("US_Market"))
  SET CALCDIAGNOSTICS { LOGSIZE 4; }
  AGG ("Product");
ENDFIXPARALLEL

Sample Diagnostic Log Output for FIXPARALLEL

The following sample output pertains to FIXPARALLEL parallel calculation.

OK/INFO - 1012899 - Statistics for [Calc1.csc], FIXPARALLEL of index [1] at line [14]: Number of FIXPARALLEL Threads = [2], Total Tasks = [261], Min/Max/Avg Thread's Time = [103.453]/[103.519]/[103.486] secs.
OK/INFO - 1012899 - For [4] Longest tasks, next rows display : Time(secs), Thread_id, (Task_index/Task_count), Task_id, Member-combinations.
OK/INFO - 1012899 - 15.131, 1, (30/132), 53, [ID_051341].
OK/INFO - 1012899 - 10.759, 2, (124/129), 211, [ID_050092].
The diagnostic output is organized into 3 sections.

Log Section 1

The following section contains general information about the command being diagnosed.

OK/INFO - 1012899 - Statistics for [Calc1.csc], FIXPARALLEL of index [1] at line [14]: Number of FIXPARALLEL Threads = [2], Total Tasks = [261], Min/Max/Avg Thread's Time = [103.453]/[103.519]/[103.486] secs.

- **Calc script name**: Calc1.csc
- **Command ID**: FIXPARALLEL at index[1] (the first FIXPARALLEL command in Calc1.csc)
- **Other information**: Up to 2 threads are used for this calculation. It contains 261 parallel tasks. The calculation time is about 104 seconds.

Log Section 2

The following section contains information about the longest running tasks.

OK/INFO - 1012899 - For [4] Longest tasks, next rows display : Time(secs), Thread_id, (Task_index/Task_count), Task_id, Member-combinations.

The per-task diagnostic information is in columnar format. The following table describes each column, to help you interpret the data.
Table 3-3  Calc Diagnostic Output Columns

<table>
<thead>
<tr>
<th>Output Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Diagnostic Message ID</td>
<td>The message ID. For example, OK/INFO - 1012899. This ID can be used to extract diagnostic information from the application log into a file.</td>
</tr>
<tr>
<td>Time (secs)</td>
<td>Task execution time in seconds. For example, 15.131. The tasks are listed in decreasing order based on execution time.</td>
</tr>
<tr>
<td>Thread ID</td>
<td>Calculation thread ID. For example, 1. This calculation uses up to 2 threads, so the thread ID will always be 1 or 2.</td>
</tr>
<tr>
<td>Task Index/Task Count</td>
<td>The task index and the total task count. For example, 30/132, which indicates that this is the 30th task executed by this thread, and that this thread executes a total of 132 tasks.</td>
</tr>
<tr>
<td>Task ID</td>
<td>The task ID number. For example, 53. The first task has an ID of 1, but 53 is listed first because it was the longest running task. Note that as indicated by Log Section 1, there are 261 total tasks.</td>
</tr>
<tr>
<td>Member Combinations</td>
<td>The member names that form the slice corresponding to a task ID. For example, 53, [ID_051341] means that this calculation task is defined by the slice specified by task 53 and the member [ID_051341].</td>
</tr>
</tbody>
</table>

Log Section 3

The following section contains a summary of information already shown in Section 2, but groups the information per separate thread.

OK/INFO - 1012899 - Summary for thread[1]: Total Time = [103.519] secs, Total Tasks = [132].
OK/INFO - 1012899 - Longest tasks executing on thread[1]: Time(secs), Thread_id, (Task_index/Task_count), Task_id.
OK/INFO - 1012899 - 15.131, 1, (30/132), 53.
OK/INFO - 1012899 - 7.192, 1, (38/132), 105.
OK/INFO - 1012899 - 9.690, 1, (42/132), 125.
OK/INFO - 1012899 - Summary for thread[2]: Total Time = [103.453] secs, Total Tasks = [129].
OK/INFO - 1012899 - Longest tasks executing on thread[2]: Time(secs), Thread_id, (Task_index/Task_count), Task_id.
OK/INFO - 1012899 - 10.759, 2, (124/129), 211.

Related Topics
•  FIXPARALLEL...ENDFIXPARALLEL
SET CALCPARALLEL

Enables parallel calculation in place of the default serial calculation. Essbase analyzes each pass of a calculation to determine whether parallel calculation is possible. If it is not, Essbase uses serial calculation. This setting is not supported in hybrid aggregation mode, which is the default query processing mode.

Syntax

```plaintext
SET CALCPARALLEL n;
```

Parameters

`n`
A required parameter, an integer from 1 to 128, specifying the number of threads to be made available for parallel calculation. The default value specifies serial calculation: no parallel calculation takes place. Values 1 to 128 specify parallel calculation with 1 to 128 threads. Values of 0 specify serial calculation. Values less than 0 return an error. Values greater than the maximum are interpreted as the maximum (128).

Notes

- If a specific calculation pass has stored members depending on dynamic members, Essbase uses hybrid aggregation mode. In this case, SET CALCPARALLEL is ignored, and the calculation will run in serial. Alternately, you can use FIXPARALLEL, which is supported in hybrid aggregation mode.
- If you need to use the DATACOPY, DATAEXPORT, or CLEARBLOCK commands, use FIXPARALLEL...ENDFIXPARALLEL for parallel calculation instead of this command.
- A number of features are affected by parallel calculation. See CALCPARALLEL Parallel Calculation Guidelines in *Designing and Maintaining Essbase Cubes* for a list of these effects and for detailed information about how Essbase performs parallel calculation.

Example

```plaintext
SET CALCPARALLEL 3;
```

Enables up to three threads to be used to perform calculation tasks at the same time.

SET CALCTASKDIMS

Specifies the number of sparse dimensions included in the identification of tasks for parallel calculation.
Syntax

SET CALCTASKDIMS n;

Parameters

n
A required parameter, an integer specifying the number of sparse dimensions to be included when Essbase identifies tasks that can be performed at the same time. A value of 1 indicates that only the last sparse dimension in the outline will be used to identify tasks. A value of 2, for example, indicates that the last and second-to-last sparse dimensions in the outline are used. Because each unique combination of members from the selected sparse dimensions is a potential task, the potential number of parallel tasks is the product of the number of members of the selected dimensions. The maximum value is the number of sparse dimensions in the outline. Essbase issues an error if the value is less than 1. A value greater than the number of sparse dimensions in the outline is interpreted as the largest valid value. Using the calculator bitmap cache can affect this value. See Calculator Cache in Designing and Maintaining Essbase Cubes.

Notes

• A number of features are affected by parallel calculation. See Relationship Between CALCPARALLEL Parallel Calculation and Other Essbase Features in Designing and Maintaining Essbase Cubes for a list of these effects and for detailed information about how Essbase performs parallel calculation.
• Use the SET CALCTASKDIMS calculation command only if your outline generates many empty tasks, thus reducing opportunities for parallel calculation.
• If you do not notice an improvement in performance after increasing the value of SET CALCTASKDIMS, consider returning the value to the optimal number that Essbase selected. Sometimes using more task dimensions can generate such a large number of tasks that performance may decrease instead of increase, because the overhead of generating and managing the tasks is too great. See Identifying Additional Tasks for Parallel Calculation and Tuning CALCPARALLEL with Log Messages in Designing and Maintaining Essbase Cubes.

Example

SET CALCTASKDIMS 2;

Specifies that the last two sparse dimensions in the outline will be used to identify potential tasks to be performed at the same time during a calculation pass.

See Also

SET CALCPARALLEL

SET CLEARUPDATESTATUS

Specifies when Essbase marks data blocks as clean. This clean status is used during Intelligent Calculation.
Syntax

```plaintext
SET CLEARUPDATESTATUS AFTER | ONLY | OFF;
```

Parameters

**AFTER**
Essbase marks calculated data blocks as clean, even if you are calculating a subset of your database.

**ONLY**
Essbase marks the specified data blocks as clean but does not actually calculate the data blocks. This does the same as AFTER, but disables calculation.

**OFF**
Essbase does not mark the calculated data blocks as clean. Data blocks are not marked as clean, even on a default calculation (CALC ALL;) of your database. The existing clean or dirty status of the calculated data blocks remains unchanged.

Notes

SET CLEARUPDATESTATUS specifies when Essbase marks data blocks as clean.

The data blocks in your database have a calculation status of either clean or dirty. When Essbase does a full calculation of your database, it marks the calculated data blocks as clean. When a data block is clean, Essbase will not recalculate the data block on subsequent calculations, provided that Intelligent Calculation is turned on.

To ensure the accuracy of your calculation results, consider carefully the effect of the SET CLEARUPDATESTATUS AFTER command on your calculation.

If you do not use SET CLEARUPDATESTATUS, Essbase does not mark calculated data blocks as clean when you calculate a subset of your database. Essbase marks data blocks as clean only on a full calculation (CALC ALL;) or when Essbase calculates all members in a single calculation pass through your database.

If you calculate a subset of your database, you may want to use the SET CLEARUPDATESTATUS AFTER command to ensure that the calculated blocks are marked as clean. However, consider carefully the effect of this command on your calculation to ensure that your calculation results are correct.

Warnings

When you use the SET CLEARUPDATESTATUS command to mark calculated data blocks as clean, consider carefully the following questions:

Which data blocks are calculated?

Only calculated data blocks are marked as clean.

Are concurrent calculations going to affect the same data blocks?

Do not use the SET CLEARUPDATESTATUS AFTER command with concurrent calculations unless you are certain that the different calculations do not need to calculate the same data block or blocks. If concurrent calculations attempt to calculate the same data blocks, with Intelligent Calculation turned on, Essbase may not recalculate the data blocks, because they are already marked as clean.
Are the same data blocks to be recalculated on a second calculation pass through the database?

If you calculate data blocks on a first calculation pass through your database, Essbase marks them as clean. If you then attempt to calculate the same data blocks on a subsequent pass with Intelligent Calculation turned on, Essbase does not recalculate the data blocks, because they are already marked as clean.

Example

The following examples are based on the Sample Basic database. They assume that Intelligent Calculation is turned on (the default). For information on turning Intelligent Calculation on and off, see the SET UPDATECALC command.

Example 1

SET CLEARUPDATESTATUS AFTER;
FIX (*New York*)
CALC DIM(Product);
ENDFIX

New York is a member on the sparse Market dimension. Essbase searches for dirty parent data blocks for New York (for example "New York"->Colas in which Colas is a parent member). It calculates these dirty blocks based on the Product dimension and marks them as clean. Essbase does not mark the child, Input blocks as clean, because they are not calculated.

Example 2

SET CLEARUPDATESTATUS ONLY;
CALC ALL;

Essbase searches for all the dirty blocks in the database and marks them as clean. It does not calculate the blocks, even though a CALC ALL; command is used.

Example 3

SET CLEARUPDATESTATUS ONLY;
FIX (*New York*)
CALC DIM(Product);
ENDFIX

New York is a member on the sparse Market dimension. Essbase searches for dirty parent data blocks for New York (for example "New York"->Colas in which Colas is a parent member). It marks them as clean. It does not calculate the data blocks. It does not mark the child blocks as clean because they are not calculated. For example, if "New York"->100-10

is dirty, it remains dirty.
Example 4

```plaintext
SET CLEARUPDATESTATUS OFF;
CALC ALL;
CALC TWOPASS;
```

Essbase calculates all the dirty data blocks in the database. The calculated data blocks remain dirty; Essbase does not mark them as clean. Essbase then calculates those members tagged as Two-Pass on the dimension tagged as Accounts. Again, it does not mark the calculated data blocks as clean.

Related Topics

• SET UPDATECALC
• SET Commands

SET COPYMISSINGBLOCK

Sets whether the DATACOPY calculation command creates #MISSING blocks during the copy of data from a dense dimension.

This setting does not apply to aggregate storage databases.

SET COPYMISSINGBLOCK allows DATACOPY to avoid creating #MISSING blocks during the copy of data from a dense dimension.

Using DATACOPY on a dense dimension can create blocks populated with #MISSING. This is done deliberately in some instances, because most batch calculations operate only on existing data blocks. Therefore, DATACOPY can be used to ensure that all necessary data blocks are created prior to batch calculation.

But if the creation of #MISSING blocks is not required, you may want to avoid the increase in database size, and the possibly slower performance that results when, for example, a default calculation visits every #MISSING block.

Syntax

```plaintext
SET COPYMISSINGBLOCK ON | OFF
```

Parameters

**ON**
This is the default value. Allows missing blocks to be created during a data copy.

**OFF**
Suppresses the creation of missing blocks during a data copy.

Notes

• Existing #MISSING blocks are not removed.
• A message is added to the Essbase Server log to indicate the number of data blocks being copied from the source data blocks. The number of #MISSING blocks skipped, if any, is also reported in the log.
Example

SET COPYMISSINGBLOCK OFF;

The following log message indicates that SET COPYMISSINGBLOCK is OFF:

[Fri May 31 10:35:03 2002]Local/Test6/Test6/essexer/Info(1012574)
Datacopy command copied [1] source data blocks to [0] target data blocks

[Fri May 31 10:35:03 2002]Local/Test6/Test6/essexer/Info(1012576)
Datacopy command skipped creating [1] target data blocks with
CopyMissingBlock OFF

See Also
DATACOPY

SET CREATENONMISSINGBLK

Controls whether potential blocks are created in memory for calculation purposes, and whether #MISSING blocks are stored. It affects the results of calculations on sparse and dense dimensions.

By default, Essbase applies dense-member formulas only to existing data blocks. SET CREATENONMISSINGBLK ON enables Essbase to create potential blocks in memory where the dense-member formulas are performed. Of these potential blocks, Essbase writes to the database only blocks that contain values; blocks resulting in only #MISSING are not written to the database.

The creation of #MISSING blocks resulting from sparse-member formulas is governed by SET CREATEBLOCKONEQ. SET CREATENONMISSINGBLK ON ensures that only non-empty blocks are created, regardless of the Create Block on Equations setting.

In order to create new blocks, setting SET CREATENONMISSINGBLK to ON requires Essbase to anticipate the blocks that will be created. Working with potential blocks can affect calculation performance. Consider the following situations carefully:

• When SET CREATENONMISSINGBLK is ON, all sparse-member formulas are executed in top-down mode. Dense member formulas are flagged for top-down calculation when they contain the following:
  – Sparse members
  – Constants (for example, Sales = 100,000)
  – @VAR
  – @XREF
• If Essbase encounters @CALCMODE(BOTTOMUP) in a member formula, it ignores @CALCMODE.
• If a batch calculation contains top-down formulas and SET CREATENONMISSINGBLK is ON, Intelligent Calculation is turned off. Within the scope of the calculation script, all blocks are calculated, regardless if they are marked clean or dirty.
• To reduce the number of blocks to be calculated, use this command within
\texttt{FIX...ENDFIX}\texttt{regions}. As a warning, when the potential number of blocks exceeds
20 million, Essbase writes an entry to the application log showing the number of
blocks to be calculated and recommending using \texttt{FIX/ENDFIX}.

• You can use multiple \texttt{SET CREATENONMISSINGBLK} commands in a calc script,
each affecting calculations that follow. However, consider that each time \texttt{SET}
\texttt{CREATENONMISSINGBLK} is encountered within a set of \texttt{FIX} and \texttt{ENDFIX}
statements, the calculator cycles through the database, potentially affecting
calculation performance.

Syntax

\texttt{SET CREATENONMISSINGBLK ON|OFF;}

Parameters

\textbf{ON}
Calculations are performed on potential blocks as well as existing blocks. If the result
of the calculation is not \#MISSING, the block is stored. The Create Blocks on
Equations setting is ignored.

\textbf{OFF}
Calculations are performed only on existing blocks. This is the default setting.

Notes

• \texttt{SET CREATENONMISSINGBLK} affects only creation of new blocks. If existing
blocks become \#MISSING after formula execution, they are not deleted.

• The value set by \texttt{SET CREATENONMISSINGBLK} stays in effect until the next
\texttt{SET CREATENONMISSINGBLK} is processed, or the calculation script terminates.

• When the calculation script includes both \texttt{SET CREATENONMISSINGBLK ON}
and \texttt{SET MSG DETAIL}, any non-stored \#MISSING block is indicated in the
application log.

• If \texttt{SET MSG} is set to \texttt{SUMMARY}, when \texttt{SET CREATENONMISSINGBLK} is set to
\texttt{ON}, Essbase writes an entry to the application log stating that Create Non
\#MISSING Blocks is enabled.

• If \texttt{SET MSG} is set to \texttt{SUMMARY}, and \texttt{SET CREATENONMISSINGBLK} is set to
\texttt{ON}, at the end of the calculation, Essbase writes an entry to the application log
showing the total number of \#MISSING blocks that were not created.

Example

The following example is based on a variation of Sample Basic. Assume that the
Scenario dimension, of which Actual is a member, is sparse. "Jan Rolling YTD Est" is
a member of the dense time dimension, Year.

\texttt{FIX (Budget)}
\texttt{SET MSG DETAIL;}
\texttt{SET CREATENONMISSINGBLK ON;}
\texttt{"Jan Rolling YTD Est"= (Jan->Actual+Feb+Mar+Apr+May+Jun+Jul+Aug+Sep+Oct}
+Nov+Dec);}
\texttt{ENDFIX}
See Also

SET CREATEBLOCKONEQ

SET CREATEBLOCKONEQ

Controls, within a calculation script, whether new blocks are created when a
calculation formula assigns anything other than a constant to a member of a sparse
dimension. SET CREATEBLOCKONEQ overrides the Create Block on Equation
setting for the database.

Syntax

SET CREATEBLOCKONEQ ON|OFF;

Parameters

ON
When a formula assigns a non-constant value to a sparse dimension member for
which no block exists, Essbase creates a block.

OFF
When a formula assigns a non-constant value to a sparse dimension member for
which no block exists, Essbase does not create a block.

Notes

If calculations result in a value for a sparse dimension member for which no block
exists, Essbase creates a block. Sometimes, new blocks are not desired; for example,
when they contain no other values. In large databases, creation and processing of
unneeded blocks can increase processing time and storage requirements.

The Create Blocks on Equation setting is designed for situations when blocks would
be created as a result of assigning something other than a constant to a member of a
sparse dimension. For example, when Create Blocks on Equation is ON and West is
assigned a value where it did not have a value before, new blocks are created. When
this setting is OFF, blocks are not created.

Create Blocks on Equation setting is a database property. Its initial value is OFF; no
blocks are created when something other than a constant is assigned to a sparse
dimension member. For more information about enabling Create Blocks on Equation,
see the MaxL documentation in the Technical Reference for Oracle Essbase.

For more specific control, you can use the SET CREATEBLOCKONEQ calculation
command within a calculation script to control creation of blocks at the time the
command is encountered in the script. Use of SET CREATEBLOCKONEQ has the
following characteristics:

• When Essbase encounters SET CREATEBLOCKONEQ within a calculation script,
the database-level setting is ignored.

• You can use multiple SET CREATEBLOCKONEQ commands in the calculation
script to define the Create Blocks on Equation setting value for the calculations
following each command.
• The value set by the SET CREATEBLOCKONEQ command stays in effect until the next SET CREATEBLOCKONEQ command is processed or the calculation script is finished.

• The Create Blocks on Equation setting is overridden by SET CREATENONMISSINGBLK ON.

• The SET CREATEBLOCKONEQ command does not change the database-level Create Blocks on Equation property.

• If no SET CREATEBLOCKONEQ command is encountered, Essbase uses the database-level setting to determine whether to create blocks.

When the Create Blocks on Equation setting is ON, Essbase uses the top-down calculation method to calculate each sparse member.

The Create Blocks on Equation setting is not consulted when Essbase assigns constants to members of sparse dimensions. The following table shows examples of sparse member calculations where constants or non-constants are assigned to them.

**Table 3-4 Examples of Constant and Non-constant Assignments on Sparse Member Calculations**

<table>
<thead>
<tr>
<th>Assigned Value</th>
<th>Sparse Member Formula Example</th>
<th>New Block Created?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant</td>
<td>West = 350</td>
<td>Yes</td>
</tr>
<tr>
<td>Non-constant</td>
<td>West = California + 120</td>
<td>Yes, if the Create Blocks on Equation setting is ON. Otherwise, no.</td>
</tr>
<tr>
<td>Non-constant</td>
<td>West = California * 1.05</td>
<td>Yes, if the Create Blocks on Equation setting is ON. Otherwise, no.</td>
</tr>
</tbody>
</table>

For a tip on controlling creation of blocks when you work with non-constants and sparse dimensions, see Nonconstant Values Assigned to Members in a Sparse Dimension in *Designing and Maintaining Essbase Cubes*.

**Example**

**Example 1**

The following example is based on Sample.Basic. Data is loaded for only one block: ("100-10", "New York").

```
SET MSG SUMMARY;
SET CREATEBLOCKONEQ OFF;
"300-10" = "100-10" + 100000;
```

This calculation creates the block ("300-10", "New York"). Upon export, the database exports two blocks: the loaded block, and the new block. The calculation runs bottom-up.
Example 2

The following example is based on Sample.Basic. Data is loaded for only one block: (“100-10”, “New York”).

```plaintext
SET MSG SUMMARY;
SET CREATEBLOCKONEQ ON;

"300-10" = "100-10" + 100000;
```

This calculation creates 25 new blocks: 300-10 crossed with 25 stored members from the Market dimension. Upon export, the database exports 26 blocks: the loaded block, and the 25 new blocks. The calculation runs top-down.

Comparison of Example 1 and Example 2

In Example 1, the calculation script writer may have hoped to turn block creation OFF by using this line:

```plaintext
SET CREATEBLOCKONEQ OFF;
```

However, the calculation script has to create at least the one dependent block, to be able to execute the assignment statement.

SET CREATEBLOCKONEQ OFF does not mute block creation in the case where dependent blocks are needed for the calculation; however, it mutes extraneous block creation.

In the case of Example 1, Essbase avoids creating blocks crossing the Product dimension with the Market dimension. In Example 2, those extra blocks are created.

See Also

SET CREATENONMISSINGBLK

SET DATAEXPORTOPTIONS

Specifies options for data export operations.

Syntax

```plaintext
SET DATAEXPORTOPTIONS
|
DataExportLevel ALL | LEVEL0 | INPUT;
DataExportDynamicCalc ON | OFF;
DataExportNonExistingBlocks ON | OFF;
DataExportDecimal n;
DataExportPrecision n;
DataExportColFormat ON | OFF;
DataExportColHeader dimensionName;
DataExportDimHeader ON | OFF;
DataExportRelationalFile ON | OFF;
```
DataExportOverwriteFile ON | OFF;
DataExportDryRun ON | OFF;
);

Notes

Each SET DATAXPORTOPTIONS command specifies a set of option values that are in place until the next SET DATAXPORTOPTIONS command is encountered. At that time, option values are reset to default and newly specified option values are set.

The option list must start with a left brace (\{) and end with a right brace followed by a semicolon (\});. Each option ends with a semicolon (;). The options can be listed in any order. When an option is not specified, the default value is assumed.

The options are described here in three categories:

• Content Options
• Output Format Options
• Processing Options

Content Options

DataExportLevel ALL | LEVEL0 | INPUT
• ALL—(Default) All data, including consolidation and calculation results.
• LEVEL0—Data from level 0 data blocks only (blocks containing only level 0 sparse member combinations).
• INPUT—Input blocks only (blocks containing data from a previous data load or grid client data-update operation). This option excludes dynamically calculated data. See also the DataExportDynamicCalc option.

In specifying the value for the DataExportLevel option, use these guidelines:

• The values are case-insensitive. For example, you can specify LEVEL0 or level0.
• Enclosing the value in quotation marks is optional. For example, you can specify LEVEL0 or “LEVEL0”.
• If the value is not specified, Essbase uses the default value of ALL.
• If the value is incorrectly expressed (for example, LEVEL 0 or LEVEL2), Essbase uses the default value of ALL.

Description

Specifies the amount of data to export.

DataExportDynamicCalc ON | OFF
• ON—(Default) Dynamically calculated values are included in the export.
• OFF—No dynamically calculated values are included in the report.

Description

Specifies whether a text data export excludes dynamically calculated data.

Notes:
Text data exports only. If DataExportDynamicCalc ON is encountered with a binary export (DATAEXPORT BINFILE ...) it is ignored. No dynamically calculated data is exported.

The DataExportDynamicCalc option does not apply to attribute values.

If DataExportLevel INPUT is also specified and the FIX statement range includes sparse Dynamic Calc members, the FIX statement is ignored.

**DataExportNonExistingBlocks** ON | OFF

- **ON**—Data from all possible data blocks, including all combinations in sparse dimensions, are exported.
- **OFF**—(Default) Only data from existing data blocks is exported.

**Description**

Specifies whether to export data from all possible data blocks. For large outlines with a large number of members in sparse dimensions, the number of potential data blocks can be very high. Exporting Dynamic Calc members from all possible blocks can significantly impact performance.

**DataExportPrecision** $n$

$n$ (Optional; default 16)—A value that specifies the number of positions in exported numeric data. If $n < 0$, 16-position precision is used.

**Description**

Specifies that the DATAEXPORT calculation command will output numeric data with emphasis on precision (accuracy). Depending on the size of a data value and number of decimal positions, some numeric fields may be written in exponential format; for example, $678123e+008$. You may consider using DataExportPrecision for export files intended as backup or when data ranges from very large to very small values. The output files typically are smaller and data values more accurate. For output data to be read by people or some external programs, you may consider specifying the DataExportDecimal option instead.

**Notes:**

- By default, Essbase supports 16 positions for numeric data, including decimal positions.
- The DataExportDecimal option has precedence over the DataExportPrecision option.

**Example**

```
SET DATAEXPORTOPTIONS {
   DataExportPrecision 6;
   DataExportLevel ALL;
   DataExportColHeader "Measures";
   DataExportDynamicCalc ON;
};
DATAEXPORT "File=", "output1.out";
```
Initial Data Load Values

"Sales" "COGS" "Margin" "Marketing" "Payroll" "Misc" "Total Expenses"
"Profit" "Opening Inventory" "Additions" "Ending Inventory" "Margin %"
"Profit %" "100-10" "New York"
"Jan" "Actual" 678123456.0 271123456.0 407123456.0 941234567890123456.0
51123456.0 0 145123456.0 262123456.0 2101123456.0 644123456.0 2067123456.0
60123456.098 38123456.6430
"Feb" "Actual" 64512358123 38712349012345 5112345112345678 14212345
24512354067123456 6191234520112345 60123437123456.98
"Mar" "Actual" 675 270 405 94 51 1 146 259 2041 742 2108 60
58.37037037037037
"Qtr1" "Actual" 1998 799 1199 278 153 2 433 766 2101 2005 2108
60.01001001001001 38.33833833833834

Exported Data Format

"Sales","COGS","Margin","Marketing","Payroll","Misc","Total
Expenses","Profit","Opening Inventory","Additions","Ending
Inventory","Margin %","Profit %","Profit per Ounce","100-10","New York"
"Jan","Actual",6.78123e+008,2.71123e+008,4.07e+008,9.41235e+017,5.11235e+
007,0,9.41235e+017,12.01121e+009,6.44123e+008,2.06712e+
+009,60.0186,-1.388e+011,-7.84362e+016
"Feb","Actual",64512358123,38712349012345,5112345112345678,14212345
24512354067123456,6191234520112345,60123437123456.98
"Mar","Actual",675,270,405,94,51,1,146,259,2041,742,2108,60
38.37037037037037

DataExportDecimal n
Where n is a value between 0 and 16.

If no value is provided, the number of decimal positions of the data to be exported is
used, up to 16 positions, or a value determined by the DataExportPrecision option if
that is specified.

Description

Specifies that the DATAEXPORT calculation command will output numeric data with
emphasis on legibility; output data is in straight text format. Regardless of the number
of decimal positions in the data, the specified number is output. It is possible the data
can lose accuracy, particularly if the data ranges from very large values to very small
values, above and below the decimal point.

Notes:

• By default, Essbase supports 16 positions for numeric data, including decimal
positions.
• If both the DataExportDecimal option and the DataExportPrecision option are
specified, the DataExportPrecision option is ignored.

Example

SET DATAEXPORTOPTIONS
{DataExportDecimal 4;
DataExportLevel "ALL";
DataExportColHeader "Measures";
DataExportDynamicCalc ON;
);
DATAEXPORT "File" "," "output1.out";

Initial Data Load Values

"Sales" "COGS" "Margin" "Marketing" "Payroll" "Misc" "Total Expenses"
"Profit" "Opening Inventory" "Additions" "Ending Inventory" "Margin %"
"Profit %" "100-10" "New York"

"Jan" "Actual" 678123456.0 271123456.0 51123456.0 145123456.0 262123456.0 2101123456.0 644123456.0 2067123456.0 60123456.029 38123456.6430

"Feb" "Actual" 645123 258123 387123 9012345 5112345 112345678 14212345 24512345 2067123456 61912345 20411234 601234 37123456.98

"Mar" "Actual" 675 270 405 94 51 146 259 2041 742 2108 60 38.37037037037037

"Qtr1" "Actual" 1998 799 1199 278 153 2 433 766 2101 2005 2108 60.01001001001001 38.33833833833834

Exported Data Format

"Sales","COGS","Margin","Marketing","Payroll","Misc","Total Expenses","Profit","Opening Inventory","Additions","Ending Inventory","Margin %","Profit %","Profit per Ounce"
"100-10","New York"

"Jan","Actual", 678123456.0000,271123456.0000,407000000.0000,941234567890123520.0000,51123456.0000,0.0000,941234567941246980.0000,-941234567534246910.0000,2101123456.0000,644123456.0000,2067123456.0000,60.0186,-138799883591.4395,-78436213961 187248.0000

"Feb","Actual", 645123.0000,258123.0000,387000.0000,9012345.0000,5112345.0000,112345678.0000,0.126470368.0000,-126083368.0000,2067123456.0000,61912345.0000,20411234.0000,0.59.9886,-19544.0820,-10506947.3333

"Mar","Actual", 675.0000,270.0000,405.0000,94.0000,51.0000,1.0000,146.0000,259.0000,2041.00 00,742.0000,2108.0000,60.0000,38.3704,21.5833

Output Format Options

DataExportColFormat ON | OFF

• ON—The data is output in columnar format.
• OFF—Default. The data is output in non-columnar format.

Description

Specifies if data is output in columnar format. Columnar format displays a member name from every dimension; names can be repeated from row to row, enabling use by
applications other than Essbase tools. In non-columnar format, sparse members identifying a data block are included only once for the block. Non-columnar export files are smaller, enabling faster loading to an Essbase database.

Notes

Do not use the DataExportColFormat option in combination with the DataExportRelationalFile option, which already assumes columnar format for files destined as input files to relational databases.

Example

SET DATAEXPORTOPTIONS
{DATAEXPORTCOLFORMAT ON;}
FIX("100-10", Sales, COGS, Jan, Feb, Mar, Actual, Budget)
DATAEXPORT "File" "," "test2.txt";
ENDFIX;

DataExportColHeader dimensionName

Description

Specifies the name of the dense dimension that is the column header (the focus) around which other data is referenced in the export file. Use the DataExportColHeader option only when you export data to a text file. For example, if from Sample Basic the Year dimension is specified, the output data starts with data associated with the first member of the Year dimension: Year. After all data for Year is output, it continues with the second member: Qtr1, and so on.

Notes

MaxL, ESSCMD shell, and Essbase exports do not provide a similar capability. With these methods, Essbase determines the focal point of the output data.

Exporting through Report Writer enables you to specify the header in the report script.

Example

SET DATAEXPORTOPTIONS {DATAEXPORTCOLHEADER Scenario;};

Specifies Scenario as the page header in the export file. The Scenario dimension contains three members: Scenario, Actual, and Budget. All Scenario data is shown first, followed by all Actual data, then all Budget data.

DataExportDimHeader ON | OFF

- ON—The header record is included.
- OFF—Default. The header record is not included.

Description

Use the DataExportDimHeader option to insert the optional header record at the beginning of the export data file. The header record contains all dimension names in the order as they are used in the file. Specifying this command always writes the data in "column format".
Example

SET DATAEXPORTOPTIONS
{
  DATAEXPORTLEVEL "ALL";
  DATAEXPORTDIMHEADER ON;
};
FIX("100-10", "New York", "Actual")
DATAEXPORT "File" "," "2222.txt" ;
ENDFIX;

Specifying the DataExportDimHeader ON option while exporting Sample Basic writes the data in column format, with common members repeated in each row. The data begins with a dimension header, as shown in the first two rows of the example file below:

"Product","Market","Year","Scenario","Measures"
"Sales","COGS","Marketing","Payroll","Misc","Opening Inventory","Ending Inventory"
"100-10","New York","Jan","Actual",678,271,94,51,0,2101,644,2067
"100-10","New York","Feb","Actual",645,258,90,51,1,2041,619,2041
"100-10","New York","Mar","Actual",675,270,94,51,1,2041,742,2108
"100-10","New York","Apr","Actual",712,284,99,53,1,2108,854,2250
"100-10","New York","May","Actual",756,302,105,53,1,2250,982,2476
"100-10","New York","Jun","Actual",890,356,124,53,0,2476,1068,2654
"100-10","New York","Jul","Actual",912,364,127,51,0,2654,875,2617
"100-10","New York","Aug","Actual",910,364,127,51,0,2617,873,2580
"100-10","New York","Sep","Actual",790,316,110,51,1,2580,758,2548
"100-10","New York","Oct","Actual",650,260,91,51,1,2548,682,2580
"100-10","New York","Nov","Actual",623,249,87,51,0,2580,685,2642
"100-10","New York","Dec","Actual",699,279,97,51,1,2642,671,2614

DataExportRelationalFile ON | OFF
• ON—The output text export file is formatted for import to a relational database.
  – Data is in column format; sparse member names are repeated. (The DataExportColFormat option is ignored.)
  – The first record in the export file is data. No dimension header is included, even if specified using the DataExportDimHeader option. No columns are labeled in the first row of the relational export file, even if the DataExportColHeader option is used; however, the dense dimension specified in the DataExportColHeader option is the focus around which other data is referenced in the export file.
  – Missing and invalid data is skipped, resulting in consecutive delimiters (commas) in the output. The optional "missing_char" parameter for DATAEXPORT is ignored.
• OFF—Default. The data is not explicitly formatted for use as input to a relational database.

Description

Using the DataExportRelationalFile option with DATAEXPORT enables you to format the text export file to be used directly as an input file for a relational database.
Example

SET DATAEXPORTOPTIONS {
    DataExportLevel "ALL";
    DataExportRelationalFile ON;
};

FIX (Jan)
    DATAEXPORT "File" "," "jan.txt"
ENDFIX;

Processing Options

DataExportOverwriteFile ON | OFF

- ON—The existing file with the same name and location is replaced.
- OFF—Default. If a file with the same name and location already exists, no file is output.

Description

Manages whether an existing file with the same name and location is replaced.

DataExportDryRun ON | OFF

- ON—DATAEXPORT and associated commands are run, without exporting data.
- OFF—Default. Data is exported

Description

Enables running the calculation script data export commands to see information about the coded export, without exporting the data. When the DataExportDryRun option value is ON, the following information is written to the output file specified in the DATAEXPORT command:

- Summary of data export settings
- Info, Warning, and Error messages
- Exact number of blocks to be exported
- Estimated time, excluding I/O time.

Notes

- The DataExportDryRun option does not work with exports to relational databases.
- If you modify the script for reuse for the actual export, besides removing the DataExportDryRun option from the script you may want to change the name of the export file.

Example

SET DATAEXPORTOPTIONS {
    DataExportLevel "ALL";
    DataExportColHeader "Measures";
    DataExportColFormat ON;
    DataExportDimHeader ON;
    DataExportDynamicCalc OFF;
}
DataExportDecimal 0;
DataExportDryRun ON;
DataExportOverwriteFile ON;
};

FIX("Qtr1")
DATAEXPORT "File" "," "log.txt" ;
ENDFIX;

Creates the file "E:\temp\log.txt" containing the following information:

<EXPORT_OPTIONS>
 <DELIMITER>
   ,
 </DELIMITER>
 <MISSING_VALUE>
   #Mi
 </MISSING_VALUE>
 <EXPORT_LEVEL>
   ALL
 </EXPORT_LEVEL>
 <DYNAMIC_CALC_EXPORT>
   OFF
 </DYNAMIC_CALC_EXPORT>
 <COLUMN HEADER>
   Measures
 </COLUMN HEADER>
 <COLUMN_FORMAT>
   ON
 </COLUMN_FORMAT>
 <DIMENSION_HEADER_WRITE>
   ON
 </DIMENSION_HEADER_WRITE>
 <FILE_OVERWRITE>
   ON
 </FILE_OVERWRITE>
 <DECIMAL_POINT>
   ON
 </DECIMAL_POINT>
 <PRECISION POINT>
   16
 </PRECISION POINT>
 <RELATIONAL_EXPORT>
   OFF
 </RELATIONAL_EXPORT>
</EXPORT_OPTIONS>

<MESSAGE>
 <INFO>
   DataExport Warning: FIX statement contains Dynamic Calc member [Qtr1]. No Dynamic Calc members are exported with the DataExportDynamicCalc option set to OFF.
 </INFO>

 <INFO>
   Data Export Completed. Total blocks: [332]. Elapsed time: [3.846]
Specifies whether to ignore the outline timestamp captured at the time the data was exported.

Syntax

```plaintext
SET DATAIMPORTIGNORETIMESTAMP ON|OFF;
```

Parameters

**ON**
Ignore the outline timestamp.

**OFF**
Default. Check the outline timestamp.

Notes

The **DATAEXPORT** "Binfile" command captures the outline timestamp when it creates a binary export file. By default, when the file is imported, Essbase checks the import file timestamp against the existing outline timestamp to ensure the correct import file is read. You can use **SET DATAIMPORTIGNORETIMESTAMP** to bypass checking the timestamp.

**Caution:**
Bypassing the check enables potentially importing the wrong file.

Example

```plaintext
SET DATAIMPORTIGNORETIMESTAMP ON;
DATAIMPORTBIN e:january\basic.bin
```

Specifies to ignore comparing the outline timestamp with the timestamp on the import file, and to import the binary export file to the database on which the calculation script is running.

Related Topics

• **DATAEXPORT**
SET EMPTYMEMBERSETS

EMPTYMEMBERSETS stops the calculation within a `FIX...ENDFIX` command if the FIX evaluates to an empty member set.

**Syntax**

```
SET EMPTYMEMBERSETS ON|OFF
```

**Parameters**

**ON**
Calculation within FIX command stops if FIX evaluates to an empty member set.

**OFF**
Entire database is calculated, even if FIX evaluates to an empty member set.

**Notes**

If EMPTYMEMBERSETS is ON, and a FIX command evaluates to a empty member set, the calculation within the FIX command stops and the following information message is displayed: “FIX statement evaluates to an empty set. Please refer to SET EMPTYMEMBERSETS command.” The calculation resumes after the FIX command.

If a calculation script contains nested FIX commands, the nested FIX commands are not evaluated.

**Example**

The following calculation script does not calculate Calc Dim(Year) within the FIX command. 100-10 has no children and therefore the FIX statement evaluates to an empty member set.

```plaintext
SET EMPTYMEMBERSETS ON;
...
FIX(@CHILDREN("100-10"))
  Calc Dim(Year);
ENDFIX
...
```

The following calculation script has nested FIX commands. Calc Dim(Product) is not calculated because FIX(@CHILDREN("100-10")) evaluates to empty member set. Calc Dim(Year) is not calculated even though the nested FIX("New York") does not evaluate to an empty member set.

```plaintext
SET EMPTYMEMBERSETS ON;
...
FIX(@CHILDREN("100-10"))
  FIX("New York")
  Calc Dim(Year);
ENDFIX
Calc Dim(Product);
```
SET FRMLBOTTOMUP

Optimizes the calculation of complex formulas on sparse dimensions in large database outlines. This command tells Essbase to perform a bottom-up calculation on formulas that would otherwise require a top-down calculation.

You might want to turn on this setting when using the CALC ALL and CALC DIM commands to calculate the database.

Syntax

SET FRMLBOTTOMUP ON|OFF;

Parameters

ON
Turns on the bottom-up sparse formula calculation method.

OFF
Turns off the bottom-up sparse formula calculation method. The default setting is OFF.

Notes

• For information on complex formulas and top-down calculations, see Optimizing Calculations in Designing and Maintaining Essbase Cubes.

• Forcing a bottom-up calculation on a formula may produce results that are inconsistent with a top-down calculation if:
  – The formula contains complex functions (for example, range functions)
  – The formula's dependencies are not straightforward

• Before using this command in a production environment, be sure to check the validity of calculation results produced when the command is enabled (set to ON).

Example

SET FRMLBOTTOMUP ON;

SET FRMLRTDYNAMIC

Enables you to turn off calculation of all dense Dynamic Calc members during batch calculation if runtime dependent functions are included in formulas on stored members. (The preprocessing phase of a calculation script cannot determine if an outline contains dense Dynamic Calc members.)

This command improves batch calculation performance by removing the overhead of calculating all Dynamic Calc members.

The SET FRMLRTDYNAMIC command can be applied to an entire calculation script segment, as shown in the example below.
Syntax

SET FRMLRTDYNAMIC ON | OFF;

Parameters

ON
Calculation of Dynamic Calc members is performed. The default value is ON.

OFF
Calculation of Dynamic Calc members is not performed.

Notes

- Runtime-dependent functions include:
  - @ANCEST
  - @SANCEST
  - @PARENT
  - @SPARENT
  - @CURRMBR

- If a stored member formula includes a runtime-dependent function on a Dynamic Calc member, it may get #MISSING as the result instead of the expected value after executing the formula on the Dynamic Calc member.

Example

The following example turns off all dense Dynamic Calc members:

SET FRMLRTDYNAMIC OFF;
FIX(@LEVMBRS(Product, 0)))
"Avg Sales" = @AVGRANGE(SKIPNONE,Sales,@CHIDREN(@CURRMBR(Product)));
ENDFIX
CALC ALL;

SET MSG

Sets the level of messaging you want returned about calculations, and enables simulated calculations.

The SET MSG command applies only to the calculation script in which it is used.

Syntax

SET MSG SUMMARY | DETAIL | ERROR | INFO | NONE | ONLY;

Parameters

SUMMARY
Displays calculation settings and provides statistics on the number of:
- Data blocks created, read, and written
- Data cells calculated

**DETAIL**
Provides the same information as SUMMARY. In addition, it displays a detailed information message every time Essbase calculates a data block.

**ERROR**
Displays only error messages.

**INFO**
Displays information and error messages.

**NONE**
Displays no messages during the life of the calculation script. However, because error messages may contain vital information, they are still displayed.

**ONLY**
Instructs Essbase to perform a simulated calculation only. You may disregard any error message during validation that indicates Essbase does not recognize a command.

**Note:**
When you use this parameter, Essbase generates some empty upper-level blocks. Make sure to clear upper-level blocks (or non-input blocks if you load data into upper level blocks in your model) at the end of the simulation/command.

Oracle recommends using SET MSG ONLY with the calculation script commands SET NOTICE HIGH and CALC ALL.
SET MSG ONLY does not generate a completion notice.

**Notes**

SET MSG SUMMARY and SET MSG DETAIL tell you:
- The status of calculation settings (for example, whether completion notice messages are enabled)
- The total number of data blocks created
- The number of data blocks read and written on sparse calculations
- The number of data blocks read and written on dense calculations
- The number of data cells calculated on sparse calculations
- The number of data cells calculated on dense calculations

In addition, the SET MSG DETAIL command provides an information message every time Essbase calculates a data block. It is useful for testing your database's consolidation path. Because it causes a high processing overhead, it should be used during test calculations only.

SET MSG SUMMARY causes a processing overhead of approximately 1% to 5%, depending on the database size.
Example

SET MSG ERROR;

Displays only the error messages.

SET MSG SUMMARY;

Produces the following sample output:

[Tue Apr 4 05:11:16 1995] local/Sample/Basic/Qatest/Info(1012672)
Calculator Information Message:

Maximum Number of Lock Blocks: [100] Blocks
Completion Notice Messages: [Disabled]
Calculations On Updated Blocks Only: [Enabled]
Clear Update Status After Full Calculations: [Enabled]
Calculator Cache With Multiple Bitmaps For: [Market]

[Tue Apr 4 05:11:19 1995] local/Sample/Basic/Qatest/Info(1012672)
Calculator Information Message:

Total Block Created: [0.0000e+00] Blocks
Sparse Calculations: [4.3000e+01] Writes and [4.3000e+01] Reads
Dense Calculations: [4.3200e+02] Writes and [4.3200e+02] Reads
Sparse Calculations: [1.7200e+02] Cells
Dense Calculations: [4.3200e+02] Cells

SET MSG DETAIL;

Produces the following sample output:

Calculator Information Message:

Maximum Number of Lock Blocks: [100] Blocks
Completion Notice Messages: [Disabled]
Calculations On Updated Blocks Only: [Enabled]
Clear Update Status After Partial Calculations: [Disabled]
Calculator Cache With Multiple Bitmaps For: [Market]

Calculator Information Message: Executing Block - [100], [East]

Calculator Information Message: Executing Block - [Product], [East]

Calculator Information Message: Executing Block - [100], [Market]

Calculator Information Message: Executing Block - [Product], [Market]

Calculator Information Message:

Total Block Created: [0.0000e+00] Blocks
Dense Calculations: [0.0000e+00] Writes and [0.0000e+00] Reads
Sparse Calculations: [3.8080e+03] Cells
Dense Calculations: [0.0000e+00] Cells

See Also

• CLEARBLOCK
• SET NOTICE
• SET Commands

SET NOTICE

Monitors the progress of your calculation by providing completion notices at intervals during the calculation. The number of notices depends on the level you specify.

Syntax

SET NOTICE HIGH | DEFAULT | LOW;

Parameters

HIGH, DEFAULT, and LOW
Levels defining the frequency and number of completion notices.
Notes

- The interval between notices is approximate. Essbase measures the interval by taking the number of data blocks already calculated as a percentage of the total number of possible data blocks in your database. For example, if there are 10,000 possible blocks and you specify 5 notices, Essbase notifies you when the calculation approximately reaches block 2000, 4000, 6000, 8,000 and 10,000. However, if only the blocks 1,000 - 4,000 exist, then Essbase displays only two notices.
- For partial calculations and calculations with multiple passes through your database, the interval between completion notices is very approximate.
- Completion notices do not significantly reduce the calculation performance, except when used with a very small database.

Related Topics

- SET MSG

SET REMOTECALC

For applications with transparent partitions, turns remote calculation to the source on or off.

Syntax

```
SET REMOTECALC ON | OFF;
```

Parameters

ON
Default. Essbase connects to the source partition enabling remote calculations.

OFF
Essbase does not connect to the source partition. Use this option only when absolutely sure the calculation script does not involve access to remote data.

Notes

- When you are working with transparent partitions and are sure that a calculation script does not include remote values in the calculations, you can use SET REMOTECALC OFF to improve calculation performance.
- Performance improvement is visible only when batch calculation is run on the target application.

Example

```
SET REMOTECALC ON;
```

```
SET REMOTECALC OFF;
```
SET RUNTIMESUBVARS

Declares runtime substitution variables that are used in a calculation script.

Every runtime substitution variable used in a calculation script must be declared in the SET RUNTIMESUBVARS command, with a name and a default value. You can include a description of the runtime substitution variable's data type and data input limit, which is a string in the `<RTSV_HINT> rtsv_description </RTSV_HINT>` tag. Each runtime substitution variable declaration must end in a semicolon.

Syntax

```
SET RUNTIMESUBVARS
{
    runtime_substitution_variable [= value] [RTSV_HINT] rtsv_description [RTSV_HINT];
}
```

Parameters

**runtime_substitution_variable**
Name of a runtime substitution variable

**value**
Default value of the named runtime substitution variable. The value can be expressed as a string, a constant, a member name, or a member combination. Default values specified in the SET RUNTIMESUBVARS command can be overwritten at runtime. See Using Runtime Substitution Variables in Calculation Scripts Run in Essbase in *Designing and Maintaining Essbase Cubes*.

**<RTSV_HINT> rtsv_description </RTSV_HINT>**
A string that describes the data type and data input limit (for example, an integer not greater than 100) of the named runtime substitution variable. When running a calculation script that contains runtime substitution variables, the `<RTSV_HINT>` tag is:

- Optional, when running the calculation script in Essbase
- Required, when running the calculation script in Smart View

See Using Runtime Substitution Variables in Calculation Scripts Run in Essbase in *Designing and Maintaining Essbase Cubes*.

The IEssIterator.getCalcFileRunTimeSubVars or IEssIterator.getCalcRunTimeSubVars Java API methods or EssGetRuntimeSubVars C API retrieves all of the information (name, value, and description) that is specified in the runtime substitution variable declaration. The `<RTSV_HINT>` string can then be used to prompt a user to input a value at runtime or to validate input data before passing the value to the calculation script.

Notes

- If a default value is not included in the runtime substitution variable declaration in SET RUNTIMESUBVARS, an error occurs when the calculation script is validated. Oracle recommends that you provide a default value to avoid the validation error and, when running the calculation script, provide the expected value. However, if
you do not provide a default value, you can still provide a value at runtime using the execute calculation MaxL statement with the `with runtimesubvars` grammar.

- If you specify a runtime substitution variable in `SET RUNTIMESUBVARS` but do not use the runtime substitution variable in the calculation script, Essbase ignores the runtime substitution variable declaration.

- If multiple runtime substitution variables have the same name but have different values, only the value of the first instance of the runtime substitution variable is used; all other subsequent values are ignored.

- To log the runtime substitution variables that are used in a calculation script, set the `ENABLERTSVLOGGING` configuration setting to TRUE.

**Example**

In the following example, three runtime substitution variables are defined with a name and a default value; for example, the runtime substitution variable named `myMarket` has a value of "New York".

```plaintext
SET RUNTIMESUBVARS
{
  myMarket = "New York";
  salesNum = 100;
  pointD = "Actual"->"Final";
};
```

In the following example, the runtime substitution variables include a default value and `rtsv_description`. The `EssGetRuntimeSubVars` API can be implemented to retrieve all of the information (name, value, and description) about the runtime substitution variable. The `<RTSV_HINT>` string can then be used to prompt a user to input a value at runtime or to validate input data before passing the value to the calculation script.

```plaintext
SET RUNTIMESUBVARS
{
  myMarket "New York" <RTSV_HINT>myMarket: Input the value as a string, such as "New York"</RTSV_HINT>;
  salesNum 10 <RTSV_HINT>salesNum: Input the value as an integer, such as 100</RTSV_HINT>;
  pointD "Actual"->"Final" <RTSV_HINT>pointD: Input the value as a member name or a member combination, such as "Actual"->"Final"</RTSV_HINT>;
};
```

The following example shows the use of XML-style tags within the `<RTSV_HINT>` tag for running a calculation script with runtime substitution variables in Smart View:

```plaintext
SET RUNTIMESUBVARS
{
  sbx = POV <RTSV_HINT>
    <svLaunch>
      <description>Sandbox to merge</description>
      <allowMissing>false</allowMissing>
      <type>member</type>
      <dimension>Sandbox</dimension>
      <choice>single</choice>
  </svLaunch>
};
```
### SET SCAPERSPECTIVE

Sets the perspective for varying attribute calculations.

**Syntax**

```plaintext
SET SCAPERSPECTIVE (mbrName1) [, (mbrName2)] ... [(mbrName)] on Attribute_Dimension | OFF ;
```

**Parameters**

- **mbrName1 [...] on Attribute_Dimension**
  Any valid single member name, or list of member names, on the specified varying attribute dimension.

- **OFF**
  Turn off the perspective setting for the calculation block.

**Notes**

- For use only in applications enabled with varying attributes.
- Only one independent member from each independent dimension is supported.

**Example**

Once the perspective is specified using this command, `@WITHATTR` can be used on a varying attribute inside a FIX statement. In the following example, the SET SCAPERSPECTIVE statements indicate that for attribute dimensions TYPE and TITLE, the subsequent FIX statement with `@WithATTR` will use their attribute association as defined at time FY03 and Jan.

```plaintext
set SCAPerspective ((FY03), (Jan)) on TYPE;
set SCAPerspective ((FY03), (Jan)) on TITLE;
FIX (@WithAttr (TYPE, "==", Contractor), @withattr (Title, "==", Senior_QA_Engineer), Local, "HSP_Historical", "BU Version_1", Target, Local, FY03)
HSP_INPUTVALUE = 100;
ENDFIX;
```

**Related Topics**

- `@ISATTRIBUTE`
SET TRACE

This calculation command selects a particular cell to be traced during the execution of member formulas in a calculation script.

Description

SET TRACE enables you to trace multiple data cells. Additionally, you can trace sections of calculation scripts by using a combination of SET TRACE mbrList and SET TRACE OFF. However, to use SET TRACE command, you must execute the calculation script outside of Smart View, using Cube Designer or the Jobs page of the cloud service.

Syntax

SET TRACE mbrList| OFF;

Parameters

mbrList
A comma-delimited list of members, member set functions, or range functions. Must contain at least one member from each dimension.

OFF
Turns off the previous SET TRACE command in the script. SET TRACE OFF has no effect when calculation traces are run from Smart View.

Notes

- Tracing is not supported for CALC ALL or CALC DIM commands.
- Trace output is logged to calc_trace.txt in the database directory on the cloud service. This file is overwritten when the next calculation script is run or verified.
- SET TRACE commands are ignored if the CALCTRACE configuration setting is set to OFF.
- Even if the CALCTRACE configuration setting is ON, SET TRACE commands are ignored when calculation scripts are executed from Smart View.

Example

In the following example, the script traces the calculation of "Actual," "Opening Inventory," and "Ending Inventory" for Cola in New York for the months of January to March:

```
SET TRACE (@CHILDREN("Qtr1"), "Cola", "New York", "Actual", "Ending Inventory");

FIX(@LEVMBRS("Year",0), "Cola", "New York", "Actual")
"Opening Inventory" ( 
    IF (NOT @ISMBR("Jan")) 
        "Opening Inventory" = @PRIOR("Ending Inventory") 
    ELSE 
        "Opening Inventory" = @LEVMBRS("Year",0, "Cola", "New York", "Actual") 
    
```


The tracing output from the above script is:

Tracing cell: [100-10][New York][Actual][Jan][Ending Inventory]  (Cell update count: 1)

Previous value: #MI
Dependent values:
[100-10][New York][Actual][Jan][Opening Inventory] = 2101.00
[100-10][New York][Actual][Jan][Additions] = 644.00
[100-10][New York][Actual][Jan][Sales] = 678.00
New value: [100-10][New York][Actual][Jan][Ending Inventory] = 2067.00

Computed in lines: [8 - 14] using:
"Opening Inventory"
IF(NOT@ISMBR("Jan"))
"Opening Inventory"=@PRIOR("Ending Inventory");
ENDIF
"Ending Inventory"="Opening Inventory" + "Additions" - "Sales";
)

Tracing cell: [100-10][New York][Actual][Feb][Opening Inventory]  (Cell update count: 1)

Previous value: #MI
Dependent values:
[100-10][New York][Actual][Jan][Ending Inventory] = 2067.00
New value: [100-10][New York][Actual][Feb][Opening Inventory] = 2067.00

Computed in lines: [8 - 14] using:
"Opening Inventory"
IF(NOT@ISMBR("Jan"))
"Opening Inventory"=@PRIOR("Ending Inventory");
ENDIF
"Ending Inventory"="Opening Inventory" + "Additions" - "Sales";
)

...
SET UPDATECALC

Turns Intelligent Calculation on or off.

Syntax

```
SET UPDATECALC ON | OFF;
```

Parameters

**ON**
Essbase calculates only blocks marked as dirty (see Description). Dirty blocks include updated blocks and their dependent parents (see Notes).

**OFF**
Essbase calculates all data blocks, regardless of whether they have been updated.

Notes

- Using Intelligent Calculation, Essbase calculates only dirty blocks, such as updated data blocks and their dependent parents. Therefore, the calculation is very efficient.
- All data blocks in the database are marked as either clean or dirty. If a data block is clean, then Essbase knows that the block does not need to be recalculated.
- By default, all data blocks are marked as clean after a full calculation of the database but not after a partial calculation of the database. If required, you can change this default behavior using the SET CLEARUPDATESTATUS command in your calculation script.
- There are several possible reasons blocks might be marked as dirty. See Understanding Intelligent Calculation in *Designing and Maintaining Essbase Cubes* for information on Intelligent Calculation and clean and dirty blocks.

Example

```
SET UPDATECALC ON;

SET UPDATECALC OFF;
```

Related Topics

- SET CLEARUPDATESTATUS

THREADVAR

Declares one or more temporary, thread-level variables within a FIXPARALLEL...ENDFIXPARALLEL block.

Syntax

```
THREADVAR varName [, varName ] ;
```
Parameters

**varName**
Name of the temporary variable(s).

Notes

- THREADVAR variables must be declared within the `FIXPARALLEL...ENDFIXPARALLEL` block, and can only be used within that context.
- Essbase creates an instance of a THREADVAR variable for each child thread.
- A THREADVAR variable cannot be initialized; Essbase initializes it to `#MISSING`.
- A THREADVAR variable cannot have the same name as a VAR variable.

Example

See the example for `POSTFIXPARALLEL`.

Related Topics

- `FIXPARALLEL...ENDFIXPARALLEL`
- `POSTFIXPARALLEL`

**USE_MDX_INSERT**

For the current calculation script, enables execution of aggregate storage custom calculations and allocations through MDX Insert.

This command is applicable to aggregate storage calculation scripts only. It must be added as the first line of the custom calculation script.

Syntax

```
USE_MDX_INSERT;
```

Example

```
USE_MDX_INSERT;
[Original Price] := Units/7;
[Price Paid] := Units/7;
[Returns] := Units/7;
```

See Also

- Performing Custom Calculations and Allocations on Aggregate Storage Databases
- `CUSTOMCALCANDALLOCTHRUINSERT`
- MDX Insert Specification
VAR

Declares a temporary variable that contains a single value. The variable lasts for the scope of the calculation script.

Note:
You can also use a single VAR command to declare multiple variables by supplying a comma-delimited list of variable names.

Syntax

VAR varName [= value] ;

Parameters

varName
Name of the temporary variable.

value
Optional parameter that declares the data value.

Notes

- The name of the variable cannot duplicate a database member name.
- If a value is not declared, it is set to #MISSING.
- VAR commands can only be assigned values within a member calculation or when VAR is declared.

Example

VAR Target = 1200;

VAR Break1, Break2, Break3;

See Also

ARRAY
4

Configuration Settings

You can use a variety of configuration settings to customize the behavior of your applications.

- Configuration Settings Categorical List
- Aggregate Storage and Block Storage Settings Comparison
- Config Settings List

Configuration Settings Categorical List

This section lists all of the configuration settings, grouped categorically. Some may appear in more than one category.

- Calculation Configuration Settings
- Data Import and Export Configuration Settings
- Logging and Error Handling Configuration Settings
- Memory Management Configuration Settings
- Miscellaneous Configuration Settings
- Partitioning Configuration Settings
- Ports and Connections Configuration Settings
- Query Management Configuration Settings

Calculation Configuration Settings

- ASODYNAMICAGGINBSO
- ASODYNAMICAGGINBSOFOLDERPATH
- CALCCACHE
- CALCCACHEHIGH
- CALCCACHEDEFAULT
- CALCCACHELOW
- CALCLIMITFORMULARECURSION
- CUSTOMCALCANDALLOCTHRUINSERT
- DYNALCCACHEMAXSIZE
- FORCEALLDENSECALCON2PASSACCOUNTS
- HYBRIDSOINCALCSCRIPT
- RTDEPCALCOPTIMIZE
Data Import and Export Configuration Settings

- DLSINGLETHREADPERSTAGE
- DLTHREADSPREPARE
- DLTHREADSWRITE

Memory Management Configuration Settings

- DATACACHESIZE
- DYNALCCACHEMAXSIZE
- INDEXCACHESIZE
- MAXFORMULACACHESIZE
- NUMBLOCKSTOEEXTEND
- SSOPTIMIZEDGRIDPROCESSING
- SSPROCROWLIMIT

Logging and Error Handling Configuration Settings

- ENABLERTSVLOGGING
- GRIDEXPANSIONMESSAGES

Miscellaneous Configuration Settings

- AUTOMERGE
- AUTOMERGEMAXSLICENUMBER
- MAXNUMBEROFACTIVEDB
- NUMBLOCKSTOEEXTEND
- RESTRUCTURETHREADS
- TARGETTIMESERIESOPT

Partitioning Configuration Settings

- MAX_REQUEST_GRID_SIZE
- MAX_RESPONSE_GRID_SIZE

Ports and Connections Configuration Settings

- AGENTTHREADS
- MAXLOGINS
- SERVERTHREADS
Query Management Configuration Settings

- GRIDEXPANSION
- GRIDEXPANSIONMESSAGES
- GRIDSUPPRESSINVALID
- QRYGOVEXECBLK
- QRYGOVEXECTIME
- SSANCESTORONTOP
- SSOPTIMIZEDGRIDPROCESSING
- SSPROCROWLIMIT
- SUPNA
- TARGETASOOPT

See also Query Logging Configuration, which you can enable by means of a separate configuration file.

Aggregate Storage and Block Storage Settings Comparison

Some configuration settings apply only to block storage databases, some apply only to aggregate storage databases, and some are applicable to both.

Topics:
- Block Storage and Aggregate Storage Configuration Settings
- Aggregate Storage Configuration Settings
- Block Storage Configuration Settings

Block Storage and Aggregate Storage Configuration Settings

The following settings apply to aggregate storage databases and to block storage databases.

- AGENTTHREADS
- DLSINGLETHREADPERSTAGE
- DLTHREADSPREPARE
- GRIDEXPANSION
- GRIDEXPANSIONMESSAGES
- GRIDSUPPRESSINVALID
- MAXLOGINS
- MAXNUMBEROFACTIVEDB
- QRYGOVEXECTIME
- SERVERTHREADS
- SSOPTIMIZEDGRIDPROCESSING
Aggregate Storage Configuration Settings
The following settings apply only to aggregate storage databases.

- AUTOMERGE
- AUTOMERGEMAXSLICENUMBER
- CUSTOMCALCANDALLOCTHRUINSERT
- MAX_REQUEST_GRID_SIZE
- MAX_RESPONSE_GRID_SIZE

Block Storage Configuration Settings
The following settings apply only to block storage databases.

- CALCCACHEHIGH
- CALCCACHEDEFAULT
- CALCCACHELOW
- DLTHREADSWRITE
- DYNCALCCACHEMAXSIZE
- FORCEALLDENSECALCON2PASSACCOUNTS
- INDEXCACHESIZE
- QRYGOVEEXECBLK
- RESTRUCTURETHREADS

Config Settings List

- AGENTTHREADS
- ASODEFAULTCACHESIZE
- ASODYNAMICAGGINBSO
- ASODYNAMICAGGINBSOFOLDERPATH
- AUDITTRAIL
- AUTOMERGE
- AUTOMERGEMAXSLICENUMBER
- CALCCACHE
- CALCCACHEDEFAULT
- CALCCACHEHIGH
- CALCCACHELOW
- CALCLIMITFORMULARECURSION
• CALCTRACE
• CUSTOMCALCANDALLOCTRUINSERT
• DATACACHESIZE
• DLSINGLETHREADPERSTAGE
• DLTHREADSPREPARE
• DLTHREADSWRITE
• DYNCALCCACHEMAXSIZE
• ENABLERTSVLOGGING
• FORCEALLDENSECALCON2PASSACCOUNTS
• FORCESHUTDOWNINTERVAL
• GRIDEXPANSION
• GRIDEXPANSIONMESSAGES
• GRIDSUPPRESSINVALID
• HYBRIDBSOINCALCSCRIPT
• IGNORECONSTANTS
• INDEXCACHESIZE
• MAXFORMULACACHESIZE
• MAXLOGINS
• MAXNUMBEROFACTIVEDB
• MAX_REQUEST_GRID_SIZE
• MAX_RESPONSE_GRID_SIZE
• MDXINSERTBUFFERAGGMETHOD
• MDXINSERTREQUESTTIMEOUT
• MDXQRYGOVCOUNT
• NUMBLOCKSTOEXTEND
• QUERYRESULTLIMIT
• QRYGOVEXECBLK
• QRYGOVEXECTIME
• QUERYTRACE
• QUERYTRACETHRESHOLD
• RENEGADELOG
• RESTRUCTURETHREADS
• RTDEPCALCOPTIMIZE
• SERVERTHREADS
• SSANCESTORONTOP
• SSMEMBERIDPROCESSING
• SSOPTIMIZEDGRIDPROCESSING
AGENTTHREADS

Specifies the maximum number of threads that Essbase can spawn for operations such as logging in and out of Essbase Server and starting and stopping an application.

Syntax

AGENTTHREADS n

n—Specifies the number of threads that Essbase can spawn, where n can be 5 to 500, inclusive.

The default value is 5.

Notes

- While the actual maximum value you can set is 500, the maximum number of threads an operating system can handle might be much lower. Before specifying a value greater than the default value, check with your system administrator, as higher values can significantly consume system resources.
- If you specify a number that is less than 5, over the maximum, or a decimal value, Essbase overrides the value with a closely approximate value of its own.
- One thread is required for each initial connection to an application and database.
- The AGENTTHREADS configuration setting does not apply to Essbase Java Agent, which uses the WebLogic Server thread pool configuration for the total number of threads that can be spawned at the server and domain levels. This total thread count is limited to 500 and is specified in the config.xml file. If the value of AGENTTHREADS is less than the value of the WebLogic Server total thread count, Essbase uses the value specified in AGENTTHREADS; if the value of AGENTTHREADS is more than the WebLogic Server total thread count, Essbase uses the value specified in the config.xml file.

Example

AGENTTHREADS 100

Sets the maximum number of threads that Essbase can spawn to 100, assuming that the total thread count specified in the config.xml file is 100 or more.

See Also

SERVERTHREADS
ASODEFAULTCACHESIZE

Sets the default size for the aggregate storage cache associated with aggregate storage databases. The aggregate storage cache grows dynamically until it reaches this limit.

This setting applies only to aggregate storage databases.

Syntax

ASODEFAULTCACHESIZE [appname] n

- **appname**—Optional. Specifies the application to which the setting applies. If omitted, the setting applies to all new applications.
- **n**—An integer value indicating size in megabytes.

Description

ASODEFAULTCACHESIZE specifies, in megabytes, the size of the aggregate storage cache for aggregate storage databases.

The aggregate storage cache facilitates memory usage during data loads, aggregations, and retrievals. When an aggregate storage application is started, Essbase allocates a small area in memory as the aggregate storage cache for the application. As needed, Essbase increases the cache size incrementally until the maximum cache size specified for the application is reached or until the operating system denies additional allocations.

Example

ASODEFAULTCACHESIZE 200

Sets the aggregate storage cache size of all newly created or migrated aggregate storage databases as 200 megabytes.

ASODYNAMICAGGINBSO

Controls whether block storage databases use hybrid aggregation mode for queries. Hybrid aggregation for block storage databases means that wherever possible, block storage queries execute with efficiency similar to that of aggregate storage databases.

Hybrid aggregation is enabled by default for queries (this configuration setting is implicitly set to FULL). To enable hybrid aggregation for batch calculations, you must enable HYBRIDBSOINCALCSCRIPT in the application configuration.

This setting applies only to block storage databases.

Syntax

ASODYNAMICAGGINBSO [appname [dbname]] NONE | PARTIAL | FULL | ONLY
• **appname**—Optional. Specifies the application for which hybrid aggregation query mode is used.

If you specify a value for **appname** and do not specify a value for **dbname**, the setting applies to all databases in the specified application.

To enable the setting for a specific database, you must specify an application and database.

• **dbname**—Optional. Specifies the database, in the application specified by **appname**, for which hybrid aggregation query mode is used.

If you specify a value for **dbname** but do not specify a value for **appname**, your specification is ignored.

• **NONE**—Disable hybrid aggregation in block storage databases. This is the default.

• **PARTIAL**—Turn on hybrid aggregation only for simple outline aggregations based on the consolidation operators +, −, and −, but excluding the operators *, /, and ‰. Leave formulas to be calculated in block storage mode.

• **FULL**—Turn on hybrid aggregation for simple aggregations and formula calculations. If enabled, hybrid aggregation is in effect for member formulas using any of the supported functions. For a list of supported and unsupported functions, see [Functions Supported in Hybrid Aggregation Mode](#).

• **ONLY**—Same as FULL, but if a query is not supported in hybrid mode, return an error instead of defaulting to block storage execution. This can be useful for testing purposes while you are migrating a database from block storage execution to hybrid aggregation mode.

**Example**

```
ASODYNAMICAGGINBSO Sample PARTIAL
```

**See Also**

[HYBRIDBSOINCALCSCRIPT](#)

Introduction to Hybrid Aggregation in *Designing and Maintaining Essbase Cubes*.

Oracle Analytics Cloud - Essbase Calculation and Query Processor in *Using Oracle Analytics Cloud - Essbase*.

[ASODYNAMICAGGINBSO FOLDERPATH](#)

**Alter Application** `set cache_size` and **Query Application** `get cache_size`, for managing the size of block-storage application cache.

**ASODYNAMICAGGINBSO FOLDERPATH**

Changes the location specification for hybrid aggregation mode directories.

This setting applies only to block storage databases.

When a block storage database uses hybrid aggregation mode, the following subdirectories are created under `$ARBORPATH/hybrid/AppName`:

• **default**
These subdirectories are similar to those found in aggregate storage application directories. When the application stops, the directories are removed, and when the application restarts, they are replaced.

**Syntax**

```
ASODYNAMICAGGINBSO FOLDERPATH [appname] path_to_directory
```

- **appname**—Optional application specification.
  If you do not specify an application, the setting applies to all applications and databases on Essbase Server.

- **path_to_directory**—Path to the new directory after you have moved it.

**Example**

```
ASODYNAMICAGGINBSO FOLDERPATH Sample \machine-name\directory
```

**See Also**

ASODYNAMICAGGINBSO

### AUDITTRAIL

Use an audit trail to track changes to cube data, including Smart View updates, changes to Linked Reporting Objects (LROs), adding notes, attaching files, and referencing URLs.

This setting is applicable only for block storage cubes.

You can view the audit log in Smart View, and also on the cloud service.

**Syntax**

```
AUDITTRAIL DATA
```

**Example**

```
AUDITTRAIL DATA
```

### AUTOMERGE

Specifies whether incremental data slices are automatically merged during a data load to an aggregate storage database.

This setting applies only to aggregate storage databases.
Syntax

AUTOMERGE ALWAYS | NEVER | SELECTIVE

- **ALWAYS**—Specifies to automatically merge incremental data slices during a data load to an aggregate storage database. By default, merges are executed once for every four consecutive incremental data slices. If, however, the AUTOMERGEMAXSLICENUMBER configuration setting is used, the auto-merge process is activated when the AUTOMERGEMAXSLICENUMBER value is exceeded.

The size of the incremental data slices is not a factor in selecting which ones are merged.

The default value is ALWAYS.

- **NEVER**—Specifies to never automatically merge incremental data slices during a data load to an aggregate storage database.

To manually merge incremental data slices, use the `alter database MaxL` statement with the `merge` grammar.

- **SELECTIVE**—Specifies to activate the incremental data slice auto-merge process when the number of incremental data slices specified in the AUTOMERGEMAXSLICENUMBER configuration setting is exceeded. If the number of incremental data slices in the data load does not exceed the value of AUTOMERGEMAXSLICENUMBER, the auto-merge process is not activated.

Example

AUTOMERGE SELECTIVE

```
Specifies that the value of the AUTOMERGEMAXSLICENUMBER configuration setting determines whether the process of automatically merging incremental data slices is activated.
```

See Also

AUTOMERGEMAXSLICENUMBER

**AUTOMERGEMAXSLICENUMBER**

Specifies the maximum number of incremental data slices that can exist in a data load without activating the process of automatically merging incremental data slices. When the value of AUTOMERGEMAXSLICENUMBER is exceeded, the auto-merge process is activated.

**Note:**

To use the AUTOMERGEMAXSLICENUMBER configuration setting, the AUTOMERGE configuration setting must be set to SELECTIVE or ALWAYS.
This setting applies only to aggregate storage databases.

Syntax

AUTOMERGEMAXSLICENUMBER n

n—Specifies the maximum number of incremental data slices that can exist in a data load without activating the process of automatically merging incremental data slices.

- When the number of incremental data slices is equal to (≥) or less than (<) n, the incremental data slices are not merged.
- When the number of incremental data slices is greater than (>) n, the auto-merge process is activated.

The default value is 4.

During the auto-merge process, Essbase determines the maximum size, as a percentage, that any one incremental data slice can contribute to the maximum number of incremental input cells. Essbase counts the number of cells in all committed incremental data slices. Assume that r represents the maximum percentage. If the size of an incremental data slice, as a percentage, is:

- Equal to or less than r, the incremental data slice is added to the list of incremental data slices to be automatically merged
- Greater than r, the incremental data slice is not added to the list of incremental data slices to be automatically merged

Example

AUTOMERGEMAXSLICENUMBER 5

Activates the incremental data slice auto-merge process when the number of incremental data slices exceeds 5.

See Also

AUTOMERGE

CALCCACHE

Specifies whether Essbase uses a calculator cache when calculating the database.

This setting does not apply to aggregate storage databases.

Syntax

CALCCACHE [appname [dbname]] TRUE | FALSE

- appname—Optional. Specifies the application for which the setting applies.
  If you specify a value for appname and do not specify a value for dbname, the setting applies to all databases in the specified application.
To enable the setting for a specific database, you must specify an application and database.

If you do not specify an application, you cannot specify a database, and the setting applies to all applications and databases on Essbase Server.

- **dbname**—Optional. Specifies the database, in the application specified by **appname**, for which the setting applies.
  
  If you specify a value for **dbname** but do not specify a value for **appname**, your specification is ignored.

- **TRUE**—Essbase uses a calculator cache when calculating the database. This is the default.

- **FALSE**—Essbase does not use a calculator cache when calculating the database.

**Description**

Essbase uses the calculator cache to create and track data blocks during calculation. Using the calculator cache significantly improves calculation performance. The size of the performance improvement depends on the database configuration.

If required during a calculation, you can override this default setting using the SET CACHE command in a calculation script.

You can specify the size of the calculator cache using the SET CACHE command in a calculation script and the CALCCACHE {HIGH | DEFAULT | LOW} configuration settings.

When the CALCCACHE setting is set to TRUE, Essbase uses the calculator cache, providing that:

- The database has at least two sparse dimensions.
- You calculate at least one full sparse dimension (unless you specify the CALCCACHE ALL option in a calculation script).

**Example**

```
CALCCACHE Sample Basic FALSE
```

**See Also**

- CALCCACHEDEFAULT
- CALCCACHEHIGH
- CALCCACHELOW
- SET CACHE calculation command

**CALCCACHEDEFAULT**

Sets a default value for the calculation script SET CACHE command.

This setting does not apply to aggregate storage databases.
Syntax

CALCCACHEDEFAULT [appname [dbname]] n

- **appname**—Optional. Specifies the application for which the setting applies. If you specify a value for `appname` and do not specify a value for `dbname`, the setting applies to all databases in the specified application. To enable the setting for a specific database, you must specify an application and database. If you do not specify an application, you cannot specify a database, and the setting applies to all applications and databases on Essbase Server.

- **dbname**—Optional. Specifies the database, in the application specified by `appname`, for which the setting applies. If you specify a value for `dbname` but do not specify a value for `appname`, your specification is ignored.

- **n**—The default calculator cache size, in bytes. If you do not set a default value, it is 200,000 bytes.

Description

Essbase uses the calculator cache to create and track data blocks during calculation. Using the calculator cache significantly improves calculation performance. The size of the performance improvement depends on the database configuration.

You can specify whether Essbase uses a calculator cache by default using the CALCCACHE setting in the Essbase configuration. If required during a calculation, override this default setting using the SET CACHE command in a calculation script.

Example

Assume the Essbase configuration specifies these settings:

```
CALCCACHEHIGH 1000000
CALCCACHEDEFAULT 300000
CALCCACHELOW 200000
```

Tip:

In the Essbase configuration, a parameter is not followed by a semicolon; in a calculation script, a parameter must be followed by a semicolon.

You could then use the following SET CACHE commands in a calculation script:

```
SET CACHE HIGH;
```
Sets a calculator cache of 1,000,000 bytes for the duration of the calculation script.

SET CACHE DEFAULT;

Sets a calculator cache of 300,000 bytes for the duration of the calculation script.

SET CACHE LOW;

Sets a calculator cache of 200,000 bytes for the duration of the calculation script.

See Also
CALCCACHE
CALCCACHEHIGH
CALCCACHESLOW
SET CACHE (calculation script command)

CALCCACHEHIGH

Sets the high value for the calculation script SET CACHE command.
This setting does not apply to aggregate storage databases.

Syntax

CALCCACHEHIGH [appname [dbname]] n

- **appname**—Optional. Specifies the application for which the setting applies.
  If you specify a value for *appname* and do not specify a value for *dbname*, the setting applies to all databases in the specified application.
  To enable the setting for a specific database, you must specify an application and database.
  If you do not specify an application, you cannot specify a database, and the setting applies to all applications and databases on Essbase Server.
- **dbname**—Optional. Specifies the database, in the application specified by *appname*, for which the setting applies.
  If you specify a value for *dbname* but do not specify a value for *appname*, your specification is ignored.
- **n**—The maximum calculator cache size, in bytes (not to exceed 200,000,000 bytes).
**Description**

Essbase uses the calculator cache to create and track data blocks during calculation. Using the calculator cache significantly improves calculation performance. The size of the performance improvement depends on the database configuration.

You can specify whether Essbase uses a calculator cache by default using the `CALCCACHE TRUE | FALSE` command in the Essbase configuration. If required during a calculation, override this default setting using the `SET CACHE` command in a calculation script.

**Example**

Assume the Essbase configuration contains these settings:

```
CALCCACHEHIGH  1000000
CALCCACHEDEFAULT  300000
CALCCACHELOW  200000
```

**Note:**

In the Essbase configuration, a parameter is not followed by a semicolon; in a calculation script, a parameter must be followed by a semicolon.

You could use the following `SET CACHE` calculator commands in a calculation script:

```
SET CACHE HIGH;
```

Sets a calculator cache of 1,000,000 bytes for the duration of the calculation script.

```
SET CACHE DEFAULT;
```

Sets a calculator cache of 300,000 bytes for the duration of the calculation script.

```
SET CACHE LOW;
```

Sets a calculator cache of 200,000 bytes for the duration of the calculation script.

**See Also**

- CALCCACHE
- CALCCACHEDEFAULT
- CALCCACHELOW
**SET CACHE** (calculation script command)

**CALCCACHELOW**

Sets the low value for the calculation script **SET CACHE** command.

This setting does not apply to aggregate storage databases.

**Syntax**

CALCCACHELOW [appname [dbname]] n

- **appname**—Optional. Specifies the application for which the setting applies.
  
  If you specify a value for **appname** and do not specify a value for **dbname**, the setting applies to all databases in the specified application.

  To enable the setting for a specific database, you must specify an application and database.

  If you do not specify an application, you cannot specify a database, and the setting applies to all applications and databases on Essbase Server.

- **dbname**—Optional. Specifies the database, in the application specified by **appname**, for which the setting applies.

  If you specify a value for **dbname** but do not specify a value for **appname**, your specification is ignored.

- **n**—The minimum calculator cache size, in bytes.

**Description**

Essbase uses the calculator cache to create and track data blocks during calculation. Using the calculator cache significantly improves calculation performance. The size of the performance improvement depends on the database configuration.

You can specify whether Essbase uses a calculator cache by default using the **CALCCACHE** setting in the Essbase configuration. If required during a calculation, override this default setting using the **SET CACHE** command in a calculation script.

**Example**

Assume the Essbase configuration specifies these settings:

```
CALCCACHEHIGH  1000000
CALCCACHEDEFAULT  300000
CALCCACHELOW  200000
```

**Note:**

In the Essbase configuration, a parameter is not followed by a semicolon; in a calculation script, a parameter must be followed by a semicolon.
You could then use the following `SET CACHE` commands in a calculation script:

```
SET CACHE HIGH;
```

Sets a calculator cache of 1,000,000 bytes for the duration of the calculation script.

```
SET CACHE DEFAULT;
```

Sets a calculator cache of 300,000 bytes for the duration of the calculation script.

```
SET CACHE LOW;
```

Sets a calculator cache of 200,000 bytes for the duration of the calculation script.

See Also

CALCCACHE
CALCCACHEDEFAULT
CALCCACHEHIGH
SET CACHE (calculation script command)

**CALCLIMITFORMULARECURSION**

When set to true, prevents the server from going beyond 128 formula execution levels.

**Syntax**

```
CALCLIMITFORMULARECURSION TRUE | FALSE
```

- **TRUE**—Imposes a limit of 128 on the number of formula execution levels. This is the default.
- **FALSE**—Imposes no limit on the number of formula execution levels.

**Description**

`CALCLIMITFORMULARECURSION` limits the number of execution levels of Essbase formulas. If a calculation involves formulas referencing one or more members from sparse dimensions and there are formulas along dense dimension members, the formula execution may be recursive (have multiple execution levels). Formulas with excessive execution levels may crash the server. Setting `CALCLIMITFORMULARECURSION` to **TRUE** prevents excessive execution levels from crashing the Essbase Server.

If a formula reaches 128 execution levels and `CALCLIMITFORMULARECURSION` is set to **TRUE** (or default), Essbase stops processing that formula and writes error
messages in the application log. If a formula reaches 128 execution levels and CALCLIMITFORMULARECURSION is set to FALSE, Essbase continues processing that formula and writes an information message in the application log.

Note:
This setting does not affect formulas in MDX queries (for example, calculated members).

Example

Payroll / @SUMRANGE(Payroll, @IRDESCENDANTS(Market))

If you added a member named Payroll2 to the Measure dimension in Sample.Basic and used the following formula to calculate it, you would get a recursion error if Market has more than 128 members:

CALCTRACE

This application configuration parameter enables calculation tracing to help debug calculation scripts.

The tracing is done on the cell specified by using the SET TRACE calculation command, or by selecting the cell in Smart View. The output is available in Smart View, as well as in a file, calc_trace.txt, located in the database files directory on the cloud service.

Syntax

CALCTRACE OFF | ON

• OFF – Calculations are not traced. Any SET TRACE commands in calculation scripts are ignored.
• ON – Calculations can be traced. You can specify a cell to be traced in Smart View by selecting a cell in the grid before executing a calculation script. You can also use SET TRACE commands in calculation scripts if you need to trace multiple cells.

Example

CALCTRACE ON

See Also

SET TRACE

Tracing Calculations in Using Oracle Analytics Cloud - Essbase.
CUSTOMCALCANDALLOCTHRUINSERT

Enables execution of aggregate storage custom calculations and allocations through MDX Insert.

**Syntax**

CUSTOMCALCANDALLOCTHRUINSERT [appname [dbname]] TRUE | FALSE

- **appname**—Optional. Specifies the aggregate storage application to which the configuration applies.
- **dbname**—Optional. Specifies the aggregate storage cube to which the configuration applies.
- **TRUE**—The execution of aggregate storage custom calculations and allocations goes through MDX Insert.
- **FALSE**—Custom calculations and allocations do not execute through MDX Insert. This is the default.

**See Also**

- Performing Custom Calculations and Allocations on Aggregate Storage Databases
- USE_MDX_INSERT
- MDX Insert Specification

DATACACHESIZE

Defines the value for the data cache size for Essbase databases. The data cache is a buffer in memory that holds data blocks. Essbase allocates this memory during data load, calculation, and retrieval operations, as needed.

This setting does not apply to aggregate storage databases.

**Syntax**

DATACACHESIZE n

**n**—An integer value expressed in bytes (B), kilobytes (K), megabytes (M), or gigabytes (G):

- Minimum value: 3 megabytes (3 M)
- Maximum value: 2 gigabytes (2 G)
- Default value: 3 megabytes (3 M)

If a value is given without a B, K, M, or G qualifier, it is assumed the value is in bytes.

The qualifier can be in upper or lowercase and can be entered adjacent to the value (10M) or separated by a space (10 M).
Description

DATACACHESIZE specifies, in bytes, kilobytes, megabytes, or gigabytes, the size of the data cache for databases.

Example

```
DATACACHESIZE 90M
```

Sets the data cache size of databases to 90 megabytes.

DLSINGLETHREADPERSTAGE

Instructs Essbase to load data using a single thread per processing stage, or to use the thread values specified in the DLTHREADSPREPARE and DLTHREADSWRITE configuration settings. By working with these three configuration settings, you may be able to test and improve data load performance.

You can specify this setting for individual databases, for all databases within an application, or for all applications and databases on the server.

Syntax

```
DLSINGLETHREADPERSTAGE [appname [dbname]] TRUE | FALSE
```

- **appname**—Application name. Optional parameter for applying the TRUE or FALSE setting to one or all databases within the application. If you specify a value for **appname** and do not specify a value for **dbname**, the setting applies to all databases in the specified application. If you do not specify an application, you cannot specify a database and the setting applies to all applications and databases on the Essbase Server.

- **dbname**—Database name. Optional parameter for applying the TRUE or FALSE setting to a specific database within the specified application. If you do not specify a value for **dbname**, the setting applies to all databases within the specified application. If **appname** is not specified, you cannot specify **dbname**.

- **TRUE**—Tells Essbase not to use the values in the DLTHREADSPREPARE and DLTHREADSWRITE configuration settings when it performs a data load. Consequently, it performs all data load processes in single-thread stages.

- **FALSE**—Tells Essbase to use the thread values specified in the configuration settings DLTHREADSPREPARE and DLTHREADSWRITE as the numbers of threads to use in the preparation and write stages of data load processing. The default value is FALSE.

Description

This setting, and related settings DLTHREADSPREPARE and DLTHREADSWRITE, are related to parallel data load processing. Data load processing is divided up into stages that are performed by Essbase using separate processing threads for each stage. By default, a single thread is used for each stage. Taking advantage of the multithreading capabilities of the server machine, the separate single-thread stages can be performed in parallel.
To improve data load performance by maximizing use of processor resource for your situation, you can use these settings to enable additional multiple-thread processing within the preparation and write stages of data load processing. For more information about parallel thread processing in data loads, see Optimizing Data Loads in Designing and Maintaining Essbase Cubes.

Notes

- While testing thread values for the DLTHREADSPREPARE and DLTHREADSWRITE configuration settings, you can use the DLSINGLETHREADPERSTAGE setting to quickly revert to using a single thread per stage.
- Enabling use of multiple threads during the preparation and write stages may produce little if any benefit on a single-processor machine.
- Optimizing factors such as the content and organization of the data source can enhance performance more than increasing the numbers of threads to be used. See Optimizing Data Loads in Designing and Maintaining Essbase Cubes.

Examples

Example 1

DLSINGLETHREADPERSTAGE Sample Basic TRUE
DLTHREADSPREPARE Sample Basic 3
DLTHREADSWRITE Sample Basic 4

Essbase ignores any values specified by DLTHREADSPREPARE and DLTHREADSWRITE while loading data to the Sample Basic application and database. As a result, Essbase uses single threads in each stage.

Example 2

DLSINGLETHREADPERSTAGE FALSE
DLTHREADSPREPARE Sample Basic 3
DLTHREADSWRITE Sample Basic 4

Based on the first setting, Essbase uses the number of threads specified by the DLTHREADSPREPARE and DLTHREADSWRITE configuration settings for all databases on the server. The settings on the second and third lines specify use of 3 processing threads for the preparation stages and 4 processing threads for the write stages when loading the Sample Basic application and database. Assuming that there are no further related settings, the default value 1 (one) is assumed for all other applications and databases on the server.
Example 3

DLSINGLETHREADPERSTAGE Sample FALSE
DLTHREADSWRITE Sample Basic 3
DLTHREADSWRITE Sample Interntl 4

In this example Essbase uses the number of threads specified by the DLTHREADSPREPARE and DLTHREADSWRITE configuration settings for all databases within the application named Sample. To enable usage of different numbers of threads for the write stage for the two different databases, two DLTHREADSWRITE settings are included with different thread values for each specific database. Because no DLTHREADSPREPARE setting is specified, the preparation stage is single-threaded.

DLTHREADSPREPARE

Specifies how many threads Essbase may use during the data load preparation stage, which organizes the source data in memory in preparation for storing the data into blocks. Multiple threads, processing in parallel, may improve data load performance.

You can specify this setting for individual databases, for all databases within an application, or for all applications and databases on the server.

In order for Essbase to use the value specified for this setting, the DLSINGLETHREADPERSTAGE setting must be set to FALSE.

Syntax

DLTHREADSPREPARE [appname [dbname]] n

- **appname**—Application name. Optional parameter for using the specified number of threads in one or all databases within the application. If you specify a value for appname and do not specify a value for dbname, the setting applies to all databases in the specified application. If you do not specify an application, you cannot specify a database and the setting applies to all applications and databases on the Essbase Server.

- **dbname**—Database name. Optional parameter for using the specified number of threads when loading the specified database within the specified application. If you do not specify a value for dbname, the setting applies to all databases within the specified application. If appname is not specified, you cannot specify dbname.

- **n**—The number of threads the data load process may use for preparing the data to be loaded. Specify an integer between 1 and 32. The default value is 1.
  
  If n is greater than the maximum or a negative number, the value is assumed to be 32.

Description

This setting, and related settings DLTHREADSWRITE and DLSINGLETHREADPERSTAGE, are related to parallel data load processing. The
concept of a pipeline is relevant to Essbase data loads. A pipeline is a series of data processing elements in memory that may be executed serially or in parallel. An Essbase data load operation uses a pipeline consisting of 5 stages. By default, a single thread is used for each stage. Therefore, all data load operations need a minimum of 5 threads.

To improve data load performance by maximizing use of processor resource for your situation, you can use these settings to enable additional multiple-thread processing within the preparation and write stages of data load processing. For more information about parallel thread processing in data loads, see Optimizing Data Loads in Designing and Maintaining Essbase Cubes.

Notes

- You can use another configuration setting, DLTHREADSWRITE, to specify the number of threads for the write stage of data load processing.
- Many factors affect the possible optimal values for DLTHREADSPREPARE including the number of processors on the machine and the number of other processes running on the machine. If you want to set this setting to a value higher than the default (1), check with your system administrator, as higher values can consume considerable system resources. As a rule of thumb, do not expect performance advantages if the number of threads for this setting is greater than the number of processors on the server machine.
- Setting the value for DLTHREADSPREPARE to be greater than 1 (one) may produce little if any benefit on a single-processor machine.

Example

DLSINGLETHREADPERSTAGE Sample Basic FALSE
DLTHREADSPREPARE Sample Basic 3

Because DLSINGLETHREADPERSTAGE is set to FALSE for the Sample Basic application and database, Essbase uses 3 parallel threads during the preparation stage when loading data to Sample Basic.

See Also

DLTHREADSWRITE
DLSINGLETHREADPERSTAGE

DLTHREADSWRITE

Specifies how many threads Essbase may use during the stage of the data load process that writes blocks on the disk. Multiple threads, processing in parallel, may improve data load performance.

Since Essbase uses a single thread during the write stage of the aggregate storage data load process, this setting does not apply to aggregate storage databases.
Syntax

`DLTHREADSWRITE [appname [dbname]] n`

- **appname**—Application name. Optional parameter for using the specified number of threads in one or all databases within the application. If you specify a value for `appname` and do not specify a value for `dbname`, the setting applies to all databases in the specified application. If you do not specify an application, you cannot specify a database and the setting applies to all applications and databases on the Essbase Server.

- **dbname**—Database name. Optional parameter for using the specified number of threads when loading the specified database within the specified application. If you do not specify a value for `dbname`, the setting applies to all databases within the specified application. If `appname` is not specified, you cannot specify `dbname`.

- **n**—The number of threads the data load process may use for writing data blocks to the disk. Specify an integer between 1 and 32. The default value is 1. If `n > 32`, or a negative number, the value is assumed to be 32.

Description

This setting, and related settings `DLTHREADSPREPARE` and `DLSINGLETHREADPERSTAGE`, are related to parallel data load processing. The concept of a `pipeline` is relevant to Essbase data loads. A pipeline is a series of data processing elements in memory that may be executed serially or in parallel. An Essbase data load operation uses a pipeline consisting of 5 stages. By default, a single thread is used for each stage. Therefore, all data load operations need a minimum of 5 threads.

To improve data load performance by maximizing use of processor resource for your situation, you can use these settings to enable additional multiple-thread processing within the preparation and write stages of data load processing.

You can specify `DLTHREADSWRITE` for individual databases, all databases within an application, or for all applications and databases on the server.

In order for Essbase to use the value specified for `DLTHREADSWRITE`, the configuration setting `DLSINGLETHREADPERSTAGE` must be set to FALSE.

For more information about parallel thread processing in data loads, see Optimizing Data Loads in *Designing and Maintaining Essbase Cubes*.

Notes

- You can use another configuration setting, `DLTHREADSPREPARE`, to specify the number of threads for the preparation stage of data load processing.

- Many factors affect the possible optimal values for `DLTHREADSWRITE` including the number of processors on the machine and the number of other processes running on the machine. If you want to set this setting to a value higher than the default (1), check with your system administrator, as higher values can consume considerable system resources. As a rule of thumb, do not expect performance advantages if the number of threads for this setting is greater than the number of processors on the server machine.
• Setting the value for DLTHREADSWRITE to be greater than 1 (one) may produce little if any benefit on a single-processor machine.

Example

DLSINGLETHREADPERSTAGE Sample Basic FALSE
DLTHREADSWRITE Sample Basic 3

Because DLSINGLETHREADPERSTAGE is set to FALSE for the Sample Basic application and database, Essbase uses 3 parallel threads during the write stage when loading data to Sample Basic.

See Also

DLTHREADSPREPARE
DLSINGLETHREADPERSTAGE

**DYNCALCCACHEMAXSIZE**

Specifies the maximum amount of memory allocated for the dynamic calculator cache for each database. The specified value takes effect for all databases that are opened after the server is started.

The dynamic calculator cache is a memory buffer that holds data blocks that are expanded to include dynamically calculated members. Essbase allocates memory in the dynamic calculator cache to store these blocks during retrievals or calculations that involve dynamically calculated members.

Using dynamic calculator cache may improve retrieval performance by reducing the number of calls to the operating system to do memory allocations.

This setting does not apply to aggregate storage databases.

**Syntax**

```
DYNACALCCACHEMAXSIZE [appname [dbname]] n
```

- **appname**—If you specify an application name, the setting applies to all databases within the application. If you do not specify an application name, the setting applies to all applications and databases on the server.
- **dbname**—If you specify a database name, the setting applies only to the database. If you do not also specify an application name, the setting applies to all applications and databases on the server.
- **n**—An integer expressed in bytes (B), kilobytes (K), megabytes (M), or gigabytes (G)
  - Minimum value: 0 megabytes (0 M). If the value is 0, Essbase does not use dynamic calculator cache.
  - Default value: 20 megabytes (20M, which is 20,971,520 bytes)
  - The maximum amount of memory that can be allocated is 256 GB:
– If a value is given without a B, K, M, or G qualifier, it is assumed the value is in bytes.
– The qualifier can be in upper or lowercase and can be entered adjacent to the value (10M) or separated by a space (10 M).

Example

DYNCALCCACHEMAXSIZE 30M

Sets 30 megabytes as the maximum size for the dynamic calculator cache.

ENABLERTSVLOGGING

Determines whether Essbase logs runtime substitution variables that are used in a calculation script.

Runtime substitution variable log entries are written to the application log file.

Syntax

ENABLERTSVLOGGING [appname [dbname]] TRUE | FALSE

• appname—Optional. Specifies the application for which runtime substitution variable logging is to be set.
  If you specify a value for appname and do not specify a value for dbname, the setting applies to all databases in the specified application.
  To enable the setting for a specific database, you must specify an application and database.
  If you do not specify an application, you cannot specify a database, and the setting applies to all applications and databases on Essbase Server.

• dbname—Optional. Specifies the database, in the application specified by appname, for which runtime substitution variable logging is to be set.
  If you specify a value for dbname but do not specify a value for appname, your specification is ignored.

• TRUE—Runtime substitution variables that are used in a calculation script are logged. For information about the format of these log entries, see Logging Runtime Substitution Variables in Designing and Maintaining Essbase Cubes.

• FALSE—Runtime substitution variables that are used in a calculation script are not logged. The default value is FALSE.

Example

ENABLERTSVLOGGING TRUE

See Also

SET RUNTIMESUBVARS calculation command
FORCEALLDENSECALCON2PASSACCOUNTS

Normally, a two-pass tagged member of a dense accounts dimension triggers a second calculation pass on all dense cells of the data block. The false parameter value for this setting blocks the second pass for all other than the cells for the member tagged as two-pass.

Syntax

FORCEALLDENSECALCON2PASSACCOUNTS TRUE | FALSE

- TRUE—(Default value) When a two-pass member of a dense accounts dimension is calculated, the second calculation pass calculates all dense cells of the data block.
- FALSE—In the same situation, the FALSE setting blocks the second calculation pass for all dense cells except those affiliated with the two-pass member.

Description

This setting addresses the situation where a two-pass member of a dense accounts dimension links through @XREF to a two-pass member of a dense accounts dimension in another database outline, and that two-pass member links back to the original outline. The additional calculations in the second calculation pass can result in an infinite loop. The FALSE parameter value blocks the additional calculations. If you are very cautious about data correctness, check calculation results.

Example

FORCEALLDENSECALCON2PASSACCOUNTS FALSE

FORCESHUTDOWNINTERVAL

This setting applies to block storage and aggregate storage databases.

The default interval is 10 seconds.

Syntax

FORCESHUTDOWNINTERVAL n

Example

FORCESHUTDOWNINTERVAL 2000

Checks for a heartbeat every 2000 seconds.

GRIDEXPANSION

When set to ON, improves performance when transparent partitions are queried.
Syntax

GRIDEXPANSION [appname [dbname]] ON | OFF

• appname—Optional. If you specify an application name, the setting applies to all databases within the named application. If you do not specify an application name, the setting applies to all applications and databases on the Essbase Server.

• dbname—Optional. If you specify a database name and an application name, the setting applies only to the named database. If you do not also specify an application name, the database is ignored and the setting applies to all applications and databases on the Essbase Server.

• ON—This is the default value. Enables grid expansion.

• OFF—Suppresses grid expansion.

Description

GRIDEXPANSION improves performance of some queries. If all of the following conditions are met, however, client queries may receive incorrect results (such as most data values displaying as #MISSING, whether or not cells contain data):

• The client queries the target database of a transparent partition.

• The client query requests values from a dynamically calculated block.

• Cells requested from the dynamically calculated block reference dense, dynamically calculated members.

• Dense, dynamically calculated members depend on values from one or more source databases.

See Also

GRIDEXPANSIONMESSAGES

GRIDEXPANSIONMESSAGES

Sets whether grid expansion-related messages are displayed to Smart View and other grid client users, and are written to the application log.

Syntax

GRIDEXPANSIONMESSAGES [appname [dbname]] ON | OFF

• appname—Optional. If you specify an application name, the setting applies to all databases within the named application. If you do not specify an application name, the setting applies to all applications and databases on the Essbase Server.

• dbname—Optional. If you specify a database name and an application name, the setting applies only to the named database. If you do not also specify an application name, the database is ignored and the setting applies to all applications and databases on the Essbase Server.

• ON—Allows grid-expansion-related messages.

• OFF—This is the default value. Suppresses grid-expansion-related messages.
Description

If a grid client user retrieves data from a partition, the following message may be displayed repeatedly and written to the application log:

Grid expansion enabled for this query

To prevent this message from appearing, set GRIDEXPANSIONMESSAGES to OFF.

Example

GRIDEXPANSIONMESSAGES OFF

See Also

GRIDEXPANSION

GRIDSUPPRESSINVALID

Sets whether invalid attribute combinations, which are represented on the grid by #invalid, are suppressed in Smart View. An invalid attribute combination is the result of an intersection of a dimension member for which an attribute is not assigned or, if an attribute is assigned to the member, the attribute combination is not within the scope of the grid query or the assigned attribute is incorrect. Invalid attribute combinations are suppressed when the row contains all #invalid values. Valid combinations with #MISSING values are not suppressed.

This configuration setting applies to block storage and aggregate storage databases.

Syntax

GRIDSUPPRESSINVALID [appname [dbname]] TRUE | FALSE

- **appname**—Optional. If you specify an application name, the setting applies to all databases within the named application. If you do not specify an application name, the setting applies to all applications and databases on the Essbase Server.

- **dbname**—Optional. If you specify a database name and an application name, the setting applies only to the named database. If you do not also specify an application name, the database is ignored and the setting applies to all applications and databases on the Essbase Server.

- TRUE—Enables suppressing invalid attribute combinations on the grid. This is the default value.

- FALSE—Invalid attribute combinations are not suppress on the grid.

Example

GRIDSUPPRESSINVALID Sample Basic TRUE

Suppresses #invalid values in the Sample.Basic database.
HYBRIDBSOINCALCSCRIPT

Controls whether cubes in the application use hybrid aggregation mode in calculation scripts when stored members depend on dynamic members. When set to FULL, the calculation engine uses the hybrid aggregation query engine to calculate the results, and then stores them.

Hybrid aggregation mode means that wherever possible, data calculation executes with efficiency similar to that of aggregate storage databases.

This setting is not applicable to aggregate storage databases.

If you enable this setting, do not disable ASODYNAMICAGGINBSO, which is on by default (meaning hybrid aggregation mode is enabled for queries).

Syntax

HYBRIDBSOINCALCSCRIPT [appname [dbname]] FULL|NONE

- **appname**—Optional. Specifies the application for which hybrid aggregation mode is used.
  
  If you specify a value for appname and do not specify a value for dbname, the setting applies to all databases in the specified application.
  
  To enable the setting for a specific database, you must specify an application and database.
  
- **dbname**—Optional. Specifies the database, in the application specified by appname, for which hybrid aggregation mode is used.
  
  If you specify a value for dbname but do not specify a value for appname, your specification is ignored.
  
- **FULL**—Calculation scripts run in hybrid aggregation mode.
  
- **NONE**—Calculation scripts run in block storage mode. This is the default.

Notes

The following limitations apply to hybrid aggregation mode for calculation scripts. If encountered, Essbase defaults to block storage execution for these kinds of calculation scripts.

- **CALC DIM, CALC ALL, AGG**, and any other assignment-free expressions that calculate a sub-tree, do not use hybrid mode.

  Oracle recommends limiting your use of CALC DIM and AGG to dimensions wherein no stored members are dependent on dynamic members. To calculate upper-level stored members that depend on dynamic members, use assignment formulas with calculation functions.

- **DATAEXPORT** for dynamic members does not use hybrid mode.
• Intelligent calculation does not use hybrid mode.
• Do not use CREATENONMISSINGBLOCK or CREATEBLOCKONEQ in calculation scripts you want to run in hybrid mode.
• CALCPARALLEL is not supported in hybrid mode. For parallel calculation, use FIXPARALLEL.
• See also the limitations listed in Calculation and Query Processor.

Example

HYBRIDBSOINCALCSCRIPT FULL

See Also

ASODYNAMICAGGINBSON

IGNORECONSTANTS
Controls whether #Missing values, when used as operands in formulas, should remain #Missing after the formula calculation.

Syntax

IGNORECONSTANTS TRUE | FALSE

• TRUE—Default option. #Missing values remain missing regardless of interaction with formula constants.
• FALSE— #Missing values can be changed by interaction with formula constants.

Description

If a #Missing data value is processed in a formula with a constant or other data-independent construct, the default behavior is that #Missing is not treated like a data value. For example, if A is missing, A+5 returns #Missing.

If you set IGNORECONSTANTS to FALSE, #Missing is treated like a data value. For example, if A is missing, A+5 returns 5.

Example

If the configuration is as follows:

IGNORECONSTANTS TRUE

then the result for X in the following formula is #Missing

IF (X)
5;
ELSE
3
ENDIF
INDEXCACHESIZE

Defines the value for the index cache size for Essbase databases. The index cache is a buffer in memory that holds index pages. Essbase allocates this memory at startup of the database.

The value of the index cache size can be expressed in bytes, kilobytes, megabytes, or gigabytes. Terabytes must be expressed in gigabytes.

This setting does not apply to aggregate storage databases.

Syntax

INDEXCACHESIZE n

n—An integer value expressed in bytes (B), kilobytes (K), megabytes (M), or gigabytes (G):

- Minimum value: 1 megabytes (1 M)
- Maximum value: 256 TB

Default value for buffered I/O: 1 megabyte (1 M)

If a value is given without a B, K, M, or G qualifier, it is assumed the value is in bytes.

The qualifier can be in upper or lowercase and can be entered adjacent to the value (10 M) or separated by a space (10 M).

Example

INDEXCACHESIZE 100M

Sets the index cache size of databases to 100 megabytes.

MAXFORMULACACHESIZE

Applies to aggregate storage databases, or to block storage databases when hybrid aggregation is enabled. Specifies the maximum size of the formula cache to be made available for calculating members with formulas.

Syntax

MAXFORMULACACHESIZE [appname [dbname]] n

- appname—Optional. To set the cache size maximum for a specific application, specify the application name.
- dbname—Optional. To set the cache size maximum for a specific database, specify the database name. If dbname is specified, appname must also be specified.
• \( n \)—An integer that specifies the number of kilobytes (KB) to set as the maximum cache size to be made available for calculating members with formulas. The default is 1024.

Description

If the amount of cache that Essbase sets aside for calculating members of outlines is insufficient, the following error is generated: "ERROR - 1200601 - Not enough memory for formula execution. Set MAXFORMULACACHESIZE configuration parameter to \([n]\) and try again." The error recommends a value to use with the MAXFORMULACACHESIZE setting.

Error 1200601 is likely to occur in situations where one dimension's dynamic calculations would expand beyond the current formula cache size setting.

If you see error 1200601, the following guidelines can help you determine what value to use for \( n \):

1. Identify which queried dimensions are represented by dynamic members.
2. Multiply the sizes of those dimensions to get a number of members.
3. Multiply the number of members by 8 to get the recommended \( n \) value (not more than 4G).

For example, the default formula cache size of 1024 allows 1024/8=128 members to be in the cache.

Notes

• This setting is only relevant if your query references at least one dynamic member with a formula, or if your MDX query has a calculated member in the WITH section.
• This cache is allocated per calculation thread. Concurrent MDX requests can be allocated multiple cache objects, each with a maximum size specified in MAXFORMULACACHESIZE.
• The entire specified amount is not used unless needed.
• The memory is released after the query completes.
• Oracle recommends that you use this setting only in response to error 1200601.
• Error 1200601 appears, stopping the current query, only in the case where one of the queried dimensions with dynamic members does not fit the formula cache size. Otherwise, the query runs even if the requested cache size is larger than the MAXFORMULACACHESIZE setting; however, the query is split internally into multiple calculators. The split query may run, but cause a decrease in performance. Therefore, it is optimal for you to ensure that all dimensions in a query do fit within the formula cache size.

Example

MAXFORMULACACHESIZE 2048

Sets the aggregate storage formula cache size maximum to 2048 KB for every application and database.
See Also

ASODYNAMICAGGINBSO

MAXLOGINS

Sets a limit on the number of user sessions that can be connected to the Essbase Server at any one time.

Syntax

MAXLOGINS n

n—Any integer from 1000 to 1048575 is valid. The default value is 10000.

Description

This setting limits the maximum number of user sessions allowed to connect to the Essbase Server at any one time. This number includes multiple instances of the same user.

You may wish to adjust the value of MAXLOGINS to match computer resources, or to more closely manage concurrent ports and user sessions. A concurrent port is used for each unique combination of client machine, Essbase Server and login name. For example, the same user with five open Excel worksheets connected to the same Essbase Server use one port, but five sessions.

Notes

• Increasing the value of MAXLOGINS increases memory use approximately 6 bytes per user session.
• If the setting is less than the minimum value, 1000, the value is assumed to be 1000.

Example

MAXLOGINS 50000

Increases the maximum number of simultaneous logins possible, from the default of 10000 to 50000.

See Also

SERVERTHREADS

MAXNUMBEROFACTIVEDB

Specifies the maximum number of active databases that can be accessed concurrently. If the maximum number of active databases is exceeded, the database does not start.
Syntax

MAXNUMBEROFACTIVEDB \( n \)

\( n \)—Specifies the maximum number of databases that can be accessed concurrently. A value of 0 means that there is no maximum limit. The default value is 0.

Example

MAXNUMBEROFACTIVEDB 10

Specifies that 10 databases can be active.

MAX_REQUEST_GRID_SIZE

Specifies the maximum size of the request grid. The request grid is the number of cells requested from the target (an aggregate storage database) and sent to the data source. Limiting the size of the request grid, which can be millions of cells, ensures a reasonable response time.

If you find that you must set a small request grid size, you should look into improving the design of the application.

Syntax

MAX_REQUEST_GRID_SIZE [\( appname \) [\( dbname \)]] \( n \)

- \( appname \)—Optional. Specifies the application for which the request grid size is to be set.
  
  If you specify a value for \( appname \) and do not specify a value for \( dbname \), the setting applies to all databases in the specified application.
  
  To enable the setting for a specific database, you must specify an application and database.
  
  If you do not specify an application, you cannot specify a database, and the setting applies to all applications and databases on Essbase Server.

- \( dbname \)—Optional. Specifies the database, in the application specified by \( appname \), for which the request grid size is to be set.
  
  If you specify a value for \( dbname \) but do not specify a value for \( appname \), your specification is ignored, and logging diagnostic messages is enabled for all applications and databases on Essbase Server.

- \( n \)—Specifies the size of the request grid to be returned from the data source.
  
  The default value is 10 million (10000000) cells.
  
  The maximum value is limited by the unsigned int value of 4294967295.

You must restart Essbase Server to initialize any change to the configuration file.
Example

MAX_REQUEST_GRID_SIZE ASOSamp 5000000

Limits the request grid to 5 million cells for all databases associated with the ASOSamp application.

See Also

MAX_RESPONSE_GRID_SIZE configuration setting

MAX_RESPONSE_GRID_SIZE

Specifies the maximum size of the response grid. The response grid is the number of cells that the target (an aggregate storage database) sends to the source.

The amount of memory required to temporarily hold the response grid in the data target is proportional to the size of the request grid (MAX_REQUEST_GRID_SIZE). In the case of a huge request grid with millions of cells, the amount of memory required for the response grid to be sent in one operation could pose problems (for example, the system could reach memory boundaries or fail to allocate enough memory). With the MAX_RESPONSE_GRID_SIZE configuration setting, Essbase splits the request grid into slices of data and sends multiple, smaller response grids to the source.

Syntax

MAX_RESPONSE_GRID_SIZE [appname [dbname]] n

• appname—Optional. Specifies the application for which the response grid size is to be set.
  If you specify a value for appname and do not specify a value for dbname, the setting applies to all databases in the specified application.
  To enable the setting for a specific database, you must specify an application and database.
  If you do not specify an application, you cannot specify a database, and the setting applies to all applications and databases on Essbase Server.

• dbname—Optional. Specifies the database, in the application specified by appname, for which the response grid size is to be set.
  If you specify a value for dbname but do not specify a value for appname, your specification is ignored, and logging diagnostic messages is enabled for all applications and databases on Essbase Server.

• n—Specifies the size of the slice of the response grid to be sent to the data target.
  The default value is one million (1000000) cells, which requires 8 MB of memory.
  For example, if MAX_REQUEST_GRID_SIZE is set to one billion (1000000000) cells and MAX_RESPONSE_GRID_SIZE is set to one million (1000000) cells, the size of the response grid is one thousand (1000) cells.

You must restart Essbase Server to initialize any change to the configuration file.
Example

MAX_RESPONSE_GRID_SIZE ASOSamp 500000

Limits the response grid to a half-million cells (which requires 4 MB of memory) for all databases associated with the ASOSamp application.

See Also

MAX_REQUEST_GRID_SIZE configuration setting

MDXINSERTBUFFERAGGMETHOD

Defines how the output buffer should be created for an MDX Insert request on the database.

Syntax

MDXINSERTBUFFERAGGMETHOD ADD | LAST

- LAST—This is the default behavior, if MDXINSERTBUFFERAGGMETHOD is unset. If, during an Insert operation, a value needs to be written to an output buffer location that already contains a value, the latest value overwrites the older value.
- ADD—If, during an Insert operation, a value needs to be written to an output buffer location that already contains a value, the latest value is summed with the older value.

Description

During execution of the MDX Insert query, an output buffer is created in memory which accumulates with values until the query is completed. This setting enables you to define the method with which values are aggregated in the output buffer. The method that you use can have an effect on the data results of the MDX Insert operation.

Assume that in an MDX Insert query, two source tuples are mapped to a single target tuple, as shown:

```
INSERT
([Payroll], [Jan]) TO ([Revised Payroll], [Jan])
([Payroll], [Feb]) TO ([Revised Payroll], [Jan])
...
```

Assume that the value of ([Payroll], [Jan]) is 100, and the value of ([Payroll], [Feb]) is 200.

Using the default buffer aggregation behavior (LAST),

```
MDXINSERTBUFFERAGGMETHOD LAST
```
1. The value for ([Payroll], [Jan]) is written to the output buffer for ([Revised Payroll], [Jan]), making its value 100.

2. The value for ([Payroll], [Feb]) is written to the same output buffer for ([Revised Payroll], [Jan]), overwriting the previous value, and changing it to 200.

If you change the buffer aggregation behavior to ADD,

MDXINSERTBUFFERAGGMETHOD ADD

1. The value for ([Payroll], [Jan]) is written to the output buffer for ([Revised Payroll], [Jan]), making its value 100.

2. The value for ([Payroll], [Feb]) is added to the same output buffer for ([Revised Payroll], [Jan]), increasing its value to 300.

See Also

MDX Insert Specification

**MDXINSERTREQUESTTIMEOUT**

Sets the number of seconds after which Essbase times out an MDX Insert request on the database.

**Syntax**

MDXINSERTREQUESTTIMEOUT n

Where n is the number of seconds the MDX Insert request is permitted to run before timing out. The default is -1, meaning there is no timeout.

MDXINSERTREQUESTTIMEOUT 240

Sets the timeout for MDX Insert requests at four minutes.

See Also

MDX Insert Specification

**MDXQRYGOVCOUNT**

Initializes a counter (number of check conditions) to control how often Essbase checks for conditions that would warrant termination of an MDX query. Using this counter can reduce or increase the default number of checks (1000); reducing the number of checks (by setting n higher) improves performance. The counter starts at n and decrements until the counter reaches zero: at that time Essbase performs a check.

**Syntax**

MDXQRYGOVCOUNT [appname [dbname]] n
- **appname**—Optional. Specifies the application for which to apply the checking counter. If you specify a value for `appname` and do not specify a value for `dbname`, the setting applies to all databases in the specified application. To enable the setting for a specific database, you must specify an application and database.

- **dbname**—Optional. Specifies the database, in the application specified by `appname`, for which to apply the checking counter. If you specify a value for `dbname` but do not specify a value for `appname`, your specification is ignored.

- **n**—Integer specifying the counter (number of check conditions) that Essbase checks for conditions that warrant query termination. You must specify this parameter or Essbase ignores this setting. If do not specify `appname` or `dbname`, the counter applies to the entire server. The default value is 1000. The minimum value is 100, and the maximum value is 5000.

**Note:**
You can use the **Esc** key to cancel any query running from MaxL Shell.

**Example**

`MDXQRYGOVCOUNT 1500`

**See Also**

- `QRYGOVEXECTIME`
- `QRYGOVEXECBLK`

### NUMBLOCKSTOEXTEND

Determines the number of bytes by which data files in block storage databases are extended to accommodate block updates that require additional disk space.

**Syntax**

```
NUMBLOCKSTOEXTEND [appname [dbname]] n
```

The product of `n` and the currently requested block size is the number of bytes by which the data file is extended.

The default value is 2,048.

**Description**

When the Essbase block storage kernel updates a block, it writes to a new disk location. The block storage kernel searches free space to find a new disk location to use. If there is not enough free space to service the current request, the data file is extended.
Note:

Upon first upgrading to this release, there is an increase in the amount of disk space pre-allocated for page files unless you set NUMBLOCKSTOEXTEND to 1.

Example

NUMBLOCKSTOEXTEND Sample Basic 2240

QUERYRESULTLIMIT

Sets the maximum number of cells returned by an MDX or grid client query. Applies to block storage, aggregate storage and hybrid aggregation databases.

Syntax

QUERYRESULTLIMIT [appname [dbname]] n

appname—Optional. Applies the query result limit to the application specified. If you specify appname, you must also specify a value for n, or Essbase ignores QUERYRESULTLIMIT. If you do not specify an application, you cannot specify a database, and the query result limit applies to all applications and databases on the server. If you specify a value for appname and do not specify a value for dbname, the query time limit applies to all databases in the specified application.

dbnname—Optional. Must be used with appname and n, or the server ignores QUERYRESULTLIMIT. If you specify dbname, appname, and n, the query result limit is applied only to the specified database.

n—An integer value between 0 and $2^{31}$ specifies the number of query result cells that the server allows a query to return.

The default value is 1000000 (1M).

Description

QUERYRESULTLIMIT specifies the maximum number of result cells that an MDX query or grid client query can retrieve before Essbase terminates the query and returns an error message.

Use this setting to limit the result volume of queries, and prevent a query from freezing when a very large number of result cells are returned.

Examples

QUERYRESULTLIMIT Sample Basic 100000
Sets 100,000 cells as the maximum number of results cells returned in a query to the Basic database for the Sample application.

QUERYRESULTLIMIT 150000

Sets 150,000 cells as the maximum number of cells that a query can return before being terminated. The query result limit applies to all applications and databases on the Essbase Server that corresponds to this configuration.

QRYGOVEXECBLK

Sets the maximum number of blocks that a query can access before the query is terminated.

This setting does not apply to aggregate storage databases.

Syntax

QRYGOVEXECBLK [appname [dbname]] n

- **appname**—Optional. Applies the query block limit to the application specified. If you specify *appname*, you must also specify a value for *n*, or Essbase Server ignores QRYGOVEXECBLK. If you do not specify an application, you cannot specify a database, and the query block limit applies to all applications and databases on the server. If you specify a value for *appname* and do not specify a value for *dbname*, the query time limit applies to all databases in the specified application.

- **dbname**—Optional. Must be used with *appname* and *n*, or Essbase Server ignores QRYGOVEXECBLK. If you specify *dbname*, *appname*, and *n*, the query block limit is applied only to the specified database.

- **n**—The value of *n* specifies the number of blocks that Essbase Server allows a query to access before the query is terminated. You must specify this parameter or the server ignores QRYGOVEXECBLK. If you do not specify *appname* or *dbname*, the query block limit applies to the entire server.

Description

QRYGOVEXECBLK specifies the maximum number of blocks that a query can retrieve before Essbase Server terminates that query (a request for information sent to a database). You can apply this setting to an entire server, to all the databases in a single application, or to a single database.

When a query exceeds the block limit and is terminated, an error message is written to the application log of the application accessed for the query.

Restarting Essbase Server after adding or changing this setting activates the new setting values.

Use QRYGOVEXECBLK to prevent these types of queries:
A long-running query against a database that accesses attributes at a high level, forcing many dynamic calculations to occur.

A query that uses the zoom-in "Drill to bottom" option in a large dimension.

A query that uses the zoom-in "Drill to all levels" option in a large dimension.

Use QRYGOVEXECBLK, for example, if you have users who try to retrieve so much data in a single query that their query appears to hang for minutes at a time. A query launched against the database involving attribute dimensions, for example, may be larger than the user realizes.

Notes

If you use an invalid value (such as a negative number, a letter, a word, or a special character) for \( n \), Essbase Server ignores QRYGOVEXECBLK.

Query governor settings are ignored during data load and calculation. You can leave query governor settings in the configuration file whether you are performing these operations or querying against the data.

Example

```
QRYGOVEXECBLK Sample Basic 3
```

Sets three blocks as the maximum number of blocks that a query to Sample Basic can access before being terminated. A block is created for each unique combination of sparse dimension members. If a user issues a query that accesses four unique combinations of sparse dimensions, Essbase Server terminates the query and writes a message to the application log.

```
QRYGOVEXECBLK 5
```

Sets five blocks as the maximum number of blocks that a query can access before being terminated. The query time limit applies to all applications and databases on Essbase Server.

See Also

QRYGOVEXECTIME

### QRYGOVEXECTIME

Sets the maximum amount of time a query can use to retrieve and deliver information before the query is terminated.

**Syntax**

```
QRYGOVEXECTIME [appname [dbname]] n
```
- **appname**—Optional. Applies the query time limit to the application specified. If you specify `appname`, you must also specify a value for `n`, or Essbase Server ignores QRYGOVEXECTIME. If you do not specify an application, then you cannot specify a database, and the query time limit applies to all applications and databases on Essbase Server. If you specify a value for `appname` and do not specify a value for `dbname`, the query time limit applies to all databases in the specified application.

- **dbname**—Optional. Must be used with `appname` and `n`, or Essbase Server ignores QRYGOVEXECTIME. If you specify `dbname`, `appname`, and `n`, the query time limit is applied only to the specified database.

- **n**—Integer specifying the number of seconds that Essbase Server allows a query to run before the query is terminated. The default value is 300 seconds. If you do not specify `appname` or `dbname`, the query time limit applies to the entire server.

**Description**

QRYGOVEXECTIME specifies the maximum amount of time that a query can run before Essbase Server terminates the query (a request for information sent to a database). You can apply this setting to an entire server, to all the databases in a single application, or to a single database.

When a query exceeds the time limit and is terminated, an error message is written to the application log of the application accessed for the query.

Restarting Essbase Server after adding or changing this setting activates the new setting values.

Use QRYGOVEXECTIME to prevent these types of queries:

- A long-running query against a database that accesses attributes at a high level, forcing many dynamic calculations to occur.
- A query that uses the “Drill to bottom” option in a large dimension.
- A query that uses the “Drill to all levels” option in a large dimension.

Use QRYGOVEXECTIME, for example, if you have users who try to retrieve so much data in a single query that their query appears to hang for minutes at a time.

**Notes**

- Because the query time setting is evaluated in 10 second increments, the query may actually run nine seconds longer than specified before being terminated.
- If you use an invalid value (such as a negative number, a letter, a word, or a special character) for `n`, the server ignores QRYGOVEXECTIME.
- Query governor settings are ignored during data load and calculation. You can leave query governor settings in the configuration file whether you are performing these operations or querying against the data.

**Example**

QRYGOVEXECTIME Sample Basic 20
QRYGOVEXECTIME 45

Sets 45 seconds as the maximum time that a query can run before being terminated. The query time limit applies to all applications and databases on the server.

See Also
QRYGOVEXECBLK

QUERYTRACE

Sets a query calculation flow trace to be run and the results to be printed out to a file.

Description
This setting enables query tracing for calculation flows. The query tracing output file includes:

• The input query
• An expanded query odometer
• General information about query calculation units
• A list of formulas and aggregations
• An ordered list of all output cells that are calculated or aggregated during the query, according to solve order

Notes

• This setting applies to block storage and hybrid aggregation databases.
• The query tracing output file, query_trace.txt, is written to the database files location.

Syntax

QUERYTRACE n

Where n should be set to -1, to enable query tracing.

Example

QUERYTRACE -1

Sets a tracing query to be run that includes all tracing features listed in Description.
See Also

QUERYTRACETHRESHOLD

QUERYTRACETHRESHOLD

Sets the maximum number of cells than can be displayed for each formula or aggregation number for query tracing for calculation order analysis.

Description

This setting specifies the maximum number of cells (or tuples) that a QUERYTRACE query will display for an MDX query or grid client query. Use this setting to limit the number of cells printed for each formula or aggregation number.

Notes

This setting applies to block storage and hybrid aggregation databases.

Syntax

QUERYTRACETHRESHOLD n

Where n is an integer value between 0 and unlimited, specifying the maximum number of cells to display for each formula calculation path. The default value is 100.

Example

QUERYTRACETHRESHOLD 50

Sets 50 as the maximum number of cells to be displayed for a query tracing the solve order or calculation order.

See Also

QUERYTRACE

RENEGADELOG

Enables logging of members loaded into a renegade member intersection.

By default, Essbase does not create a log file to track data loaded to renegade members. If RENEGADELOG is set to true, Essbase creates a log file in the Essbase logs directory. The log file name is renDataload_filenamefilename_timestamp.log for non-SQL data loads and renDataload_SQL_timestamp.log for SQL-based data loads.

The log file records the data value loaded to the renegade member. If more than one member in a given data load is missing for a dimension with renegade members enabled, the log file lists only one value. Information on the remaining missing data values is provided in comments.
Syntax

RENEGADELOG [appname [dbname]] TRUE | FALSE

- **appname**—Application name. Optional parameter for applying the TRUE or FALSE setting to one or all databases within the application. If you specify a value for `appname` and do not specify a value for `dbname`, the setting applies to all databases in the specified application. If you do not specify an application, you cannot specify a database, and the setting applies to all applications and databases on the Essbase Server.

- **dbname**—Database name. Optional parameter for applying the TRUE or FALSE setting to the specified database within the specified application. If you do not specify a value for `dbname`, the setting applies to all databases within the specified application. If `appname` is not specified, you cannot specify `dbname`.

- **TRUE**—Creates a log file to track data loaded to renegade members.

- **FALSE**—No log file is created. This is the default value.

Example

RENEGADELOG TRUE

RESTRUCTURETHREADS

Specifies whether parallel restructuring is enabled for a database and the number of threads to use.

This setting does not apply to aggregate storage databases.

Syntax

RESTRUCTURETHREADS [ appname [ dbname] ] n

- **appname**—Application name. Optional parameter for enabling parallel restructuring for one or all databases in an application.

- **dbname**—Database name. Optional parameter for enabling parallel restructuring for an individual database. This parameter must be used in combination with `appname`.

- **n**—Number of threads to use in parallel restructuring.

Notes

- Use the value `xxxxx` to indicate "all" for any application or database argument. For example:
  
  RESTRUCTURETHREADS `xxxxx Basic 2`

  enables parallel restructuring for any application with a Basic database.

- Settings for nonexistent applications or databases are ignored.

- If RESTRUCTURETHREADS is not defined, the default is one thread.
• Oracle recommends setting an application’s RESTRUCTURETHREADS to 2 for most systems, or 4 if the application runs on Exalytics. Check your calculation and restructure performance after making any changes.

Description
This setting specifies whether parallel restructuring is enabled for a database and the number of threads to use. You can enable parallel restructuring for individual databases, or for all databases in an application. For more information about parallel restructuring, see Parallel Restructuring in *Designing and Maintaining Essbase Cubes*.

Examples

RESTRUCTURETHREADS Sample 2

| Specifies two threads and applies to all databases in the Sample application |

RESTRUCTURETHREADS Sample Basic 4

| Specifies four threads and applies to the Basic database in the Sample application |

RTDEPCALCOPTIMIZE

Sets whether the @CURRMBRRANGE calculation function behaves as runtime dependent or non runtime dependent.

Syntax

RTDEPCALCOPTIMIZE [appname [dbname]] TRUE | FALSE

- **appname**—Optional. If you specify an application name, the setting applies to all databases within the named application. If you do not specify an application name, the setting applies to all applications and databases on the Essbase Server.

- **dbname**—Optional. If you specify a database name and an application name, the setting applies only to the named database. If you do not also specify an application name, the database is ignored and the setting applies to all applications and databases on the Essbase Server.

- **TRUE**—This is the default. @CURRMBRRANGE behaves as a non runtime dependent formula. This, the default behavior, could result in incorrect calculation results if the @CURGEN or @CURLEV functions are used as arguments to @CURRMBRRANGE, because Essbase would fail to generate the correct dependency list to compute @CURRMBRRANGE.

- **FALSE**—@CURRMBRRANGE behaves as runtime dependent formula, but only when @CURGEN or @CURLEV are passed as an argument to @CURRMBRRANGE. Calculations involving @CURRMBRRANGE may run slowly, as computation of runtime dependent formulas requires more memory.
Example

RTDEPCALCOPTIMIZE FALSE

SERVERTHREADS

Overrides the default value of the number of threads that the application process (ESSSVR) can spawn. Application threads are used in calculations, client requires, administrative activities, etc.

When a transaction is requested, the application process (ESSSVR) assigns a thread to the transaction and releases the thread when the transaction is completed.

Syntax

SERVERTHREADS [appname] n

- **appname**—Optional. Specifies an application; the SERVERTHREADS setting applies to all databases within the named application.
  
  If you do not specify an application, the setting applies to all applications and databases on Essbase Server.

- **n**—Specifies the number of threads that the application process (ESSSVR) can spawn; 20 to 1024, inclusive.
  
  The default value is 20.

  If you specify a value that is:

  - Less than the minimum, Essbase interprets the value as 20
  - Greater than the maximum, Essbase interprets the value as 1024

Notes

- While the actual maximum value you can set is 1024, the maximum number of threads an operating system can handle might be much lower. Before specifying a value greater than the default value, check with your system administrator, as higher values can significantly consume system resources.

- If the computer on which Essbase Server runs freezes while running multiple reports simultaneously, increase the value of SERVERTHREADS by one for each report you run.

- Each application thread may create child threads for tasks such as parallel calculation, parallel data load or export, and parallel restructuring. If the total number of running threads is too high, threads may lose efficiency in contending for server resources.

Example

SERVERTHREADS 25
Allows all applications on Essbase Server to spawn up to 25 threads.

SERVERTHREADS Sample 100

Allows the Sample application on Essbase Server to spawn up to 100 threads.

See Also
AGENTTHREADS
@XREF calculation function

SSANCESTORONTOP

Controls whether users can specify that ancestors be positioned at the top, in grid client operations.

Syntax

SSANCESTORONTOP [appname [dbname]] TRUE | FALSE

- **appname**—Optional. Specifies the application to which the configuration applies.
- **dbname**—Optional. Specifies the cube to which the configuration applies.
- **TRUE**—Smart View users can specify the ancestor position for hierarchies in ad hoc grids.
- **FALSE**—Smart View users cannot specify the ancestor position. This is the default.

Description

If this configuration property is set to TRUE, Smart View users can specify ancestor position for hierarchies in ad hoc grids. By default, this parameter is not enabled, and the ancestor is positioned at the bottom.

Example

SSANCESTORONTOP Sample TRUE

See Also

Specifying Ancestor Position in Ad Hoc Grids in Working with Oracle Smart View for Office (available on the Books tab)

SSMEMBERIDPROCESSING

Controls whether Smart View keeps track of members in a report by using stable member IDs instead of (less stable) uniquely qualified member names.
For a database that has duplicate member names enabled, an internal member ID is associated with each member. However, member IDs are only applicable for cubes deployed from Essbase Business Intelligence Acceleration Wizard.

Syntax

SSMEMBERIDPROCESSING [appname [dbname]] TRUE | FALSE

- **appname**—Optional. Specifies the application for which member IDs should be used.
  
  If you specify a value for *appname* and do not specify a value for *dbname*, the setting applies to all databases in the specified application.

  To enable the setting for a specific database, you must specify an application and database.

  If you do not specify an application, you cannot specify a database, and the setting applies to all applications and databases on Essbase Server.

- **dbname**—Optional. Specifies the database, in the application specified by *appname*, for which member IDs should be used.
  
  If you specify a value for *dbname* but do not specify a value for *appname*, your specification is ignored.

- **TRUE**—Essbase tracks members using stable member IDs. This is the default for BI outlines, if DISPLAY_KEY and MEMBER_VALUE alias tables exist in the outline.

- **FALSE**—Essbase tracks members using qualified member names.

Description

For Smart View reports on duplicate member name outlines, member IDs can help Smart View maintain report validity for all members, even when members in the outline are moved or renamed.

Notes

If you opt to track members using qualified member names instead of member IDs, Smart View reports may become invalid if members in the outline are moved or renamed.

Example

SSMEMBERIDPROCESSING Sample TRUE

SSOPTIMIZEDGRIDPROCESSING

Specifies whether optimized grid processing, which cuts the input grid into symmetric grids to create fewer symmetric queries, is enabled for grid client operations.

Syntax

SSOPTIMIZEDGRIDPROCESSING [appname [dbname]] TRUE | FALSE
• **appname**—Optional. Specifies the application for which optimized grid processing is to be set.

If you specify a value for `appname` and do not specify a value for `dbname`, the setting applies to all databases in the specified application.

To enable the setting for a specific database, you must specify an application and database.

If you do not specify an application, you cannot specify a database, and the setting applies to all applications and databases on Essbase Server.

• **dbname**—Optional. Specifies the database, in the application specified by `appname`, for which optimized grid processing is to be set.

If you specify a value for `dbname` but do not specify a value for `appname`, your specification is ignored.

• **TRUE**—Enables optimized grid processing for grid client operations.

  The default value is TRUE.

• **FALSE**—Disables optimized grid processing for grid client operations.

For changes to the configuration file to take effect, you must restart Essbase Server.

**Example**

```
SSOPTIMIZEDGRIDPROCESSING FALSE
```

Turns off optimized processing for grid client operations on all applications and databases on Essbase Server.

### SSSPROCROWLIMIT

Controls the maximum number of rows Essbase processes on a Smart View or other grid client request.

**Syntax**

```
SSPROCROWLIMIT n
```

**n**—An integer value of 16,384 or higher. The default value is 250,000.

**Description**

`SSPROCROWLIMIT` controls the maximum number of rows Essbase processes on a Smart View or other grid client user request. `SSPROCROWLIMIT` is in effect only for grid clients when the Suppress #Missing Rows option is selected. The rows are counted before suppression; that is, missing rows and rows containing zero values are included.

When users zoom in on one or more members, Essbase must process a larger grid containing selected members expanded to the zoom-in level set in the options. When the Suppress #Missing Rows option is set, Essbase returns only rows with at least one column containing a non-missing value. `SSPROCROWLIMIT` defines the maximum size (number of rows) of the larger grid that Essbase needs to process. This setting prevents excessive memory usage for a single grid operation.
When the Excel Suppress #Missing Rows option is not selected, the limit is 64000.

**Notes**

- SSPROCROWLIMIT applies to unprocessed rows; that is, it is the number of rows Essbase accepts before processing. Row processing eliminates missing rows. After processing, the number of rows that the client can retrieve depends on grid-client-defined limits.
- If SSPROCROWLIMIT is exceeded, Essbase issues an error message and stops processing the request.
- This setting is not used in the Smart View Free form mode.
- Oracle does not recommend using a limit higher than 500,000.

**Example**

SSPROCROWLIMIT 300000

**SUPNA**

Controls whether the Suppress #Missing Rows option in Smart View or another grid client interface suppresses the display of cells for which a user has no access (in addition to suppressing #MISSING rows).

**Syntax**

SUPNA ON | OFF

- **ON**—The Suppress #Missing Rows option suppresses the display of cells for which a user has no access.
- **OFF**—The Suppress #Missing Rows option does not suppress the display of cells for which a user has no access. This is the default.

**Description**

The Suppress #Missing Rows option in Smart View or other grid clients suppresses the display of data rows that contain only missing values. SUPNA specifies whether Essbase also suppresses the display of cells for which a user has no access.

**Example**

SUPNA OFF

For all databases on the server, Essbase does not suppress cells for which a user has no access. These cells appear in the grid as #NoAccess. Rows of missing data are suppressed.
SVRIDLETIME

Sets the number of minutes an Essbase application can be idle before it is shut down. If an application is idle for the time period set here, it will stop running. When any new request is made (for example, Smart View query, Outline, or CLI activity), the application starts automatically.

Syntax

SVRIDLETIME [appname] n

Where n is the number of minutes of idle time permitted before shutdown. The default value is 120 minutes. The minimum value is 1 minute. The maximum value is 20160 minutes (two weeks). To disable automatic shutdown, set n to 0.

Notes

• By default, applications are set to shut down after 2 hours of idle time.
• The SVRIDLETIME value should always be set to a few more minutes than the idle user session logout interval, which you can set using the set session_idle_limit grammar in the MaxL alter system statement.

Example

SVRIDLETIME Sample 15

TARGETASOOPT

Potentially optimizes large queries (from Smart View or other grid clients, MDX, or Report Writer) to an aggregate storage database across a transparent partition when the source outline and target outline are identical in the partition region definition area.

Syntax

TARGETASOOPT [appname] TRUE | FALSE

- **appname**—Optional. Application name. If you specify a value for appname, the setting applies to all databases in the specified application. If you do not specify an application, the setting applies to all applications and databases on the Essbase Server.
- **FALSE**—The default. Optimization is not enabled, even if queries match the required criteria (see Description).
- **TRUE**—Optimization is enabled for queries that match the required criteria (see Description).

When TARGETASOOPT is TRUE, Essbase completes the following steps:

1. When the partition is next validated, automatically determines if the partition region definition outlines are identical on the source and target databases
2. If the partition region definition outlines are identical, the query is sent in the compact format from the target database to the source database.

You must restart Essbase Server to initialize any change to the configuration file.

Description

TargetASOOpt enables an alternate (compact) format for sending a query (from Smart View or other grid clients, MDX, or Report Writer) to an aggregate storage source database, and hence may speed up large queries between databases that match the following criteria:

- Databases are transparently partitioned (for example, to enable write-back for aggregate storage databases)
- Source is an aggregate storage database
- Partitioned area definitions in the source and target are identical (for example in the Sample Basic database, if the partition region definition is @idesc("100"), then the outline hierarchies below Time, Market, Measures, Scenario, and 100, must be identical on the source and target databases)
- Source outline and target outline are identical

Notes

If at query time the source and target outlines have been modified after the last validation, even if the partition region definition outlines are still identical, TARGETASOOPT is disabled for the query. To enable TARGETASOOPT for the query, you must revalidate the partitions.

Example

TARGETASOOPT TRUE

See Also

TARGETTIMESERIESOPT

TARGETTIMESERIESOPT

Globally sets query optimization across transparent partitions for outlines that have a time dimension with Dynamic Time Series members. If this setting is specified, queries with Dynamic Time Series members will incur faster query times. Use this setting only if the time dimensions on the source and target partitions are identical. If the time dimensions on the source and target partitions are not the same, this setting may produce incorrect results. Restart Essbase to enable this setting to take effect for the Dynamic Time Series members that have been enabled at run time.

Syntax

TARGETTIMESERIESOPT TRUE | FALSE

- TRUE—Enables query optimization across transparent partitions for outlines that have a time dimension with Dynamic Time Series members.
- FALSE—Query optimization is not enabled. This is the default.
Example

TARGETTIMESERIESOPT TRUE

See Also

TARGETASOOPT
MaxL

Using MaxL, you can administer and query Oracle Analytics Cloud – Essbase through a scripting language.

- Overview of MaxL and MDX
- How to Read MaxL Railroad Diagrams
- MaxL Data Definition Language (DDL)
- MaxL Statements
- MaxL Definitions
- MaxL Shell Commands
- Reserved Words List
- MaxL BNF
- MaxL Statements (Aggregate Storage)
- MaxL Use Cases

Overview of MaxL and MDX

MaxL is the multi-dimensional database access language for Essbase. MaxL is a practical, expressive interface for administering and querying the Essbase system. With the MaxL language, you use statements to make requests. MaxL statements usually begin with a verb, and read like English sentences.

MaxL has two functional domains:

- MaxL DDL is the data-definition language for Essbase.
  Data definition means structural control of a database system. This includes operations like creation, deletion, and updating of, applications, databases, and database objects. Therefore, statements in MaxL DDL include verbs like CREATE, ALTER, DROP, GRANT, and DISPLAY.

- MDX is the data-manipulation language for Essbase.
  Data manipulation means access to the actual data within a database system. MDX provides the ability to perform advanced data extraction and querying by means of statements that typically include the verb SELECT. The equivalent conceptual tool would be Report Writer.

How to Read MaxL Railroad Diagrams

The MaxL grammar is illustrated using a railroad syntax notation. The railroad diagrams illustrate all the valid (grammatically correct) statements that can be parsed by MaxL.

- Anatomy of MaxL Statements
Anatomy of MaxL Statements

- A **keyword** (see, represented in plain, lower-case font, is a unit of MaxL grammar. Keywords must be entered literally and in the correct order in MaxL statements. See the examples of keywords in the following diagram excerpt:

  ![Keyword Diagram](image)

  - **create database DBS-NAME using non_unique_members;**

- A **terminal**, represented in upper-case without brackets, is replaced by values in the appropriate format as defined in the **Terminals** table. In the above diagram, DBS-NAME is a terminal. Terminals need to be replaced with a valid name; for example, *sample.basic*.

  Keywords cannot be used as terminals, unless enclosed in single quotation marks. For example, to create a database named database, the statement `create database database;` would return an error, but `create database "database";` would work.

- The **semicolon** indicates the end of a statement. Omitting a semicolon, or placing one before the expected end of a statement, results in a syntax error.

- A **non-terminal**, represented in upper-case with angle brackets `< >`, is defined in an additional diagram, usually below the main diagram. No non-terminal is shown here.

### Railroad Diagram Symbols

The following table describes the meaning of symbols used in railroad diagrams.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>⬤</td>
<td>Statement begins here.</td>
</tr>
<tr>
<td>⬤</td>
<td>Statement continues on next line.</td>
</tr>
<tr>
<td>⬤</td>
<td>Statement is continued from previous line.</td>
</tr>
</tbody>
</table>
### Table 5-1  (Cont.) Railroad Diagram Symbols

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Symbol" /></td>
<td>Statement ends here.</td>
</tr>
<tr>
<td><img src="image2" alt="Symbol" /> <img src="image3" alt="Symbol" /></td>
<td>Alternatives: optionally select one keyword. Boldface indicates default if no selection is made.</td>
</tr>
<tr>
<td><img src="image4" alt="Symbol" /> <img src="image5" alt="Symbol" /> <img src="image6" alt="Symbol" /></td>
<td>Alternatives: selection of one keyword is required.</td>
</tr>
<tr>
<td><img src="image7" alt="Symbol" /> , <img src="image8" alt="Symbol" /></td>
<td>A comma-separated list of any length is permitted.</td>
</tr>
<tr>
<td><img src="image9" alt="Symbol" /></td>
<td>TERMINAL-NAME: Word is not further defined. Replace with value of format shown in the Terminals table.</td>
</tr>
<tr>
<td><img src="image10" alt="Symbol" /></td>
<td>&lt;NON-TERMINAL&gt;: Word used in statement is further defined.</td>
</tr>
<tr>
<td><img src="image11" alt="Symbol" /> ::=</td>
<td>&lt;NON-TERMINAL&gt;: Non-terminal used in statements is defined here.</td>
</tr>
</tbody>
</table>

### Sample Railroad Diagram

The following diagram illustrates a variant grammar that parses the following English sentence:

“The quick brown fox jumps over the lazy dog.”

![Diagram](image)
Keywords and variables on the main line (with arrow markings) are required; optional grammar is recessed (lower than the main line). A vertical stack of words represents alternatives. Bold words indicate defaults when no word is chosen.

Valid sentences parse-able by the example grammar may include:

- The fox jumps over the dog. Bold letters indicate a default value when no option is entered; therefore, entry of this statement would be interpreted as The brown fox jumps over the dog.
- The quick brown fox jumps over the dog.
- The red fox jumps over the lazy cat.
- The quick brown fox jumps onto the tired elephant.

MaxL Data Definition Language (DDL)

MaxL DDL is the database definition language for Essbase. MaxL is a practical, expressive interface for administering Essbase. With the MaxL language, you use statements to make requests. MaxL statements begin with a verb and read like English sentences.

In order for Essbase Server to receive MaxL statements, you must pass the statements to Essbase Server using MaxL Shell.

Oracle recommends that you proceed in the following order:

1. Start Essbase Server.
2. Invoke MaxL Shell and log in to Essbase Server.
3. Create statements for data access and system administration.
4. Learn about syntax, numbers, permissions, and names in the MaxL language (see MaxL Definitions).

MaxL Statements

The MaxL data-definition language has its own grammar that you use to create statements. In this document, the syntax for the MaxL DDL is illustrated using railroad diagrams.

The MaxL grammar is case-insensitive. Semicolon statement-terminators are required when using the MaxL Shell.

Key words of the MaxL grammar are represented in this document in lower-case. Terminals, represented in upper-case, are to be replaced by the appropriate names, numbers, privileges, or strings. For more information about components of MaxL statements, see MaxL Definitions.

Topics covered in this section:

- Performance Statistics in MaxL
- Listed By Verbs
- Listed by Objects
- MaxL Statement Reference
Performance Statistics in MaxL

Query database returns medium and long performance statistics for the database and application. The statistics appear as tables in the MaxL output. To gather performance statistics, you must first enable statistics gathering using alter database <dbs-name> set performance statistics enabled. You also use alter database to return to zero the statistical persistence (length) and scope (granularity).

Collecting and analyzing performance statistics can help you understand whether the databases are in good running condition or could use modifications to improve performance.

Topics related to performance statistics:
- The Essbase Performance Statistics Tables
- MaxL Script Example

The Essbase Performance Statistics Tables

The Essbase system gathers a variety of statistics regarding the performance of the system and the connected applications. The output of query database can vary depending on what the system has just done, how long statistics have been gathered and the persistence of the gathered statistics. The tables give information on a typical set of statistics. It can be very helpful to compare two sets of statistics gathered at similar points in the server's operation, such as after two comparable updates or after two restructure operations. Statistics should be gathered at intervals and compared to each other to identify differences. Compare the statistics gathered before and after any changes to the system and if the system performance changes.

Note:

Depending on the calculations you choose to perform, if any, some tables may or may not be displayed in your output log.

Performance statistics for which tables are available:
- Kernel Input/Output Statistics
- Kernel Cache Statistics
- Cache End-Transaction Statistics
- Database Synchronous Input/Output Statistics
- Database Asynchronous Input/Output Statistics
- Dynamic Calc Cache Statistics

Kernel Input/Output Statistics

The Kernel I/O Statistics table summarizes input/output for the entire application. There is one kernel I/O table per application.

Persistence/Scope of this table: med/server
Table 5-2  Kernel IO Statistics

<table>
<thead>
<tr>
<th>Kernel I/O</th>
<th>Read (OS reads from disk)</th>
<th>Write (OS writes to disk)</th>
</tr>
</thead>
<tbody>
<tr>
<td># Index</td>
<td>I/O Number of reads that occurred through the index cache.</td>
<td>Number of writes that occurred through the index cache.</td>
</tr>
<tr>
<td># Data I/O</td>
<td>Number of reads that occurred through the data cache.</td>
<td>Number of writes that occurred through the data cache.</td>
</tr>
<tr>
<td># Fground I/O</td>
<td>Number of data reads that occurred in the foreground (while a process waited for data to be read).</td>
<td>Number of data writes that occurred in the foreground (while a process waited for data to be written).</td>
</tr>
<tr>
<td># Index bytes</td>
<td>Number of bytes read from .IND files.</td>
<td>Number of bytes written to .IND files.</td>
</tr>
<tr>
<td># Data bytes</td>
<td>Number of bytes read from .PAG files.</td>
<td>Number of bytes written to .PAG files.</td>
</tr>
<tr>
<td>Av byte/dat I/O</td>
<td>Average byte size of data reads. A high number is preferable.</td>
<td>Average byte size of data writes. A high number is preferable.</td>
</tr>
</tbody>
</table>

Kernel Cache Statistics

The Kernel Cache Statistics table assists in sizing database caches. Make caches only as large as necessary for optimum performance. Note that cache sizes are listed in order of importance: index, data file, data.

- The index cache is a buffer in memory that holds index pages.
- The data file cache is a physical data cache layer designed to hold compressed data blocks.
- The data cache is a buffer in memory that holds data pages.

The Kernel Cache Statistics table assists you in determining how to size Essbase caches. The Essbase kernel uses these caches to manage memory. As a rule, data that is useful to processes should be kept in memory rather than on a disk. Replacements occur when something needed for a process is moved from disk to cache and something in the cache is thrown away to make room for it.

Use this table to help you decide how to size your caches. Make the caches as small as possible; however, if replacements for a cache are greater than 0, the cache may be too small. Appropriate sizing of the Index cache is the most important for optimal performance; appropriate sizing of the Data cache is the least important.

Persistence/Scope of this table: long/db

Table 5-3  Kernel Cache Statistics

<table>
<thead>
<tr>
<th>Kernel Cache Statistic</th>
<th>Description</th>
</tr>
</thead>
</table>
Table 5-3  (Cont.) Kernel Cache Statistics

<table>
<thead>
<tr>
<th>Kernel Cache Statistic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td># Replacements</td>
<td>Number of replacements per cache. Replacements occur when data moves from disk to cache and something in the cache is deleted to make room. If the number or replacements is low or zero, the cache might be set too large.</td>
</tr>
<tr>
<td># Dirty repl</td>
<td>Number of dirty replacements per cache. A dirty replacement is one that requires a write to the disk before cache memory can be reused by a process. The data needed for the process is “dirty” because it was modified in memory but not saved to the disk. Dirty replacements are inefficient and expensive. They indicate that a cache might be too small.</td>
</tr>
<tr>
<td>Log blk xfer in</td>
<td>Number of logical blocks transferred to the Data file cache and Data cache (this measurement is not applicable for the Index cache.) If you are changing cache sizes, it may be instructive to study this statistic and note changes in data traffic.</td>
</tr>
</tbody>
</table>

Cache End-Transaction Statistics

The **Cache End-Transaction Statistics** table measures DBWriter efficiency. DBWriter is an asynchronous (or no-wait) Essbase thread, which searches the cache finding information that needs to be written to a disk.

The Cache End-Transaction Statistics table shows the cleanup state at the end of a transaction. These statistics are designed to measure DBWriter efficiency. DBWriter is an asynchronous (or no-wait) thread, which searches the cache and finds information that needs to be written to a disk. Because the DBWriter only operates during idle times, measuring the DBWriter activity can give an idea of the amount of idle time. This number should be high, indicating that the DBWriter had enough idle time to support the database effectively. Keep these statistics available for diagnostic purposes, in case you need to call technical support.

Persistence/Scope of this table: **med/db**

Database Synchronous Input/Output Statistics

The **Database Synchronous I/O** table tracks synchronous input/output. Synchronous means that the thread or program waits for the I/O to finish before proceeding. The **Tave (us)** column shows the bandwidth (bytes/Ttotal).

Persistence/Scope of this table: **med/db**
### Table 5-4  DB Sync IO Statistics

<table>
<thead>
<tr>
<th>DataBase Synch I/O</th>
<th>Count</th>
<th>Bytes</th>
<th>Ttotal (ms)</th>
<th>Tave (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index Read</td>
<td>Number of times the OS went to the disk to read a .IND file.</td>
<td>Number of bytes the OS read from .IND files.</td>
<td>Total amount of time the OS took to complete index reads.</td>
<td>Average amount of time the OS took to complete one index read. This equals $\frac{\text{Ttotal (ms)}}{\text{Count}}$.</td>
</tr>
<tr>
<td>Index Write</td>
<td>Number of times the OS wrote information to a .IND file.</td>
<td>Number of bytes the OS wrote to .IND files.</td>
<td>Total amount of time the OS took to complete index writes.</td>
<td>Average amount of time the OS took to complete one index write. This equals $\frac{\text{Ttotal (ms)}}{\text{Count}}$.</td>
</tr>
<tr>
<td>Data Read</td>
<td>Number of times the OS went to the disk to read a .PAG file.</td>
<td>Number of bytes the OS read from .PAG files.</td>
<td>Total amount of time the OS took to complete data reads.</td>
<td>Average amount of time the OS took to complete one data read. This equals $\frac{\text{Ttotal (ms)}}{\text{Count}}$.</td>
</tr>
<tr>
<td>Data Write</td>
<td>Number of times the OS wrote information to a .PAG file.</td>
<td>Number of bytes the OS wrote to .PAG files.</td>
<td>Total amount of time the OS took to complete data writes.</td>
<td>Average amount of time the OS took to complete one data write. This equals $\frac{\text{Ttotal (ms)}}{\text{Count}}$.</td>
</tr>
</tbody>
</table>

**Note:**

Bandwidth = bytes/Ttotal. Average bandwidth = bytes/Tave.

### Database Asynchronous Input/Output Statistics

The Database Asynchronous I/O table tracks asynchronous input/output. Asynchronous means no-wait: the I/O happens at an unknown time, while the program does other things. The effective bandwidth for the application is determined by bytes/Twait.

Persistence/Scope of this table: med/db

### Table 5-5  DB Async IO Statistics

<table>
<thead>
<tr>
<th>DataBase Asynch I/O</th>
<th>Count</th>
<th>Bytes</th>
<th>Ttotal (ms)</th>
<th>Tave (ms)</th>
<th>Twait (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index Read</td>
<td>Number of times the OS went to the disk to read a .IND file.</td>
<td>Number of bytes the OS read from .IND files.</td>
<td>Time elapsed between request for an index read, and verification of its completion.</td>
<td>Average time elapsed between requests for index reads, and verification of their completion.</td>
<td>Wait time if the OS had not completed index reads at the time polled.</td>
</tr>
</tbody>
</table>
### Table 5-5  (Cont.) DB Async IO Statistics

<table>
<thead>
<tr>
<th>Database Asynch I/O</th>
<th>Count</th>
<th>Bytes</th>
<th>Ttotal (ms)</th>
<th>Tave (ms)</th>
<th>Twait (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index Write</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>An occurrence of the OS writing index information to a .IND file.</td>
<td>Number of times the OS wrote information to a .IND file.</td>
<td>Number of bytes the OS wrote to .IND files.</td>
<td>Time elapsed between request for an index write, and verification of its completion.</td>
<td>Average time elapsed between requests for index writes and verification of their completion.</td>
<td>Wait time if the OS had not completed index writes at the time polled.</td>
</tr>
<tr>
<td>Data Read</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>An occurrence of the OS reading information from a .PAG file on the disk.</td>
<td>Number of times the OS went to the disk to read to a .PAG file.</td>
<td>Number of bytes the OS read from .PAG files.</td>
<td>Time elapsed between request for a data read, and verification of its completion.</td>
<td>Average time elapsed between requests for data reads, and verification of their completion.</td>
<td>Wait time if the OS had not completed data reads at the time polled.</td>
</tr>
<tr>
<td>Data Write</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>An occurrence of the OS writing data to a .PAG file.</td>
<td>Number of times the OS wrote information to a .PAG file.</td>
<td>Number of bytes the OS wrote to .PAG files.</td>
<td>Time elapsed between request for a data write, and verification of its completion.</td>
<td>Average time elapsed between requests for data writes and verification of their completion.</td>
<td>Wait time if the OS had not completed data writes at the time polled.</td>
</tr>
</tbody>
</table>

**Note:**

(1) Because asynchronous I/O is ideally no-wait, and happens at an unknown time, you cannot determine how long reads and writes actually took to complete. (2) You cannot determine the bandwidth (bytes per microsecond). Effective bandwidth, as seen by the application, is determined by bytes/Twait.

### Dynamic Calc Cache Statistics

The **Dynamic Calc Cache table** shows where blocks that are expanded to contain calculated members (BigBlks) are calculated: in dynamic calculator cache (DCC), or in regular memory (nonDCC). By viewing the total number of big blocks allocated versus the maximum number of big blocks held simultaneously, and by analyzing block wait statistics, you can determine the efficiency of your dynamic calc cache configuration settings. For more information, refer to the DYNCALCCACHEMAXSIZE configuration setting.
### Table 5-6  Dynamic Calc Cache Statistics

<table>
<thead>
<tr>
<th>Dynamic Calc Cache Statistic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BigBlks Allocated</td>
<td>The number of big block allocations that have been requested, so far, irrespective of where the system got the memory (DC cache or regular). For three queries Q1, Q2, and Q3 executed, requiring 25, 35, and 10 big blocks, respectively, BigBlks Allocated would be 70. This does not mean that Q1 needed all 25 blocks at the same time. It may have used some blocks for a while, then released some of them, and so on, until the query finished and released all remaining blocks (returned to DC cache or regular memory).</td>
</tr>
<tr>
<td>Max BigBlks Held</td>
<td>The maximum number of big blocks simultaneously held, so far. For each query Qi executed so far, there will be a number Ni, which gives the maximum number of big blocks that the query needed to have at the same time (includes both DCC and regular memory blocks). MaxBigBlksHeld under the Total column is the maximum over all values of Ni. The values under the DCC and non-DCC columns are similar except that they restrict themselves to the maximum blocks held in the respective portions of memory.</td>
</tr>
<tr>
<td>DCC Blks Waited</td>
<td>The number of dynamic calculator blocks that the system had to wait for.</td>
</tr>
<tr>
<td>DCC Blks Timeout</td>
<td>The number of times that Essbase timed out waiting for free space in the dynamic calculator cache.</td>
</tr>
<tr>
<td>DCC Max ThdQLen</td>
<td>The highest number of threads that were left waiting in queue for Dynamic Calc cache memory to be freed.</td>
</tr>
</tbody>
</table>

### MaxL Script Example

The following MaxL script creates an output file of performance statistics tables.

```maxl
/* to execute: 
essmsh scriptname username password
*/
login $1 $2;
spool on to 'c:\mxlouts\pstatsouts.txt';
alter database sample.basic set performance statistics enabled;
execute calculation
  'SET MSG ERROR;
    CALC ALL;'
on Sample.basic;
alter database sample.basic set performance statistics mode to medium persistence server scope;
query database sample.basic get performance statistics kernel_io table;
alter database sample.basic set performance statistics mode to long
```
persistence database scope;
query database sample.basic get performance statistics kernel_cache table;
alter database sample.basic set performance statistics mode to medium
persistence database scope;
query database sample.basic get performance statistics end_transaction
table;
query database sample.basic get performance statistics database_synch
table;
query database sample.basic get performance statistics database_asynch
table;
spool off;
logout;

Listed By Verbs

alter
create
display
drop
execute
export
grant
import
query
refresh

Alter

application
database
drillthrough
filter
object
partition
session
system		
tablespace
trigger

Create

application
display

drop
drillthrough
filter
location alias
lock
object
partition
trigger
trigger spool

Execute
aggregate process
aggregate selection
aggregate build
allocation
calculation
custom calculation (aggregate storage)

Export
data
LRO
outline
query_tracking

Grant
Grant

Import
data
dimensions
lro
query_tracking

Query
database
database backup archive file
application (for aggregate storage)
application (for block storage)

Refresh

outline
replicated partition

Listed by Objects

aggregate_build
aggregate_process
aggregate_selection
application
archive_file
calculation
data
database
dimensions
drillthrough
filter
group
location alias
lock
lro
object
outline
partition
privilege
session
system
tablespace
trigger
trigger spool
user
variable
Aggregate Build
   execute aggregate build

Aggregate Process
   execute aggregate process

Aggregate Selection
   execute aggregate selection

Allocation
   execute allocation

Application
   alter
   create
   display
   drop
   query (for aggregate storage only)

Archive_file
   query

Calculation
   create
   display
   drop
   execute
   execute custom (aggregate storage)

Data
   export
   import

Database
   alter
   create
display
drop
query

Dimensions
import

Drillthrough
alter
cREATE
display
drop

Filter
alter filter
create filter
display filter
display filter row
drop filter

Group
display

Location Alias
create
display
drop

Lock
display
drop

LRO
export
import
Object

alter
display
drop

Outline

create
refresh
see also Dimensions

Partition

alter
create
display
drop
refresh replicated
refresh outline for outline synchronization

Privilege

display
grant

Session

alter
display
alter system to stop a session

System

alter
display

Tablespace

alter
display
Trigger

- alter
- create or replace
- display
- drop

Trigger Spool

- display
- drop

User

- display
- grant to assign permissions

Variable

- display variable
- To add, drop, or set substitution variables:
  - alter application
  - alter database
  - alter system

MaxL Statement Reference

Consult the Contents pane for an alphabetical list of MaxL statements, or see Listed By Verbs.

Alter Application

- Click here for aggregate storage version
- Change application-wide settings.
- Permission required: Application Manager.

Syntax
Use `alter application` to change the following application-wide settings:

**Keywords**

- **set lock_timeout**
  Change the maximum time interval that locks on data blocks can be held by Smart View (or other grid clients') users. When a client data-block lock is held for more than the time out interval, Essbase removes the lock and the transaction is rolled back. The default interval is 60 minutes. This setting affects all databases in the application.

- **set max_iro_file_size**
  Specify a maximum file size for Linked Reporting Objects (LRO) attachments. There is no default. There is no minimum or maximum value, excepting limitations imposed by your system resources.

- **set minimum permission**
  Grant all users a minimum level of permission to all databases in the application. Users with higher permissions than this minimum are not affected.
set variable
Assign a string value to an existing substitution-variable name. If the variable does not exist, first create it using **add variable**. Substitution variables may be referenced by calculations in the application.

set cache_size
Set the maximum size to which the application cache may grow. The application cache grows dynamically until it reaches this limit. The application cache is used for hybrid aggregation in block storage databases, and can help you manage memory usage for retrievals. This setting takes effect after you restart the application. To check the currently set limit, use the following MaxL statement:

```maxl
query application APP-NAME get cache_size;
```

set type unicode_mode
Migrate an application to Unicode mode. Migration to Unicode mode cannot be reversed.

load database
Start (by loading into memory) an idle database. The statement will fail if you do not have at least read privilege for the database.

unload database
Stop (by unloading from memory) an active database. The statement will fail if you do not have at least read privilege for the database.

enable startup
Permit all users to load (start) the application. This only applies to users who have at least read privilege for the application. Startup is enabled by default.

disable startup
Prevent all users from loading (starting) the application. Startup is enabled by default.

enable autostartup
Start the application automatically when Essbase Server starts. By default, autostartup is disabled.

disable autostartup
Do not start the application automatically when Essbase Server starts. By default, autostartup is disabled.

enable commands
Allow all users with sufficient permissions to make requests to databases in the application. Use to reverse the effect of **disable commands**. The disable commands setting remains in effect only for the duration of your session. By default, commands are enabled.

disable commands
Prevent all requests to databases in the application, including non-data-specific requests, such as viewing database information or changing database settings. All users are affected, including other administrators. Administrators are affected by this setting as a safety mechanism to prevent accidental updates to databases during maintenance operations. This setting remains in effect only for the duration of your session. The setting takes effect immediately, and affects users who are currently logged in, as well as users who log in later during your session.
Caution:
If performing maintenance operations that require disabling commands, you must make those maintenance operations within the same session and the same script as the one in which commands were disabled.

By default, commands are enabled.

**enable updates**
Allow all users with sufficient permissions to make requests to databases in the application. Use to reverse the effect of disable updates. Disabling updates remains in effect only for the duration of your session. By default, updates are enabled.

**disable updates**
Prevent all users from making requests to databases in the application. Use before performing update and maintenance operations. The disable updates setting remains in effect only for the duration of your session.

Caution:
If performing maintenance operations that require updates to be disabled, you must make those maintenance operations within the same session and the same script as the one in which updates were disabled. By default, updates are enabled.

**enable connects**
Allow all users with sufficient permissions to make connections to databases in the application. Use to reverse the effect of disable connects. By default, connections are enabled.

**disable connects**
Prevent any user with a permission lower than Application Managers from making connections to the databases that require the databases to be started. This includes starting the databases or performing the ESSCMD shell SELECT command on the databases. Database connections remain disabled for all databases in the application, until the application setting is re-enabled by the administrator. By default, connections are enabled.

**enable security**
When security is disabled, Essbase ignores all security settings in the application and treats all users as Application Managers. By default, security is enabled.

**disable security**
When security is disabled, Essbase ignores all security settings in the application and treats all users as Application Managers. By default, security is enabled.

**comment**
Enter an application description (optional). The description can contain up to 80 characters.
**clear logfile**
Delete the application log located in the application directory. A new log is created for entries recording subsequent application activity.

**add variable**
Create an application-level substitution variable by name, and optionally assign a string value for the variable to represent. You can assign or change the value later using `set variable`. A substitution variable acts as a global placeholder for information that changes regularly. Substitution variables may be referenced by calculations and report scripts.
If substitution variables with the same name exist at server, application, and database levels, the order of precedence for the variables is as follows: a database level substitution variable supersedes an application level variable, which supersedes a server level variable.

**drop variable**
Remove a substitution variable and its corresponding value from the application.

**rename to**
Rename the application. When you rename an application, the application and the application directory (`ARBORPATH\app\appname`) are renamed.

**Example**

```
alter application Sample set minimum permission read;
```

Grants all users read access to all databases in the Sample application. Users can retrieve data values and run report scripts.

```
alter application Sample disable commands;
```

Prevents all users from making requests to the application scope. Use this statement before performing application-wide update and maintenance operations.

```
alter application Acme set variable Current_month July;
```

Assigns the string value July to the substitution variable "Current_month." "Current_month" may be referenced by calculations in the Acme application.

**Alter Database**

Click here for aggregate storage version

Select a subset of **alter database**:

- Alter Database enable | disable
- Alter Database Set
• Alter Database (Misc)

Alter Database enable | disable

Click here for aggregate storage version

Change database-wide settings.

Permission required: create_application.

Syntax

```
alter database DBS-NAME enable | disable
```

**DBS-NAME**

Use *alter database* to change the following database-wide settings:

**Keywords**

**enable two_pass_calc**
Recalculate (after a default calculation) database outline members tagged as Two Pass, so they will be recalculated after other database members have been consolidated. This setting is enabled by default.

Members that usually require a two-pass calculation are those members of the Accounts dimension that are calculated by a formula rather than by hierarchical consolidation. These members are typically ratios, such as "Profit % Sales" (profit percentage of sales), which has a member formula.

This setting is ignored during a calculation script; it is used only during a default calculation. To use two-pass calculation in a non-default calculation, use the CALC TWOPASS command in the calculation script.

**disable two_pass_calc**
Do not recalculate database outline members tagged as Two Pass after a default calculation. Two-pass calculation is enabled by default.

**enable aggregate_missing**
Consolidate #MISSING values along with the regular database consolidation. If you never load data at parent levels, aggregating #MISSING values can improve calculation performance, depending on the ratio between upper level blocks and input blocks in the database.

If this setting is enabled and you load values directly at the parent level, these parent-level values will be replaced by the results of the consolidation, even if the results are #MISSING values. The aggregate missing setting is disabled by default.
disable aggregate_missing
Do not consolidate #MISSING values. This is the default. Data that is loaded at parent levels is not overwritten by #MISSING values of children below it. However, if any of the child data values are not #MISSING, these values are consolidated and overwrite the parent values.

enable startup
Enable users to start the database directly or as a result of requests requiring the database to be started. Startup is enabled by default.

disable startup
Prevent all users from starting the database directly or as a result of requests that would start the database. Startup is enabled by default.

enable autostartup
Automatically start the database when the application to which it belongs starts. Autostartup is enabled by default. This setting is applicable only when startup is enabled.

disable autostartup
Prevent automatic starting of the database when the application to which it belongs starts. Autostartup is enabled by default.

enable compression
Enable data compression. By default, Bitmap compression is enabled. To switch to a different compression type, use alter database set compression.

disable compression
Disable data compression. By default, Bitmap compression is enabled.

enable create_blocks
Allow Essbase to create a data block when you assign a non-constant value to a member combination for which a data block does not already exist. Block creation on equation is disabled by default, because it can result in a very large database. When you assign a constant to a member on a sparse dimension, you do not need to enable Create Blocks on Equation, because Essbase would create a data block anyway. For example, "West = 5;" would result in the creation of data blocks, with or without the Create Blocks on Equation setting enabled. You do need to check this option if you want blocks created when you assign anything other than a constant to a member on a sparse dimension for which a data block does not already exist. For example, if no data exists for Actuals, a member of a sparse Scenario dimension, then you need to enable Create Blocks on Equation in order to perform the following allocation:

2002Forecast = Actuals * 1.05;

disable create_blocks
Turn off the Create Blocks on Equation setting. The setting is disabled by default.

enable committed_mode
Set the database isolation level to committed access, meaning that only one transaction at a time can update data blocks. Essbase holds read/write locks on all data blocks until the transaction and the commit operations are performed. If pre-image access is enabled, users (or transactions) can still have read-only access to
data at its last commit point. For more information, see the enable pre_image_access setting. The default isolation-level mode is Uncommitted.

**disable committed_mode**
Turn off the Committed Mode setting, reverting to the default isolation level of Uncommitted for the database.

**Note:**
Smart View and other grid clients’ data-update operations are always in committed mode.

In uncommitted mode, Essbase allows transactions to hold read/write locks on a block-by-block basis. Essbase releases a block after it is updated, but does not commit blocks until the transaction is completed, or until a specified number of blocks or rows (a "synchronization point") has been reached. You can set this limit using the implicit_commit settings.

**enable pre_image_access**
Allow users (or other transactions) read-only access to data at its last commit point, when the database is in committed mode (meaning that data blocks may be locked for the duration of a concurrent transaction). Pre-image access is enabled by default when the database is in committed mode.
See also the enable committed_mode setting.

**disable pre_image_access**
Disable pre-image access, disallowing read-only access to locked blocks of data at their last commit point (this setting is only applicable while the database is in committed mode). Pre-image access is enabled by default when the database is in committed mode.

**Example**

```maxl
alter database Sample.Basic disable two_pass_calc;
```

Prevents recalculation (after a default calculation) of members tagged as Two Pass.

**Alter Database Set**

Click here for aggregate storage version

Change database-wide settings.

Permission required: create_application.

**Syntax**
Use `alter database set` to change the following database-wide settings:

**Keywords**

- `DBS-NAME`
- `SIZE-STRING`
- `DBS-STRING`
- `MEMBER-NAME`
- `DBS-SYSTEM-ROLE`
- `INTEGER`
- `VARIABLE-NAME`
- `CALC-NAME-SINGLE`
- `CALC-STRING`
- `ALT-NAME-SINGLE`
- `COMMENT-STRING`

`retrieve_buffer_size` size

Change the database retrieval buffer size. This buffer holds extracted row data cells before they are evaluated by the RESTRICT or TOP/BOTTOM Report Writer commands. The default size is 20 KB. The minimum size is 2 KB. Increasing the size may improve retrieval performance.

`retrieve_sort_buffer_size` size

Change the database retrieval sort buffer size. This buffer holds data until it is sorted. The Report Writer and Essbase Query Designer use the retrieval sort buffer. The
default size is 20 KB. The minimum size is 2 KB. Increasing the size may improve retrieval performance.

**data_cache_size**
Change the data cache size. The data cache is a buffer in memory that holds uncompressed data blocks. Essbase Server allocates memory to the data cache during data load, calculation, and retrieval operations as needed. The default and minimum size is 3072 KB.

**index_cache_size**
Change the index cache size. The index cache is a buffer in memory that holds index pages. When a data block is requested, Essbase looks at the index pages in the index cache to find its location on disk.

- Minimum value: 1 MB (1,048,576 bytes)
- Maximum value: 256 TB
  
  Default value for buffered I/O: 1 MB (1,048,576 bytes)

Buffered I/O is the default for this release.

**currency_database**
Link the database with a currency database. A currency database enables you to convert currency values in a database from one currency into another currency.

**currency_member**
Specify the member to use as a default value in currency conversions. You can specify any valid member of the dimension defined as “Currency Type” in the currency database.

**currency_conversion**
Specify whether during currency conversion, the calculation method multiplies the currency database exchange rates with the main database values, or that the currency database exchange rates are divided by the main database values.

**minimum permission**
Set a level of permission that all users or groups can have to the database. Users or groups with higher granted permissions than the minimum permission are not affected.

**compression rle**
Set the database to use run-length encoding (RLE) compression. Essbase compresses repetitive, consecutive values, including zeros and #MISSING values. The default compression type is bitmap.
When a compressed data block is brought into the data cache, Essbase expands the block to its full size, regardless of the scheme that was used to compress it.

**compression bitmap**
Set the database to use bitmap compression, the default. Essbase stores only non-missing values and uses a bitmapping scheme.
When a compressed data block is brought into the data cache, Essbase expands the block to its full size, regardless of the scheme that was used to compress it.

**lock_timeout**
Change the interval to wait for blocks to be unlocked when the database is in committed mode. If a transaction request is made that cannot be granted in the allotted time, the transaction is rolled back until a lock can be granted.
**Note:**

Smart View and other grid clients’ data-update operations are always in committed mode.

**implicit_commit after <number> blocks**

When uncommitted access is enabled, set the frequency at which Essbase commits data blocks (after the specified number of blocks has been reached). The default frequency, if unspecified, is 3000, and may adjust dynamically during a calculation.

If Essbase Server runs on Oracle Exalytics In-Memory machine, for calculation and data load requests, the commit happens at the end of the command or request, and the default interval of 3000 (or any other value you specify) is ignored.

**implicit_commit after <number> rows**

When uncommitted access is enabled, set the frequency at which Essbase commits data blocks (after the specified number of rows has been reached).

**variable**

Change the value of an existing substitution variable on the database. The value must not exceed 256 bytes. It may contain any character except a leading ampersand (&).

**default calculation**

Change the default calculation (which, by default, is `CALC ALL;`) to the stored calculation script you specify, or to an anonymous (unstored) calculation string.

**active alias_table**

Set an alias table as the primary table for reporting and any additional alias requests. Only one alias table can be used at a time. This setting is user-specific; it only sets the active alias table for the user issuing the statement.

**performance statistics enabled**

Turn on performance-statistics gathering. You might do this when you want to tune the system, change hardware configuration, or monitor I/O. The measurement begins for current processes as soon as you enable it. Any subsequent queries for statistics return measurements spanning from the time of enablement to the time of the query. Performance statistics can be retrieved using `query database`.

**performance statistics disabled**

Turn off performance-statistics gathering. This halts the collection of statistics; it does not prevent anyone from retrieving old statistics using `query database`.

**performance statistics mode to <PST-SPEC>**

Reset performance-statistics gathering for a specified persistence and scope. Each of the statistics tables available using `query database` has a pre-defined persistence and scope. When you use `set performance statistics mode`, you select the persistence and scope to reset, and the collecting of measurements starts over for the applicable tables.
Create an informational note about the database that Smart View or other grid client users can see from the login dialog box. For example, 'Calc in progress: do not update.' Database notes can be up to 64 kilobytes long.

**Example**

```
alter database Sample.Basic set lock_timeout after 120;
```

Changes the number of seconds to wait for blocks to be unlocked. If a transaction request is made which cannot be granted in 120 seconds, the transaction is rolled back until a lock can be granted.

**Alter Database (Misc)**

- [Click here for aggregate storage version](#)
- Change database-wide settings.
- Permission required: create_application.

**Syntax**
Use `alter database` to change the following database-wide settings:

**Keywords**

- **reset**
  Clear all data and linked-reporting objects from the database, but preserve the outline.

- **reset all**
  Clear all data, Linked Reporting Objects, and the outline.

- **reset data**
  Same as using `reset`.

- **force restructure**
  Explicitly restructure the database to eliminate or reduce fragmentation. By default, this statement is run in serial. To enable parallel restructuring, see `RESTRUCTURETHREADS`.

**Variables**

- **DBS-NAME**
- **FILE-NAME**
- **ALT-NAME-SINGLE**
- **USER-NAME**
- **DATE**
- **LOG-TIME**
- **ID-RANGE**
- **DBS-STRING**
- **COMMENT-STRING**
**load alias_table**
Load an alias table from a file to the current database. The feeder file *(FILE-NAME)* must follow these rules:
- Must be correctly formatted.
- Must be located on the Essbase Server computer, not on a client computer.
- *(FILE-NAME)* must include the full path.

Sample contents of a feeder file for loading an alias table:

```
$ALT_NAME
"400-10"        Guava
"400-20"        Tangerine
"400-30"        Mango
$END
```

**unload alias_table**
Delete the specified alias table.

**add variable**
Create a database-level substitution variable by name, and optionally assign a string value for the variable to represent. You can assign or change the value later using *(set variable)*. A substitution variable acts as a global placeholder for information that changes regularly. Substitution variables may be referenced by calculations and report scripts.

If substitution variables with the same name exist at server, application, and database levels, the order of precedence for the variables is as follows: a database level substitution variable supersedes an application level variable, which supersedes a server level variable.

**drop variable**
Remove a substitution variable and its corresponding value from the database.

**delete lro**
Delete Linked Reporting Objects linked to the active database for a given user name or modification date.

**unlock all objects**
Unlock all objects on the database that are in use by a user or process.

**begin archive to file**
Prepare the database for backup by an archiving program, and prevent writing to the files during backup.

This statement requires the database to be started.

Begin archive achieves the following outcomes:
- Commits any modified data to disk.
- Switches the database to read-only mode. The read-only state persists, even after the application is restarted, until it is changed back to read-write using *(end archive)*.
- Reopens the database files in shared, read-only mode.
• Creates a file containing a list of files that need to be backed up. Unless a different path is specified, the file is stored in the database directory.

Begin archive and end archive do not perform the backup; they simply protect the database during the backup process.

**Note:**

Using the `begin archive to file` and `end archive` grammar is the only supported way to initiate backup and recovery of a database using MaxL.

**end archive**

Return the database to read-write mode after backing up the database files. This statement requires the database to be started.

End archive achieves the following outcomes:

• Returns the database to read-write mode.
• Re-opens database files in exclusive, read-write mode.

**Note:**

Using the `begin archive to file` and `end archive` grammar is the only supported way to initiate backup and recovery of a database using MaxL.

**replay transactions**

Replays the database transactions that were logged after the last replay request was originally executed or after the last restored backup's time (whichever occurred later). Transactions that are executed and logged after the restore operation are not replayed, unless you replay those transactions using their sequence IDs. After restoring a database, Oracle recommends that you finish replaying the transactions that were logged after the backup and before the restore and that are needed to fully recover the database; then you can continue executing new transactions.

**replay transactions after LOG-TIME**

Replays the transactions that were logged after the specified time. Enclose the TIME value in quotation marks; for example: ‘11_20_2007:12:20:00’

**replay transactions using sequence_id_range ID-RANGE**

Replays the transactions specified by a comma-separated list of sequence ID ranges. A range can consist of:

• A single transaction: $n$ to $n$; for example, 1 to 1
• Multiple transactions: $x$ to $y$; for example, 20 to 100

Each logged transaction is assigned a sequence ID, indicating the order in which the transaction was performed. To ensure the integrity of the restored data after a replay, Essbase enforces the replay of transactions in the same order in which they were originally performed. The order of sequence IDs are tracked across multiple replay commands.
Note:
You can skip replaying a transaction if you are absolutely sure that the transaction results are not required to recover the database.

rename to
Rename the database. When you rename a database, the database directory is also renamed.

comment
Create a description of the database. The maximum number of characters is 80. This description is available to database administrators. To annotate the database for Smart View or other grid client users, use set note.

Example

```
alter database Sample.Basic begin archive to file 'samplebasic.arc';
```

Backs up the Sample.Basic database files to the specified archive file (samplebasic.arc).

```
alter database Sample.Basic end archive
```

Returns the Sample.Basic database to read-write mode after backing up the database files.

```
alter database Sample.Basic replay transactions using sequence_id_range 1 to 10,20 to 100;
```

Replays the transactions in the Sample.Basic database with sequence IDs 1 through 10 and 20 through 100.

```
alter database Sample.Basic replay transactions after '11_20_2007:12:20:00';
```

Replays all transactions that were logged after the specified time.

Related Topics
- Alter Database enable | disable
- Alter Database Set

Alter Drillthrough
Edit drill-through URL definitions used to link to content hosted on Oracle ERP and EPM applications.
Syntax

```plaintext
alter drillthrough URL-NAME from xml_file FILE-NAME
```

```plaintext
on [ MEMBER-EXPRESSION ]
```

• URL-NAME
• FILE-NAME
• MEMBER-EXPRESSION

Use `alter drillthrough` to edit a URL definition in the following ways:

Keywords

`alter drillthrough`
Edit drill-through URL metadata.
The number of drill-through URLs per database is limited to 255.

`from xml_file`
Indicate the path to the local URL XML file that defines the link information.
The URL XML is created by the ERP or EPM application that deployed the Essbase database. The XML contains the drill-through URL display name as well as a URL enabling the hyperlink from a cell to a Web interface to occur. For a sample URL XML file, see Create Drillthrough.

`on {<member-expression>,...}
Define the list of drillable regions, using the same Essbase member-set calculation language that is used to define security filters. The list of drillable regions must be enclosed in {brackets}.
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.

`allow_merge`
Optional: Merge the drillable-region definition instead of replacing it on update.

Example

```plaintext
alter drillthrough sample.basic.myURL from xml_file "C:/drillthrough/data/myfile.xml" on '{@children("Qtr1"), '@children("Qtr2")}' allow_merge;
```

**Alter Filter**

Add filter rows to a database security filter. Filters control security for database objects. Use `grant` to assign filters to users and groups.

Minimum permission required: Database Manager.

Syntax
Use `alter filter` in the following ways to edit a filter:

**Keywords**

- `alter filter ... add no_access on <member-expression>`
  Block access to a specified member combination.

- `alter filter ... add read on <member-expression>`
  Provide read-only access to a specified member combination.

- `alter filter ... add write on <member-expression>`
  Provide write access to a specified member combination.

- `alter filter ... add meta_read on <member-expression>`
  Restrict access to siblings and ancestors of the member expression. In case of a filtering conflict, the MetaRead filtering overrides the other filter permissions. For more information about metadata filtering, see Metadata Filtering.

**Notes**

- Filters created using MaxL must be valid. For information about filter syntax, see Designing and Maintaining Essbase Cubes.
- `MEMBER-EXPRESSION` must be enclosed in single quotation marks. It can be a comma-separated list.

**Example**

```
alter filter sample.basic.filt7 add read on '@Descendants("East")';
```

Adds a row to a Sample.Basic filter named filt7, giving read-only access to the data for the eastern states.

```
alter filter sample.basic.filt8 add read on '@Descendants("East")', add write on '@Descendants("West")';
```

Adds two rows to a Sample.Basic filter named filt8.

**Alter Object**

Rename, unlock, or copy a database-related artifact.
Syntax

```
alter object OBJ-NAME of type <OBJ-TYPE>
```

- `OBJ-NAME`
- `OBJ-NAME-SINGLE`

Use `alter object` to edit artifacts in the following ways:

**Keywords**

**rename to**

Rename the artifact. Not applicable for partition files, worksheets, or outlines.

**unlock**

Unlock an artifact that is locked by another user or process. Not applicable for alias tables and worksheets. Unlocking an artifact of type `.lro` is applicable for stored linked-reporting objects only; that is, files with the `.LRO` extension.

Note:

To unlock all database artifacts, use `alter database DBS-NAME unlock all objects;`

**copy to**

Make a copy of a server artifact. Not applicable for partition files, worksheets, or outlines. If an artifact of the new name already exists, it is replaced.

**force copy to**

Make a copy of a server artifact. Not applicable for partition files, worksheets, or outlines. If an artifact of the new name already exists, it is replaced. If an administrator issues the statement with the `force` keyword, locked artifacts are unlocked, copied, and re-locked.

**Notes**

- Specified artifacts must be persisted in the database directory.
- Attempting to rename or copy an artifact of type "partition_file" returns an error.

**Example**

```
alter object sample.basic.genref of type rules_file rename to 'level';
```
Renames a rules file in the Sample.Basic directory, named genref.rul, to level.rul.

```
alter object sample.basic.Calcdat of type text rename to 'c_data';
```

Renames a text file in the Sample.Basic directory, named calcdat.txt, to c_data.txt.

```
alter object samppart.company.company of type partition_file unlock;
```

Unlocks the partition definition file for the Samppart Company database.

### Alter Partition

Fix invalid or dangling partition references. Change the authorized user who can connect to both cubes. Change the name of an application, cube, or host (in the event that something was renamed).

**Syntax**

```
alter transparent replicated partition DBS-NAME to DBS-NAME from DBS-NAME at HOST-NAME
```

*connect as USER-NAME identified by PASSWORD*

- hostname as HOST-NAME instead of HOST-NAME direction
- application as APP-NAME instead of APP-NAME direction
- database as DBS-STRING instead of DBS-STRING

- **DBS-NAME**
- **HOST-NAME**
- **USER-NAME**
- **PASSWORD**
- **APP-NAME**
- **DBS-STRING**

Use **alter partition** to edit partitions in the following ways:

**Keywords**

...**set connect**
Change the user authorized to access the partitioned cubes.
...set hostname
Edit the partition definition to include the correct URL for the partition source cube, target cube, or both.

...set application as
Edit the partition definition to include a corrected application name. This is useful if one application name was changed; if both application names changed, the partition definition cannot be corrected and you must re-create it.

...set database as
Edit the partition definition to include a corrected cube name. This is useful if one cube name was changed; if both cube names changed, the partition definition cannot be corrected and you must re-create it.

...direction single
See Examples 2, 4, and 5.

...direction all
See Example 3.

Notes

• The first DBS-NAME is the local cube, and the second DBS-NAME is the remote cube.

• Directing a partition to the remote site means the current cube is the source. Creating a partition from the remote site means the current cube is the target.

• To change the authorized partition user, you must change the user for both partitioned cubes, as shown in Example 1.

• If a partitioned host, application, or cube is renamed, the rename does not propagate to the partition definition, so you must use alter partition to change the name in the partition definition. As shown in Examples 2 through 5, you must give the old name and the new name. If both names were changed, the partition definition is not recoverable, and must be re-created.

Example

Example 1 (Alter Partition)
The following example changes the user authorized to access the partitioned cubes.

/* To change authorized partition user on target, log in to source & then use: */
alter transparent partition app1.source to app2.target
set connect as newuser identified by newpasswd;

/* To change authorized partition user on source, log in to target & then use: */
alter transparent partition app2.target from app1.source
set connect as newuser identified by newpasswd;
Example 2 (Alter Partition)

In the following example, alter partition is used to fix a partition definition that became invalid when a URL changed and affected only one half of the partition definition (app2.target):

```
alter transparent partition app1.source to app2.target at "https://myEssbase-myDomain.analytics.us1.example.com/essbase/agent"
    set hostname as "https://myEssbase-myDomain.analytics.us2.example.com/essbase/agent" instead of "https://myEssbase-myDomain.analytics.us1.example.com/essbase/agent" direction single;
```

where **direction single** indicates that only the target URL needs to be changed.

Example 3 (Alter Partition)

In the following example, alter partition is used to fix a partition definition that became invalid when a host-name change affected both the source and the target, because both applications were on the same host:

```
alter transparent partition app1.source to app1.target at "https://myEssbase-myDomain.analytics.us2.example.com/essbase/agent"
    set hostname as "https://myEssbase-myDomain.analytics.us2.example.com/essbase/agent" instead of "https://myEssbase-myDomain.analytics.us1.example.com/essbase/agent" direction all;
```

where **direction all** indicates that the host-name change needs to be made on both the target and source halves of the partition definition.

Example 4 (Alter Partition)

In the following example, alter partition is used to fix a partition definition that became invalid when the source application name (oldAppName) changed to newAppName, and affected only one half of the partition definition:

```
alter transparent partition newAppName.source to app2.target
    set application as newAppName instead of oldAppName direction single;
```

where **direction single** indicates that only one half of the partition definition needs to be corrected.

**Note:**

The old application name can be discovered by issuing the **display partition** statement prior to correcting the partition definition.

Example 5 (Alter Partition)

In the following example, alter partition is used to fix a partition definition that became invalid when the source application name (oldAppName) changed to
newAppName, and affected both halves of the partition definition because both partitioned cubes were on the same application:

```sql
alter transparent partition newAppName.source to newAppName.target
    set application as newAppName instead of oldAppName direction all;
```

where `direction single` indicates both halves of the partition definition need to be corrected.

**Alter Session**

Set MDX display options.

**Syntax**

```sql
alter session set dml_output
```

<table>
<thead>
<tr>
<th>Setting</th>
<th>Default</th>
<th>on/off</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>alias</code></td>
<td></td>
<td>on/off</td>
</tr>
<tr>
<td><code>metadata_only</code></td>
<td></td>
<td>on/off</td>
</tr>
<tr>
<td><code>cell_status</code></td>
<td></td>
<td>on/off</td>
</tr>
<tr>
<td><code>numerical_display</code></td>
<td>default</td>
<td>fixed_decimal scientific_notation</td>
</tr>
<tr>
<td><code>precision</code></td>
<td>PRECISION-DIGITS</td>
<td></td>
</tr>
<tr>
<td><code>formatted_value</code></td>
<td></td>
<td>on/off</td>
</tr>
<tr>
<td><code>get_missing_cells</code></td>
<td></td>
<td>on/off</td>
</tr>
<tr>
<td><code>get_null_value</code></td>
<td></td>
<td>on/off</td>
</tr>
<tr>
<td><code>get_meaningless_cells</code></td>
<td></td>
<td>on/off</td>
</tr>
</tbody>
</table>

**PRECISION-DIGITS**

Use `alter session` to change the following MDX output settings:

**Keywords**

- **default**
  Revert to the default MDX display settings in the MaxL Shell. The default settings are: alias ON, metadata_only OFF, cell_status OFF.

- **alias on/off**
  Set whether to use aliases instead of member names.

- **metadata_only on/off**
  Set whether to show only the metadata, with no data.
**cell_status on|off**
Set whether to display cell status. Cell status is additional information returned with each cell value in MDX query outputs.

---

### Note:

Every cell consists of one member from each dimension. Up to four cell-status types may be returned with the output:

- **DC**: Dynamic Calc. If any of the members defining the cell is Dynamic Calc, this status is on.
- **RO**: Read Only. If the cell cannot be written to (for example, by lock-and-send), this status is on. Security filters in the database might cause cells to be read-only. Dynamic Calc cells are automatically read-only.
- **CM**: Calculated Member. If any of the members defining the cell is a calculated member, this status is on.
- **LO**: Linked Object. If the cell has any associated Linked Reporting Objects, this status is on.

---

**numerical_display fixed_decimal|scientific_notation|default**
Set whether MaxL returns data values in MDX query output as fixed decimals, scientific notation, or default format (values are returned in a reasonable combination of decimals or scientific notation).

**precision <precision-digits>**
Set the number (0-15) of decimal places to include for the data values in MDX query output.

**formatted_value on|off**
Set whether to return formatted values for all cells of type text or date, or cells associated with a format string. By default, this setting is on.

**get_missing_cells on|off**
Set whether to return #Missing valued cells for all cells of type text or date, or cells associated with a format string. By default, this setting is on.

**get_meaningless_cells on|off**
Set whether to return #Meaningless for cells that are empty only because they are unassociated with the context attribute or varying attribute. By default, this setting is off, and the empty cells display as #Missing.

The following example query gets sales for all products, but the aggregation is specified by the slicer context only for Ounces_12.

```
SELECT
{Sales, Cogs}
ON COLUMNS,
{Product.Levels(0).Members}
ON ROWS
FROM Sample.Basic
```
WHERE (Ounces_12)
;

A value of #Meaningless is displayed for any members not associated with the attribute Ounces_12.

### Alter System

**Click here for aggregate storage version**

Change the state of Essbase Server. Start and stop applications, change system-wide variables, manage password and login activity, disconnect users, end processes, or shut down the server.

Permission required: Administrator.

**Syntax**

- **APP-NAME**
- **INTEGER**
Use `alter system` to change the following system-wide settings:

**Keywords**

**load application**
Start an application, or start all applications on the Essbase Server.

**unload application**
Stop an application, or stop all applications on the Essbase Server. Unloading an application cancels all active requests and database connections, and stops the application. If Essbase encounters a problem when trying to cancel active requests and database connections, and stopping the application, an error is logged in the application log.

If you do not want to stop an application if it has active requests and database connections, use the **no_force** grammar. When using **no_force**:

- If the application has active requests and database connections, the application is not stopped; it continues running
- If the application does not have active requests and database connections, the application is stopped, as if you used `unload application` without specifying **no_force**

**set session_idle_limit**
Set the interval of time permitted for a session to be inactive before Essbase Server logs off the user. The minimum limit that you can set is five minutes (or 300 seconds).

When the session idle limit is set to **none**, all users can stay logged on until the Essbase Server is shut down.

The default user idle logout time is 60 minutes. When a user initiates a calculation in the background, after 60 minutes the user is considered idle and is logged out, but the calculation continues in the background.

Because the user may mistakenly assume that the calculation stopped because he or she was logged out, you can do one of the following to correct the user experience:

- Run the calculation in the foreground
- Increase the session idle limit in to a time that exceeds the duration of the calculation, or to none

**set session_idle_poll**
Set the time interval for inactivity checking and security-backup refreshing. The time interval specified in the session idle poll gives Essbase instructions:

- Tells it how often to check whether user sessions have passed the allowed inactivity interval indicated by `session_idle_limit` in the `alter system` statement.
- Tells it how often to refresh the security backup file. If `session_idle_poll` is set to zero, the security backup file is still refreshed every five minutes.

**set invalid_login_limit**
Set the number of unsuccessful login attempts allowed by any user before the system disables it. When you change this setting, the counter resets to 0. When the invalid login limit is set to **none**, there is no limit. By default, there is no limit.
set inactive_user_days
Set the number of days a user account may remain inactive before being disabled by the system. The counter resets when the user logs in, is edited, or is activated by an administrator. When the inactive days limit is set to `none`, user accounts remain enabled even if they are not used. By default, there is no limit.

set password_reset_days
Set the number of days users may retain passwords. After the allotted number of days, users are prompted at login to change their passwords. The counter resets for a user when the user changes the password, is edited, or is activated by an administrator. When the password reset days limit is set to `none`, there is no built-in limit for password retention. By default, there is no limit.

set variable
Change the value of an existing substitution variable on the system. The value must not exceed 256 bytes. It may contain any character except a leading ampersand (&).

set server_port
Expand a port range specified Essbase configuration properties. Each Essbase application uses two ports from this range. If no more ports are available, an error message is displayed.

```plaintext
Note:
You can expand port ranges only so that the beginning port range is less than SERVERPORTBEGIN and the ending port range is greater than SERVERPORTEND.
```

delete export_directory
Delete directories created for linked reporting objects exported from a database to a directory created in ARBORPATH\app. Use this grammar after the exported LROs are migrated into a database using `import lro`, and the directories containing the exported LRO information are not needed.

```plaintext
Note:
This process works only for directories created in ARBORPATH\app using the DBS-EXPORT-DIR option of the `export lro` statement. It does not work for directories created elsewhere using the FULL-EXPORT-DIR option of the `export lro` statement.
```

To view a list of names of exported linked-reporting-objects directories in ARBORPATH\app, use `display system export_directory`.

add variable
Create a system-level substitution variable by name, and optionally assign a string value for the variable to represent. You can assign or change the value later using `set variable`. A substitution variable acts as a global placeholder for information that changes regularly. Substitution variables may be referenced by calculations and report scripts.
If substitution variables with the same name exist at server, application, and database levels, the order of precedence for the variables is as follows: a database-level substitution variable supersedes an application-level variable, which supersedes a server-level variable.

**drop variable**
Remove a substitution variable and its corresponding value from the system.

**logout session all**
Terminate all user sessions running on the Essbase Server.

**logout session...force**
Terminate a session (or sessions) even if it is processing a request. The request is allowed to proceed to a safe point, and then the transaction is rolled back.

**logout session <session-id>**
Terminate a session by its unique session ID number. To see the session ID number, use `display session`.

**logout session by user**
Terminate all current sessions by a particular user, either across the entire Essbase Server, or limited to a specific application or database.

**logout session by user on application**
Terminate all current sessions by a particular user across a specific application.

**logout session by user on database**
Terminate all current sessions by a particular user across a specific database.

**logout session on application**
Terminate all current user sessions across a specific application.

**logout session on database**
Terminate all current user sessions across a specific database.

**shutdown**
Shut down the Essbase Server.

**kill request all**
Terminate all current requests on the Essbase Server.

---

**Note:**
To terminate your own active request in MaxL Shell, press the ESC key.

**kill request <session-id>**
Terminate the current request indicated by the session ID. You can obtain session IDs using `display session`.

**kill request by user**
Terminate all current requests by the specified user on the Essbase Server.

**kill request on application**
Terminate all current requests on the specified application.
kill request on database
Terminate all current requests on the specified database.

enable unicode
Set the Essbase Server to allow the creation of Unicode-mode applications and the migration of non-Unicode-mode applications to Unicode-mode applications.

disable unicode
Prevent the Essbase Server from allowing the creation of Unicode-mode applications or the migration of non-Unicode-mode applications to Unicode-mode applications.

reconcile
When Essbase is started using a security backup file (essbase_timestamp.bak) instead of essbase.sec, reconcile the security file to match the state of Essbase on an external disk. This grammar displays discrepancies in application and database information between the security file and the external disk:

- If an application folder is on the disk but not in the security file, display a message indicating the discrepancy. (Essbase checks for the presence of a appname.app file in the ARBORPATH/app/appname directory.)
  The force option does not apply in this scenario.

- If an application file is in the security file but not on the disk, display a message indicating the discrepancy.
  The force option removes the application from the security file.

- If an application database folder is on the disk but not in the security file, display a message indicating the discrepancy. (Essbase checks for the presence of a dbname.otl file in the ARBORPATH/apname/dbname/cube directory.)
  The force option does not apply in this scenario.

- If an application database file is in the security file but not on the disk, display a message indicating the discrepancy.
  The force option removes the database from the security file.

Notes

SESSION SPECIFICATION

A session is a single user connection to Essbase Server. The session can be identified by keywords and names indicating context, or by a unique session ID number.

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. Only one request at a time can be processed in each session.

If a session is processing a request at the time that an administrator attempts to terminate the session, the administrator must either terminate the request first, or use the force keyword available with alter system to terminate the session and the current request.
Example

alter system unload application Sample;

Stop the Sample application, if it is currently running.

alter system unload application all;

Terminates all active requests and stops all applications.

alter system unload application Sample no_force;

Essbase prepares to unload the Sample application; however, if active requests are running, the application is not stopped.

alter system shutdown;

Stops all running applications and shuts down Essbase Server.

alter system logout session by user Fiona;

Disconnects Fiona from any applications or databases to which she is connected.

To log out a user, log out the sessions owned by that user.

alter system set password_reset_days 10;
Specifies that all users will be prompted after 10 days to change their passwords. The day count for any user is reset when the user changes the password or is edited or reactivated by an administrator.

Alter Tablespace (Aggregate Storage)

Change details about a tablespace. To see a list of tablespaces, use display tablespace. You cannot change the location or size of the metadata and log tablespaces.

Tablespaces are applicable only to aggregate storage databases.

Permission required: Application Manager.

Syntax

```
alter tablespace TABLSP-NAME
  | add file_location FILE-NAME
  | alter file_location FILE-NAME
  | drop file_location FILE-NAME
  |
  | set max_file_size SIZE-STRING
  | set max_disk_size SIZE-STRING
```

- TABLSP-NAME
- SIZE-STRING

Use alter tablespace to edit tablespaces in the following ways:

Keywords

**add file_location**
Add a new file location to the tablespace.

> **Note:**

   FILE-NAME is case sensitive in this statement.

**alter file_location**
Change the maximum file-size or disk-size value for the specified file location.

> **Note:**

   FILE-NAME is case sensitive in this statement.
set max_file_size
Specify a value for the maximum size that a data file may attain before Essbase creates a new file.
The largest possible value that the aggregate storage kernel can handle is 134217727 MB. This is also the default value. If operating system limits take effect before this value is reached, the kernel creates a new file. If you enter a value that is larger than 134217727 MB, the kernel ignores the setting and caps file size at 134217727 MB. The minimum value is 8MB (8388608b), and any values you enter are rounded up to the next 8MB interval.

Note:
Some operating system platforms may enforce a maximum file size limit.

set max_disk_size
Specify the value for the maximum amount of disk space to be allocated to the file location.
The largest possible value that the aggregate storage kernel can handle is 4294967295 MB. This is also the default value. If operating system limits take effect before this value is reached, the kernel attempts to use another file location in the tablespace. If you enter a value that is larger than 4294967295 MB, the kernel ignores the setting and caps disk size at 4294967295 MB. The minimum value is 8MB (8388608b), and any values you enter are rounded up to the next 8MB interval.

drop file_location
Delete the specified file location from the tablespace. When a file location is deleted, all files in the file location are deleted, as well as the subdirectory containing the files. You cannot delete a file location if it contains data. You cannot delete the tablespace itself.

Note:
FILE-NAME is case sensitive in this statement.

Notes
• This statement requires the application to be started.
• On Windows, you can specify tablespace file locations using Uniform Naming Convention (UNC) syntax, which is \ComputerName\SharedFolder\Resource. Including the escape characters required by MaxL Shell, the UNC file name specification would look like the following:
  '\\\ComputerName\SharedFolder\Resource'

Example
alter tablespace ASOsamp.'default' add file_location 'C:\mytablespace'
set max_file_size 50mb;
Adds another file location for the default tablespace. Now the tablespace default is in C:\mytablespace in addition to the original location, C:\Hyperion\products\Essbase\EssbaseServer\app.

```
alter tablespace ASOsamp.'default' alter file_location 'C:\\Hyperion\\products\\Essbase\\EssbaseServer\\' set max_file_size 50mb;
```

Changes the maximum file size allowed in the specified location of the default tablespace. Note that the file_location string is case sensitive.

```
alter tablespace ASOsamp.'default' alter file_location '\\\ComputerName\\SharedFolder\\Resource' set max_file_size 50mb;
```

Changes the maximum file size allowed in the specified location of the default tablespace. The file_location string is specified using UNC.

**Alter Trigger**

Enable or disable a trigger created to track state changes over a selected cube area.

**Syntax**

```
 alter trigger TRIGGER-NAME enable
don database DBS-NAME disable
```

- **TRIGGER-NAME**
- **DBS-NAME**

Use `alter trigger` to edit triggers in the following ways:

**Keywords**

**enable**
Essbase monitors the trigger during data load, calculation, or lock and send, and performs the trigger action when the condition is met on the specified cube area.

**disable**
Essbase does not monitor the trigger.

**on database <DBS-NAME> disable**
Disables all triggers in the database. A restart of the application or the database following the disable restores the triggers to the same state as before the disable was
issued (all the triggers disabled using alter trigger on database DBS-NAME disable are re-enabled).

Example

alter trigger Sample.Basic.WatchCosts disable;

alter trigger on database sample.basic disable;

Create Application

Click here for aggregate storage version

Create or re-create an application, either from scratch or as a copy of another application on the same system. See APP-NAME for information on the maximum length of and special characters that are allowed in an application name. Application names are not case-sensitive.

Syntax

```
create [or replace] application APP-NAME [as APP-NAME] [comment COMMENT-STRING] [type nonunicode_mode | unicode_mode]
```

- **APP-NAME**
- **COMMENT-STRING**

Use create application to create an application in the following ways:

Keywords

create application
Create a new application. Application names are not case-sensitive.

create or replace application
Create an application, or replace an existing application of the same name. Application names are not case-sensitive.

...type nonunicode_mode
Create a Non Unicode-mode application. This is also the default if these keywords are omitted.

...type unicode_mode
Create a Unicode-mode application.

create application as
Create an application as a copy of another application. Application names are not case-sensitive.
**comment**
Create an application description (optional). The description can contain up to 80 characters.

**Example**

create application Sample comment 'This is a test application.';

Creates a new application called Sample with an associated comment.

create application Newsamp as Sample;

Creates an application called Newsamp which is a copy of the application Sample.

create or replace application Sample;

Creates an application called Sample. If an application named Sample already exists, it is overwritten.

**Create Calculation**

Create, replace, or copy a stored calculation.

Permissions required:

- Database Manager to create database-level calculations.
- Application Manager to create application-level calculations.

**Syntax**

```plaintext
create calculation CALC-NAME CALC-STRING
or replace calculation CALC-NAME as CALC-NAME
```

- **CALC-NAME**
- **CALC-STRING**

Use **create calculation** to create a calculation in the following ways:

**Keywords**

**create calculation**
Create a calculation script, the body of which is specified by **CALC-STRING**.
create or replace calculation
Create a calculation script, the body of which is specified by CALC-STRING. If a calculation script of that name already exists, it is replaced.

create calculation as
Create a calculation as a copy of another stored calculation.

Notes

• When creating database-level calculations, this statement requires the database to be started.
• A stored calculation can be associated with an application/database, or with an application only. To create an application-level calculation, use two tokens for CALC-NAME. To create a database-level calculation, use three tokens. See CALC-NAME for more details.
• Calculations created using MaxL must be valid. For information about calculation syntax, see Designing and Maintaining Essbase Cubes.

Example

create or replace calculation sample.basic.Accts
'SET UPDATECALC ON;
CALC DIM(Accounts);'
;

Creates a calculation named Accts that is associated with sample.basic.

create calculation sample.basic.Accts2 as app.db.Accts

Creates a calculation named Accts2 on sample.basic that is a copy of another database's calculation named Accts.

Create Database

Create or re-create a database. Optionally create the database as a copy of another database on the same system. See DBS-NAME for information on the maximum length of and special characters that are allowed in a database name. Database names are not case-sensitive.

Permission required: Application Manager. To copy a database, Manager permission on the source database is additionally required.

Syntax
Use `create database` to create a database in the following ways:

Keywords

- `create database`
  - Create a new database. Database names are not case-sensitive.

- `create or replace database`
  - Create a database, or replace an existing database of the same name. Database names are not case-sensitive.

- `create database using non_unique_members`
  - Create a database that supports the use of duplicate member names. Once you have created a database with a duplicate member outline, you cannot convert it back to a unique member outline.
  - For more information about duplicate member names, see Creating and Working With Duplicate Member Outlines in *Designing and Maintaining Essbase Cubes*.

- `create database as`
  - Create a database as a copy of another database. Database names are not case-sensitive.

- `create currency database`
  - Create or replace a database for currency conversion. Linking a currency database to a main database enables you to convert currency values in a database from one currency into another currency.

- `comment`
  - Create a database description (optional). The description can contain up to 80 characters.

Example

```sql
create or replace database Sample.Basic comment 'This is a test.';

create database Sample.New as Sample.Basic;
```

Creates a database called Basic within the Sample application. If a database named Basic within the Sample application already exists, it is overwritten.
Creates a database called New within the Sample application that is a copy of the database Basic within the Sample application.

create currency database Sample.Interntl;

Creates a currency database called Interntl within the Sample application.

Create Drillthrough

Create a drill-through URL within the active cube outline.

For each drillable region of a cube, you can enable drill-through access by means of a URL to Web content hosted on Oracle ERP and EPM applications.

Syntax

```plaintext
create drillthrough URL-NAME from xml_file FILE-NAME
```

```plaintext
on { MEMBER-EXPRESSION }
```

- **URL-NAME**
- **FILE-NAME**
- **MEMBER-EXPRESSION**

Use `create drillthrough` to create a drill-through URL definition in the following ways:

Keywords

- `create drillthrough`
  Create a drill-through URL as metadata.
  The number of drill-through URLs per database is limited to 255.

- `from xml_file`
  Indicate the path to the local URL XML file that defines the link information.
  The URL XML is created by the ERP or EPM application that deployed the cube. The XML contains the drill-through URL display name and a URL enabling the hyperlink from a cell to a Web interface to occur.
  The following is a sample URL XML file:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<foldercontents path="/"
  <resource name="Assets Drill through GL" description=""
type="application/x-hyperion-applicationbuilder-report">
    <name xml:lang="fr">Rapport de ventes</name>
    <name xml:lang="es">Informe de ventas</name>
```
Define the list of drillable regions, using the same member-set calculation language that is used to define security filters. The list of drillable regions must be enclosed in {brackets}. 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.

**level0 only**

Optional: Restrict the URL definition to level-0 data.

**Example**

```plaintext
create drillthrough sample.basic.myURL from xml_file "myfile1.xml" on
{"@Ichildren("Qtr1"), '@Ichildren("Qtr2")}" level0 only;
```

### Create Filter

Create or re-create a database security filter, either from scratch or as a copy of another filter on the same system. Filters control security for database objects. Use grant to assign filters to users and groups.

Minimum permission required: Database Manager.

**Syntax**

- `create filter FILTER-NAME on MEMBER-EXPRESSION as FILTER-NAME`  
- `create filter FILTER-NAME on MEMBER-EXPRESSION no_access`  
- `create filter FILTER-NAME on MEMBER-EXPRESSION read`  
- `create filter FILTER-NAME on MEMBER-EXPRESSION write`  
- `create filter FILTER-NAME on MEMBER-EXPRESSION meta_write`  
- `create filter FILTER-NAME on MEMBER-EXPRESSION definition_only`

**Keywords**

- `FILTER-NAME`
- `MEMBER-EXPRESSION`

Use **create filter** to create a filter in the following ways:

**Keywords**

- `create filter`
  Create a security filter to restrict or permit access to specified database cells.
create or replace filter
Create a security filter or replace an existing security filter of the same name.

create filter ... no_access on <member-expression>
Create a filter blocking access to a specified member combination.

create filter ... read on <member-expression>
Create a filter providing read-only access to a specified member combination.

create filter ... write on <member-expression>
Create a filter providing write access to a specified member combination.

create filter ... meta_read on <member-expression>
Create a filter restricting access to siblings and ancestors of the member expression. In case of a filtering conflict, the MetaRead filtering overrides the other filter permissions. For more information about metadata filtering, see Metadata Filtering.

create or replace filter ... definition_only;
Updates the filter definition while retaining user associations with the filter. If you replace a filter without using definition_only, then the filter must be re-granted to any users to whom it was assigned.

Notes
• Filters created using MaxL must be valid. For information about filter syntax, see Controlling Access to Database Cells Using Security Filters in Designing and Maintaining Essbase Cubes.
• MEMBER-EXPRESSION must be enclosed in single quotation marks. It can be a comma-separated list.

Example
create filter sample.basic.filt1 read on 'Jan, sales', no_access on '@CHILDREN(Qtr2)';

Creates a filter to restrict privileges to Sample.Basic as follows: gives read-only access to the intersection of Jan and sales (sales data for January only); blocks access to children of Qtr2 (April, May, and June).

create or replace filter sample.basic.filt1 read on 'Sales', @ATTRIBUTE(Bottle)';

Creates a filter (or changes an existing filter) to restrict privileges to Sample.Basic as follows: gives read-only access to sales data for products packaged in a bottle (product base dimension members associated with the Bottle attribute member).
Create Location Alias

Create on the database a location alias identifying a host name, database, user name, and password. Location aliases provide a shorthand way of referencing login information for other cubes.

Minimum permission required: Database Manager.

Syntax

```plaintext
create location alias
or replace location alias
LOC-ALIAS-SINGLE from DBS-NAME
LOCATION-ALIAS-NAME

to DBS-NAME at HOST-NAME as USER-NAME identified by PASSWORD
```

- LOC-ALIAS-SINGLE
- LOCATION-ALIAS-NAME
- DBS-NAME
- HOST-NAME
- USER-NAME
- PASSWORD

Use `create location alias` to create a location alias in the following ways:

Keywords

create location alias
Create a location alias, identifying a remote host name, database, user name, and password. The location alias can be used by the @XREF function as an abbreviated login to a remote database.

create or replace location alias
Create a location alias, replacing any existing location alias of the same name on the same database.

...from <dbs-name>
Specify the name of the current database (the database on which the location alias is being created).

...to <dbs-name>
Specify the name of the remote database to log in to.

...at <host-name>
Specify where the remote database resides (using discovery URL).

...as <user-name> identified by <password>
Specify a user name and password with which to log in to the remote database.

Notes

- This statement requires the database to be started.
Location aliases created using MaxL must be valid.

Location aliases are used by the @XREF function for cross-database calculations.

Example

create location alias EasternDB from Sample.Basic to East.Sales at "https://myEssbase-myDomain.analytics.us2.example.com/essbase/agent" as smith identified by 'password';

Creates a location alias called EasternDB on Sample.Basic that represents the following login information:

- remote host = https://myEssbase-myDomain.analytics.us2.example.com/essbase/agent
- application = East
- database = Sales
- user name = Fiona
- password = sunflower

Create Partition

Create or validate a partition definition between two databases.

Permission required: Database Manager at both sites.

Select the type of partition to create:

- transparent
- replicated

Partitions created using MaxL must be valid. To validate a partition, use the validate only clause. For information about partition definitions, see Designing and Maintaining Essbase Cubes.

Create Replicated Partition

Create or validate a replicated partition definition between two cubes. A replicated partition copies a portion of the source (or master) cube to be stored in a target cube. Users can access the target cube as if it were the source. The administrator must periodically refresh the target data from the source data.

Syntax
Use `create replicated partition` to create a partition in the following ways:

**Keywords**

- `create replicated partition`  
  Create a replicated partition. A replicated partition is a copy of a portion of the data source that is stored in the data target.

- `create or replace`  
  Create a partition definition, or replace an existing partition definition.

- `area`  
  Define the partition areas to share with the other cube. Optionally nickname the area using an `area-alias`.

- `to <dbs-name>`  
  Create a partition definition between the current source cube and the second cube (the target).
from <dbs-name>
Create a partition definition between the current target cube and the second cube (the source).

at <host-name>
Specify the discovery URL of the remote cube.

as <user-name> identified by <password>
Provide the name and password of a default partition user who can connect to both cubes. Essbase uses the login information to:

• Transfer data between the source and the target for replicated and transparent partitions. Security filters can be applied to prevent end users from seeing privileged data.

• Synchronize outlines for all partition types.

using <user-name> identified by <password> for creation
Create the partition using a different user than the one being set as the default partition user. This can be useful when you want to specify a read-only user account as the default partition user.

mapped...
Define the member-name mapping for shared sections of both cubes, if member names for sections that map are different in the two cubes.

outline...
Specify the direction in which outline synchronization should proceed, if necessary. The default direction is the same as the data-refresh direction.

update...
Allow or disallow the updating of data in a replicated-type partition target. If you do not specify update allow, by default, the replicated partition cannot be updated.

comment
Create a comment to describe the source half of the partition definition.

remote comment
Create a comment to describe the target half of the partition definition.

validate only
Validate the existing partition definition described by this statement, without actually creating it.

Notes

• Multiple area specifications are allowed, provided they are separated by whitespace. Multiple mappings are allowed, provided they are separated by whitespace. All area aliases used in a mapping should be associated with the target, and the direction of the mapped clause should go from source to target.

• The first DBS-NAME is the local cube, and the second DBS-NAME is the remote cube.

• Creating a partition to the remote site means the current cube is the source. Creating a partition from the remote site means the current cube is the target.

• Aggregate storage cubes can be the target, but not the source, of a replicated partition.
Example

create or replace replicated partition source.source
area 'DimensionA' sourceAreaA
area 'DimensionB' sourceAreaB
to target.target at "https://myEssbase-myDomain.analytics.us2.example.com/essbase/agent"
as admin identified by 'password'
area 'ParentMemberA' targetAreaA
area 'ParentMemberB' targetAreaB
mapped targetAreaA (ChildA) to (Child_a)
mapped targetAreaB (ChildB) to (Child_b)
;

Creates a partition from cube Source to cube Target where the partitioned areas between them are DimensionA and DimensionB on the source, corresponding to ParentMemberA and ParentMemberB (respectively) on the target. Differences in member names between the two partitioned areas are resolved during the partition creation, using the mapped clauses. Area aliases are used after each area specification, so that members can be mapped specifically for each area.

create or replace replicated partition sampeast.east
area '@IDESCENDANTS("Eastern Region"), @IDESCENDANTS(Qtr1)'
to samppart.company at "https://myEssbase-myDomain.analytics.us2.example.com/essbase/agent"
as partitionuser identified by 'password'
area '@IDESCENDANTS(East) @IDESCENDANTS(Qtr1)'
update disallow;

Creates a replicated partition from an area in the source cube, sampeast.east, to an area in the target cube, samppart.company.

create or replace replicated partition sampeast.east
area '@IDESCENDANTS("Eastern Region"), @IDESCENDANTS(Qtr1)'
to samppart.company at "https://myEssbase-myDomain.analytics.us2.example.com/essbase/agent"
as admin identified by 'password'
area '@IDESCENDANTS(East) @IDESCENDANTS(Qtr1)' foo
mapped foo (Year) to (Yr)
update allow validate only;

Validates the syntax of a replicated partition you might want to create. To create the partition after checking validity, simply remove the validate only phrase. For an explanation of foo as used above, see the definition for AREA-ALIAS.
Create Transparent Partition

Create or validate a transparent partition definition between two cubes. A transparent partition allows users to manipulate data that is stored in a target cube as if it were part of the source cube. The remote data is retrieved from the data source each time that users at the data target request it.

Syntax

```
create transparent partition DBS-NAME <area-spec> to DBS-NAME
or replace

at HOST-NAME as USER-NAME identified by PASSWORD
using USER-NAME identified by PASSWORD for creation <area-spec>

mapped globally (MEMBER-NAME) to (MEMBER-NAME)

outline direct reverse comment COMMENT-STRING
remote comment COMMENT-STRING

validate only

<area-spec> ::= area MEMBER-EXPRESSION AREA-ALIAS
```

- DBS-NAME
- HOST-NAME
- USER-NAME
- PASSWORD
- AREA-ALIAS
- MEMBER-NAME
- COMMENT-STRING
- MEMBER-EXPRESSION

Use `create transparent partition` to create a partition in the following ways:

Keywords

`create transparent partition`
Create a transparent partition. A transparent partition enables users to access data from the data source as though it were stored in the data target. The data is, however,
stored at the data source, which can be in another application, in another cube, or on another Essbase instance.

**create or replace partition**
Create a partition definition, or replace an existing partition definition.

**area...**
Define the partition areas to share with the other database. Optionally nickname the area using an **area-alias**.

**to <dbs-name>**
Create a partition definition between the current source cube and the second cube (the target).

**from <dbs-name>**
Create a partition definition between the current target cube and the second cube (the source).

**at <host-name>**
Specify the discovery URL of the remote cube.

**as <user-name> identified by <password>**
Provide the name and password of a default partition user who can connect to both cubes. Essbase uses the login information to:

- Transfer data between the source and the target for replicated and transparent partitions. Security filters can be applied to prevent end users from seeing privileged data.
- Synchronize outlines for all partition types.

**using <user-name> identified by <password> for creation**
Create the partition using a different user than the one being set as the default partition user. This can be useful when you want to specify a read-only user account as the default partition user.

**mapped...**
Define the member-name mapping for shared sections of both cubes, if member names for sections that map are different in the two cubes.

**outline...**
Specify the direction in which outline synchronization should proceed, if necessary. The default direction is the same as the data-refresh direction.

**comment**
Create a comment to describe the source half of the partition definition.

**remote comment**
Create a comment to describe the target half of the partition definition.

**validate only**
Validate the existing partition definition described by this statement, without actually creating it.

**Notes**
- Multiple area specifications are allowed, provided they are separated by whitespace. Multiple mappings are allowed, provided they are separated by...
whitespace. All area aliases used in a mapping should be associated with the target, and the direction of the mapped clause should go from source to target.

- The first DBS-NAME is the local cube, and the second DBS-NAME is the remote cube.
- Creating a partition to the remote site means the current cube is the source. Creating a partition from the remote site means the current cube is the target.
- Aggregate storage cubes can be the source, the target, or the source and target of a transparent partition. Outline synchronization (refresh outline statement) is not currently enabled for partitions that involve aggregate storage cubes.

Example

```maxl
create or replace transparent partition sampeast.east
  area '@CHILDREN("Eastern Region"), @CHILDREN(Qtr1)' sourceArea
  to samppart.company at "https://myEssbase-myDomain.analytics.us2.example.com/essbase/agent"
  as partitionuser identified by 'password'
  area '@CHILDREN(East) @CHILDREN(Qtr1)' targetArea;
```

Creates a transparent partition between the source, sampeast.east, and the target, samppart.company. The partition is defined only for the areas specified by the area aliases sourceArea and targetArea.

```maxl
create or replace transparent partition source.source
  area 'DimensionA' sourceAreaA
  area 'DimensionB' sourceAreaB
  to target.target at "https://myEssbase-myDomain.analytics.us2.example.com/essbase/agent"
  as smith identified by 'password'
  area 'ParentMemberA' targetAreaA
  area 'ParentMemberB' targetAreaB
  mapped targetAreaA (ChildA) to (Child_a)
  mapped targetAreaB (ChildB) to (Child_b)
;
```

Creates a partition from cube Source to cube Target where the partitioned areas between them are DimensionA and DimensionB on the source, corresponding to ParentMemberA and ParentMemberB (respectively) on the target. Differences in member names between the two partitioned areas are resolved during the partition creation, using the mapped clauses. Area aliases are used after each area specification, so that members can be mapped specifically for each area.

Create Trigger

Create or replace a trigger to track state changes over a selected cube area.

Select the type of trigger to create:
• on-update
• after-update

Create After-Update Trigger

Create or replace a trigger to track state changes over a selected cube area.

Triggers help you track whether designated constraints are violated during updates (events) in the area, and allow you to specify resultant actions to execute if violations are detected.

Minimum permission required: Database Manager.

Create an *after-update* trigger if you want the trigger to be activated after the entire data update operation is completed. This is the only type of trigger supported in aggregate storage mode. When after-update triggers are used, the trigger fires when an update operation on level-0 data cells is complete, and the update operation as a whole has met any condition specified for the cube area.

**Note:**
You cannot create or replace a trigger during a calculation, or a data load (including a lock and send).

**Note:**
If a calculation assigns the same value to a given cell as was already present before the calculation, then triggers for that cell will not activate. In other words, if cell values are not changed, blocks are not marked as dirty, and triggers for those blocks are not activated, even if the trigger condition was otherwise met.

Syntax

```
create or replace after update trigger TRIGGER-NAME

where CUBE-AREA when CONDITION then ACTION end
```

• TRIGGER-NAME
• CUBE-AREA
• CONDITION
• ACTION

Use *create after update trigger* to create a trigger in the following ways:
Keywords

**create after update trigger**
Create a new after-update trigger.

**create or replace after update trigger**
Create an after-update trigger, or replace an existing trigger of the same name.

**where <cube area>**
Define the area of the database to be tracked. Use a valid, symmetric MDX slicer specification.

**when <condition>**
Define the condition to be tested for using the keyword WHEN followed by a valid MDX conditional expression.

**then <action>**
Define the action to be taken if the WHEN condition is met. See examples in Examples of Triggers.

**end**
The END keyword must terminate every create trigger statement.

Example

```
create or replace after update trigger Sample.Basic.EastColas
where (Jan, Sales, Actual, [100], East)
when Jan > 20 then spool EastColas_Fail end;
```

Logs a message in the `$ARBORPATH\app\Sample\Basic\trig\EastColas_Fail` file.

Create On-Update Trigger

Create or replace an on-update trigger to track state changes over a selected cube area.

Triggers help you track whether designated constraints are violated during updates (events) in the area, and allow you to specify resultant actions to execute if violations are detected.

Minimum permission required: Database Manager.

An on-update trigger is the default type of trigger, even if no type is specified. During a data update process, any cell update that meets a condition specified for the cube area will immediately activate the trigger. On-update triggers are not supported in aggregate storage databases. If you are using an aggregate storage database, you can create after-update triggers.
Note:
If a calculation assigns the same value to a given cell as was already present before the calculation, then triggers for that cell will not activate. In other words, if cell values are not changed, blocks are not marked as dirty, and triggers for those blocks are not activated, even if the trigger condition was otherwise met.

Note:
You cannot create or replace a trigger during a calculation, or a data load (including a lock and send).

Syntax

```
create [on update] trigger TRIGGER-NAME

or replace on update trigger TRIGGER-NAME

log_value OFF
log_value ON

where CUBE-AREA

when CONDITION then ACTION
else ACTION
```

- TRIGGER-NAME
- CUBE-AREA
- CONDITION
- ACTION

Use `create on update trigger` to create a trigger in the following ways:

Keywords

create [on update] trigger
Create a new on-update trigger. The **on update** keywords are optional; an on-update trigger is created by default.

create or replace [on update] trigger
Create an on-update trigger, or replace an existing trigger of the same name.

log_value OFF
Optional. Log no data values to the trigger spool file. This is the default.

log_value ON
Optional. Log new and old data values to the trigger spool file.
where <cube area>
Define the area of the database to be tracked. Use a valid, symmetric MDX slicer specification.

when <condition>
Define the condition to be tested for using the keyword WHEN followed by a valid MDX conditional expression.

then <action>
Define the action to be taken if the WHEN condition is met. See examples in Examples of Triggers.

else <action>
Optional. Define an action to be taken if the WHEN condition is not met. See examples in Examples of Triggers.

end
The END keyword must terminate every create trigger statement.

Example

create or replace on update trigger Sample.Basic.EastColas
where (Jan, Sales, Actual, [100], East)
when Jan > 20 then spool EastColas_Fail end;

Logs a message in the $ARBORPATH\app\Sample\Basic\trig\EastColas_Fail file.

Display Application

View information about current application-wide settings.

Syntax

display application

APP-NAME

Use display application to display application information in the following ways:

Keywords

all
Display all applications on the system.

<app-name>
Display the named application.
<app-name> message_level
Display the message-level settings for the named application.
Sample output:

<table>
<thead>
<tr>
<th>component</th>
<th>message_level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample</td>
<td>info</td>
</tr>
</tbody>
</table>

**Output Columns**

**application**
String. Name of the application.

**comment**
String. Optional description of the application.

**startup**
TRUE or FALSE. Whether all users who have at least read permission can start the application.

**autostartup**
TRUE or FALSE. Whether the application starts when Essbase Server starts.

**minimum permission**
String. Minimum level of permission all users can have to databases in the application.

**connects**
TRUE or FALSE. Whether any user with a permission lower than Application Manager can make connections to the databases in this application which would require the databases to be started.

**commands**
TRUE or FALSE. Whether users with sufficient permissions can make read requests (or higher) to databases in the application.

**updates**
TRUE or FALSE. Whether users with sufficient permissions can make write requests (or higher) to databases in the application.

**security**
TRUE or FALSE. If FALSE, the Essbase security settings are disabled for the application, and all users are treated as Application Managers.

**lock_timeout**
Number. Maximum time interval (in seconds) that locks on data blocks can be held by clients.

**max_lro_file_size**
Number. If 0, there is no limit on the size of LRO attachments. All other sizes are displayed in kilobytes.
**application_type**
The type of encoding for the application.

0  Unspecified encoding type. The application was created using a pre-Release 7.0 version of Essbase.
1  This value is not in use.
2  Non-Unicode-mode application
3  Unicode-mode application

**application_locale**
The language of the character set in use by the application.

**server**
The name of the computer hosting the Essbase Server.

**application_status**

0  Not Loaded
1  Loading
2  Loaded
3  Unloading

**elapsed_time**
How long the application has been loaded.

**users_connected**
The number of users currently connected to the application.

**storage_type**
The data storage type of the application.

0  Default data storage (same as 1)
1  Block storage (multidimensional)
4  Aggregate storage

**number_of_databases**
The number of databases in the application namespace.

**Example**

display application;

| Displays information about all applications on the system. |

display application Sample;

| Displays information about the Sample application. |

**Display Calculation**

View a list of stored calculations on the system.
Syntax

```plaintext
display calculation

- all
  - CALC-NAME
  - on application APP-NAME
  - on database DBS-NAME
```

- CALC-NAME
- DBS-NAME
- APP-NAME

Use `display calculation` to display calculations in the following ways:

**Keywords**

- `all`
  Display all stored calculations on the system.

- `<calc-name>`
  Display the named calculation.

- `on application` (with or without `APP-NAME`)
  Display all calculations on the specified application.

- `on database` (with or without `DBS-NAME`)
  Display all calculations on the specified database.

**Example**

```
display calculation;
```

**Display Database**

View information about current database-wide state and settings.

**Syntax**

```plaintext
display database

- all
  - request_history
  - DBS-NAME
  - on application APP-NAME
```

- DBS-NAME
- APP-NAME

Use `display database` to display database information in the following ways:
Keywords

**all**
Display information for all databases on the system.

**<dbs-name>**
Display information about the specified database.

**on application**
Display information about all databases on the specified application.

**request_history**
Display information about recent requests for the database. Information about the last three requests is returned.

Output Columns

**application**
Name of the application

**database**
Name of the database

**comment**
Text of the database comment, if present

**startup**
Whether the database is set to start when a user attempts retrievals against it

**autostartup**
Whether the database is set to start when the application starts

**minimum permission**
Minimum permission setting for the database.

**aggregate_missing**
Whether Essbase aggregates missing values during database calculations

**two_pass_calc**
Whether Two-Pass calculation is enabled

**create_blocks**
Whether create blocks on equations is enabled

**data_cache_size**
The size setting of the data cache for holding uncompressed data blocks

**file_cache_size**
The size setting of the file cache

**index_cache_size**
The size setting of the index cache, a buffer in memory that holds index pages

**index_page_size**
The size setting for the index page, a subdivision of an index file that contains index entries that point to data blocks. This setting is not changeable
cache_pinning
Whether cache memory locking is enabled (no longer supported)

compression
Compression type. Field values are numeric, and translate as follows:

1               Run-length encoding
2               Bitmap
3               (no longer supported)

retrieve_buffer_size
The size of the retrieval buffer, used to process and optimize retrievals from grid clients

retrieve_sort_buffer_size
The size of the retrieval sort buffer, used to hold data to be sorted during retrievals

io_access_mode
The current I/O access mode. Only buffered I/O is supported.

pending_io_access_mode
Values are numeric, and translate as follows:

0               Invalid / Error
1               Buffered
2               Direct /* no longer supported

no_wait
Whether Essbase is set to wait to acquire a lock on data blocks that are locked by another transaction

committed_mode
Whether Essbase is set to enable transactions to hold read/write locks on all data blocks involved with a transaction until the transaction completes and commits

pre_image_access
Whether Essbase is set to allow users read-only access to data blocks that are locked for the duration of another concurrent transaction

lock_timeout
The maximum number of minutes that data blocks can be locked by users

commit_blocks
The number of data blocks updated before Essbase performs a commit (The default is 3000)

commit_rows
The number of rows of a data file processed during a data load before Essbase performs a commit (The default is 0)

currency_database
Name of a linked currency database, if one exists
currency_member
The member to use as a default value in currency conversions

currency_conversion
The method of currency conversion.
Values are numeric, and translate as follows:

1 division
2 multiplication

note
Annotation accessible from the login dialog box

db_type
Database type. Values are numeric, and translate as follows:

0 Normal
1 Currency /* no longer supported

read_only_mode
Values are numeric, and translate as follows:

0 Not read only
1 Read only

db_status
Running status of the database. Values are numeric, and translate as follows:

0 Not Loaded
1 Loading
2 Loaded
3 Unloading

elapsed_time
How long the database has been running, in hours:minutes:seconds

users_connected
Number of connected users

blocks_locked
How many data blocks are locked

number_dimensions
Number of dimensions

number_disk_volume
Number of disk volumes
**data_status**
Values are numeric, and translate as follows:

0       No Data
1       Data Loaded without Calculation
2       Data is Calculated

**current_data_cache**
Current size of the data cache

**current_file_cache**
Current size of the file cache

**current_index_cache**
Current size of the index cache

**current_index_page**
Current size of the index page

**currency_country_dim**
For currency databases, the country dimension

**currency_time_dim**
For currency databases, the time dimension

**currency_category_dim**
For currency databases, the accounts dimension where currency categories are defined

**currency_type_dim**
For currency databases, the currency type dimension, which contains members that identify various currency scenarios

**request_type_n / request_user_n / request_start_n / request_end_n**
If you use the request_history keyword, information about the last three requests is returned under columns request_type_n, request_user_n, request_start_n, and request_end_n, where n is 1, 2, and 3. The request_user fields return the names of the users who made the requests. The request_start and request_end fields return the date and time of the requests. request_type field values are numeric, and translate as follows:

0       Data Load
1       Calculation
2       Outline Update
        Unknown

**Example**

display database;
Displays information about all databases on the system.

display database Sample.Basic;

Displays information about the Sample.Basic database.

Display Drillthrough

View drill-through URL definitions used to link to content hosted on Oracle ERP and EPM applications.

Syntax

\[ \text{display drillthrough} \quad \text{DBS-NAME} \quad \text{to FILE-NAME-PREFIX} \quad \text{URL-NAME} \quad \text{to FILE-NAME} \]

- **DBS-NAME**
- **FILE-NAME-PREFIX**
- **URL-NAME**
- **FILE-NAME**

Use **display drillthrough** to display URL information in the following ways:

**Keywords**

- **<dbs-name>**
  Display all drill-through URL definitions on the database. The number of drill-through URLs per database is limited to 255.

- **<dbs-name> to <file-name-prefix>**
  Display all drill-through URL definitions on the database, writing the URL XML content to file names prefixed with the string given as input for FILE-NAME-PREFIX.

- **<url-name>**
  Display the specified drill-through URL definition. 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.

- **<url-name> to <file-name>**
  Display the specified drill-through URL definition, writing the URL XML content to the specified file name.

**Example**

display drillthrough sample.basic;
Displays all drill-through URL definitions on Sample.Basic.

display drillthrough sample.basic to "urlxmls";

Displays all drill-through URL definitions on Sample.Basic, writing the URL XML content to file names prefixed with urlxmls.

display drillthrough sample.basic."Drill through To EPMI";

Displays the drill-through URL definition named Drill through To EPMI.

display drillthrough sample.basic."Drill through To EPMI" to "c:/temp/drillthrough.xml";

Displays the drill-through URL definition named Drill through To EPMI, writing the URL XML content to the file drillthrough.xml.

Display Filter

View a specific filter or a list of all filters on the system.

Syntax

display filter

- all
- FILTER-NAME
- on database DBS-NAME

- FILTER-NAME
- DBS-NAME

Use display filter to display filters in the following ways. Use display filter row to display the contents of filters.

Keywords

- all
  Display all filters on the system.

- <filter-name>
  Display a filter by name.
on database
Display all filters associated with the specified database.

Example

display filter;

Displays the names of all filters on the system.

Display Filter Row

View the filter rows which define database access within a specific filter or all filters.

Syntax

display filter row

- all
- FILTER-NAME
- on database DBS-NAME

- FILTER-NAME
- DBS-NAME

You can display filter contents in the following ways using display filter row.

Keywords

all
Display all filters (and their contents) defined on the system.

<filter-name>
Display a filter and its contents by name.

on database
Display all filters (and their contents) associated with the specified database.

Example

display filter row sample.basic.filt2;

Displays the row-by-row definition of a filter named filt2 which is associated with Sample.Basic.

Display Group

View a specific group or a list of all groups on the system. To view group membership information, use display user.
Syntax

```
  display group  
  all  
  GROUP-NAME
```

**GROUP-NAME**

Use **display group** to display groups in the following ways:

**Keywords**

**all**
Display all security groups on the system.

---

**Note:**
This MaxL grammar is deprecated. Oracle recommends using Java API or Oracle Enterprise Manager to get a list of all groups.

**<group-name>**
Display a security group by name.

**Display Location Alias**

View a specific location alias or a list of all location aliases defined on the system.

**Syntax**

```
  display location alias  
  all  
  LOCATION-ALIAS-NAME  
  on application APP-NAME  
  on database DBS-NAME
```

- **LOCATION-ALIAS-NAME**
- **APP-NAME**
- **DBS-NAME**

You can display location aliases in the following ways using **display location alias**.
Keywords

all
Display all location aliases defined on the system.

<location-alias-name>
Display a location alias by name.

on application
Display all location aliases defined for the specified application.

on database
Display all location aliases defined for the specified database.

Example

display location alias all;

Displays a list of location aliases defined on the system.

Display Lock

View information about locks currently held by users or processes on data blocks.

Note:
Data locks do not apply to aggregate storage applications.

Syntax

display lock

- all
- on system
- on application APP-NAME
- on database DBS-NAME

- APP-NAME
- DBS-NAME

You can display locks in the following ways using display lock.

Keywords

all
Display all locks on the specified scope. If all is omitted, this is the default.
on system
Display all locks on the system.

on application
Display all locks associated with the specified application.

on database
Display all locks associated with the specified database.

Display Object

View a list of database-related file objects stored in database directories.

Syntax

```
display [locked] object
```

<table>
<thead>
<tr>
<th>OBJ-TYPE</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>outline</td>
<td></td>
</tr>
<tr>
<td>calc_file</td>
<td></td>
</tr>
<tr>
<td>report_file</td>
<td></td>
</tr>
<tr>
<td>rules_file</td>
<td></td>
</tr>
<tr>
<td>test</td>
<td></td>
</tr>
<tr>
<td>partition_file</td>
<td></td>
</tr>
<tr>
<td>io</td>
<td></td>
</tr>
<tr>
<td>selection</td>
<td></td>
</tr>
<tr>
<td>wizard</td>
<td></td>
</tr>
<tr>
<td>expl</td>
<td></td>
</tr>
<tr>
<td>outline_paging_file</td>
<td></td>
</tr>
<tr>
<td>worksheet</td>
<td></td>
</tr>
<tr>
<td>alias_table</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OBJ-TYPE ::=</th>
</tr>
</thead>
<tbody>
<tr>
<td>all</td>
</tr>
<tr>
<td>locked</td>
</tr>
<tr>
<td>of type</td>
</tr>
</tbody>
</table>

• APP-NAME
• DBS-NAME
• OBJ-NAME

You can display objects in the following ways using display object.

Keywords

all
Display all stored objects on the specified scope.

locked
Display only locked objects on the specified scope.

of type...
Display only the objects of type specified by OBJ-TYPE ::=.
**OBJ-NAME of OBJ-TYPE**
Display a specific object by name and type.

**on system**
Display all stored objects on the system.

**on application**
Display all objects associated with the specified application.

**on database**
Display all objects associated with the specified database.

Example

```
MAXL> display object sample.basic.Calcdat of type text;
```

<table>
<thead>
<tr>
<th>applicati database</th>
<th>object_na</th>
<th>object_ty</th>
<th>locked</th>
<th>locked_by</th>
<th>locked_time</th>
<th>all</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample</td>
<td>Basic</td>
<td>Calcdat</td>
<td>9</td>
<td>FALSE</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

**Display Partition**

View information about a specific partitioned database or all partitioned databases on the system. Only displays partition information for applications which are currently started.

**Syntax**

```
display partition [DBS-NAME] [options]
```

**DBS-NAME**

You can display partition information in the following ways using `display partition`.

**Keywords**

**all**
Display all partitions defined on the system.

**on database**
Display all partitions associated with the specified database.

**advanced**
Display full information including areas and member mappings for local and remote pieces of partitions.
Notes

If a partition definition is invalid, the same partition may be displayed twice, one time for each half. Each half will show the connection information of the other half.

Example

display partition all;

Displays information about all partitioned databases defined on the system.

Display Privilege

View a list of privileges, calculations, or filters held by users or groups.

Syntax

display privilege

• USER-NAME
• GROUP-NAME

You can display security permissions in the following ways using display privilege.

Keywords

user...
Display security permissions for all users, or for a specified user.

group...
Display security permissions for all groups, or for a specified group.

The values returned for the type field are numeric, and translate as follows:

Table 5-7  Output Columns

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>System-level system privileges (no longer supported in MaxL)</td>
</tr>
<tr>
<td>2</td>
<td>System-level system roles (no longer supported in MaxL)</td>
</tr>
</tbody>
</table>
Table 5-7  (Cont.) Output Columns

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Execute calculation</td>
</tr>
<tr>
<td>4</td>
<td>Filter</td>
</tr>
</tbody>
</table>

Example

display privilege user Fiona;

Displays the privileges user Fiona has on each database object, including any calculations or filters granted to Fiona.

display privilege group;

Displays privileges held by all groups on the system to all applications and databases on the system.

Display Session

View active login sessions on the current server, application, or database, including:

- The user that owns each session
- A session ID for each session
- How long the sessions have been active
- Information about outstanding requests (description, time started, name of computer originating the request, and status).

Syntax

```
display session
```

- all
- SESSION-ID
- by user USER-NAME
  - on application APP-NAME
  - on database DBS-NAME
- on application APP-NAME
- on database DBS-NAME

- APP-NAME
- DBS-NAME
- USER-NAME
- SESSION-ID
You can display login and request information in the following ways using `display session`.

**Keywords**

all
Display information about all current user sessions and active requests.

```
<session-id>
```
Display information about a particular user session, indicated by the numeric session ID.

by user
Display information about all current sessions by a particular user.

by user on application
Display information about all current sessions by a particular user on the specified application.

by user on database
Display information about all current sessions by a particular user on the specified database.

on application
Display information about all current sessions on the specified application.

on database
Display information about all current sessions on the specified database.

### Table 5-8  Display Session: Output Columns

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>user</td>
<td>Logged in user name</td>
<td>powerusr</td>
</tr>
<tr>
<td>session</td>
<td>Numeric session id</td>
<td>865075202</td>
</tr>
<tr>
<td>login_time</td>
<td>Number of seconds ago the session began</td>
<td>192</td>
</tr>
<tr>
<td>application</td>
<td>Name of active application</td>
<td>Sample</td>
</tr>
<tr>
<td>database</td>
<td>Name of active database</td>
<td>Basic</td>
</tr>
<tr>
<td>db_connect_time</td>
<td>Number of seconds ago the database was set active</td>
<td>11879</td>
</tr>
<tr>
<td>request</td>
<td>Type of active request in progress; for example, calculation, data load, or</td>
<td>BuildDimXml :</td>
</tr>
<tr>
<td></td>
<td>restructure. This information can help you get details about what is</td>
<td>Index Only</td>
</tr>
<tr>
<td>request_time</td>
<td>Number of milliseconds the active request has been running</td>
<td>1503869494621</td>
</tr>
<tr>
<td>connection_source</td>
<td>Host name of the connected service</td>
<td>example.com</td>
</tr>
</tbody>
</table>
Table 5-8  (Cont.) Display Session: Output Columns

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>connection_ip</td>
<td>IP address of the connected service</td>
<td>192.0.2.123</td>
</tr>
<tr>
<td>request_state</td>
<td>The status of the active request</td>
<td>in_progress</td>
</tr>
</tbody>
</table>

Example

display session;

display session on database sample.basic;

Display System

View information about current system-wide settings.

Syntax

```maxl
display system
```

**DBS-NAME**

You can display server-wide information in the following ways using `display system`.

Keywords

`display system`
Display current connections and system-wide settings.
configuration field values are numeric, and translate as follows:

1  Non-Unicode mode
2  Unicode mode

display system version
Display the server software version number.

display system ports in use
Display information about ports currently in use on the system.

display system ports overview
Display the number of ports that are available and in use on the system.

display system export_directory
Display names of directories created for linked-reporting objects exported from a
database to a directory created in $ARBORPATH\app If you used export lro and gave a full path to a directory for export files, those
directories are not listed. Only export directories created in the $ARBORPATH\App
directory using the following export lro method are listed:
export database DBS-NAME lro to <server or local> directory DBS-EXPORT-DIR;
where DBS-EXPORT-DIR is a suffix (for example, dir1) for the name of a directory
created by MaxL in $ARBORPATH\App MaxL creates the directory with a prefix of
appname-dbsname-. For example, display system export_directory would list the
following directories existing under $ARBORPATH\App:
sample-basic-dir1
sample-basic-dir2
but it would not list export directories created elsewhere by providing a full directory
path when using the export lro statement, such as: c:\MyExports\MyExportDir

display system license_info
Display information about the license settings implemented on the system.

display system security mode
The type of security in use: native or OPSS mode.
security_mode field values are numeric, and translate as follows:

1  Native Essbase security (no longer supported)
2  OPSS security

display system configuration agent
Display current Essbase Agent configuration properties.
Permission required: Administrator.

display system configuration network
Display current Essbase configuration properties applicable to the network.
Permission required: Administrator.
display system configuration errors
Display Essbase configuration properties that contain errors: an error is any line entry that is not a comment and results in nothing being set. Permission required: Administrator.

display system configuration on database DBS-NAME
Display Essbase configuration properties applicable to the named database. Permission required: Administrator.

message_level
Display the values that are set for the system message level.
Sample output:

<table>
<thead>
<tr>
<th>component</th>
<th>message_level</th>
</tr>
</thead>
<tbody>
<tr>
<td>system</td>
<td>info</td>
</tr>
</tbody>
</table>

Example

display system;

Displays current password and session management settings.

display system configuration agent;

Displays current Essbase configuration properties applicable to the Essbase Agent.

Sample Outputs for Display System Configuration

MAXL> set column_width 40;

MAXL> display system configuration agent;
KEYWORDS                          SETTINGS
+---------------------------------+------------------+
AGENTTHREADS                     50
MAXLOGINS                        100000
PORTUSAGELOGINTERVAL            600

OK/INFO - 1241044 - Records returned: [3].

MAXL> display system configuration network;

KEYWORDS                          SETTINGS
+---------------------------------+------------------+
NETDELAY                          1500
NETRETRYCOUNT                    2000

OK/INFO - 1241044 - Records returned: [2].
MAXL> `display system configuration on database democfg.basic;`

<table>
<thead>
<tr>
<th>KEYWORDS</th>
<th>SETTINGS</th>
</tr>
</thead>
<tbody>
<tr>
<td>+----------------------</td>
<td>---------</td>
</tr>
<tr>
<td>CALCCACHE</td>
<td>TRUE</td>
</tr>
<tr>
<td>CALCCACHEDEFAULT</td>
<td>1250000</td>
</tr>
<tr>
<td>CALCCACHEHIGH</td>
<td>1750000</td>
</tr>
<tr>
<td>CALCCACHELOW</td>
<td>40000</td>
</tr>
<tr>
<td>DLSINGLETHREADPERSTAGE</td>
<td>FALSE</td>
</tr>
<tr>
<td>DLTHREADSPREPARE</td>
<td>4</td>
</tr>
<tr>
<td>DLTHREADSWRITE</td>
<td>4</td>
</tr>
<tr>
<td>DYNCALCCACHEMAXSIZE</td>
<td>DB[41943040], SV[41943040]</td>
</tr>
<tr>
<td>SSPROCROWLIMIT</td>
<td>250000</td>
</tr>
</tbody>
</table>

OK/INFO - 1241044 - Records returned: [9].

Display Trigger

View details about a trigger created to track state changes over a selected cube area.

**Note:**
The application containing the trigger must be started in order to use display trigger.

**Syntax**

```bash
display trigger [all | on system | on application APP-NAME | on database DBS-NAME | TRIGGER-NAME]
```

**APP-NAME**

**Table 5-9  Output Columns**

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>application</td>
<td>The name of the application that contains the database.</td>
</tr>
<tr>
<td>database</td>
<td>The name of the database that contains the trigger. Essbase lists only databases that contain triggers.</td>
</tr>
</tbody>
</table>
Table 5-9  (Cont.) Output Columns

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name of the trigger.</td>
</tr>
<tr>
<td>definition</td>
<td>The MaxL trigger statement (for example, create or replace trigger)</td>
</tr>
<tr>
<td>enabled</td>
<td>Whether Essbase is set to monitor the trigger. Values: TRUE or FALSE. To change the value, use alter trigger.</td>
</tr>
</tbody>
</table>

Example

display trigger on database Sample.Basic;

This example displays the output columns:

Table 5-10  Display Trigger MaxL Output

<table>
<thead>
<tr>
<th>application</th>
<th>database</th>
<th>name</th>
<th>definition</th>
<th>enabled</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample</td>
<td>Basic</td>
<td>WatchCosts</td>
<td>create or replace trigger</td>
<td>TRUE</td>
</tr>
</tbody>
</table>

Display Trigger Spool

View the log file created by a trigger. Triggers track state changes over a selected cube area. For more information about triggers, see Examples of Triggers.

Syntax

display trigger_spool  [all | on application APP-NAME | on database DBS-NAME | SPOOL-NAME]

• APP-NAME
• DBS-NAME
• SPOOL-NAME

Display User

View a specific user or a list of all users defined on the system. View account and group membership information.
Syntax

display user

• USER-NAME
• GROUP-NAME

You can display user information in the following ways using display user.

Keywords

all
Display information about all users on the system.

• USER-NAME
• GROUP-NAME

Note:
This MaxL grammar is deprecated. Oracle recommends using Java API or Oracle Enterprise Manager to get a list of all users.

<user-name>
Display information about the specified user.

in group all
Display membership information for all groups on the system.

in group <group-name>
Display membership information for the specified group.

Output Columns

user
String. Name of the user.

description
No longer supported.

logged in
Values: TRUE or FALSE.

password_reset_days
Integer. The number of days before the password expires, or 0 if no expiration is set.
**enabled**
Values: TRUE if the user account is active, or FALSE if the account has been disabled by an administrator.

**change_password**
Values: TRUE if the user must change the password at the next login; FALSE otherwise.

**type**
Values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>User is set up using native Essbase security (no longer supported)</td>
</tr>
<tr>
<td>1</td>
<td>No longer used.</td>
</tr>
<tr>
<td>3</td>
<td>User is externally authenticated.</td>
</tr>
</tbody>
</table>

**protocol**
If the user is externally authenticated, this field contains the value OPSS. This field is blank if the type field is 0 (the user is not externally authenticated).

**conn param**
This field is blank.

**application_access_type**
Values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No access</td>
</tr>
<tr>
<td>1</td>
<td>Essbase access</td>
</tr>
<tr>
<td>2</td>
<td>Planning access</td>
</tr>
<tr>
<td>3</td>
<td>Essbase and Planning access (requires 2 licenses)</td>
</tr>
</tbody>
</table>

See also Descriptions section.

**Example**

display user;

Displays all users on the system and shows whether they are logged in, whether their accounts are enabled, and whether their passwords are set to expire.

display user in group;

Displays the membership information of all groups on the system.

display user in group big_group;

Displays the membership information for a group called big_group.
Display Variable

View a list of substitution variables defined on the system.

Syntax

```
display variable [all] [VARIABLE-NAME] [on application APP-NAME] [on database DBS-NAME] [on system]
```

- VARIABLE-NAME
- APP-NAME
- DBS-NAME

You can display substitution variables in the following ways using `display variable`.

Keywords

- all
  Display all substitution variables defined on the Essbase Server, including those associated with applications and databases.

- `<variable-name>`
  Display a substitution variable by name.
  Permission required:
  - Read access for the applicable database or application.
  - Administrator for system-defined variables.

on application

Display only substitution variables defined on the specified application.
Permission required: Read access for the application.

on database

Display only substitution variables defined on the specified database.
Permission required: Read access for the database.

on system

Display only the substitution variables associated with the Essbase Server.
Permission required: Administrator.

Notes

To manage substitution variables, use `alter database` (containing add, drop, and set variable).
Example

display variable;

Displays a list of all substitution variables on the Essbase Server.

Drop Application

Delete an empty application from the system. To remove an application with databases, use **cascade**. To remove an application that has locked objects in a constituent database, you can use **force**.

Minimum permission required: Application Manager.

**Syntax**

```
drop application APP-NAME
```

**APP-NAME**

You can delete applications in the following ways using **drop application**.

**Keywords**

**cascade**
Delete an application along with its constituent databases.

**force**
Delete an application that may have locked objects in a constituent database.

Drop Calculation

Delete a stored calculation from a database.

Minimum permission required: Database Manager.

**Syntax**

You can delete calculations using **drop calculation**.

```
drop calculation CALC-NAME
```

**CALC-NAME**
Keywords

drop calculation <calc-name>
Delete the specified calculation.

Example

drop calculation Sample.basic.calcname;

Deletes a calculation from Sample.basic.

Drop Database

Delete a database from the system. If the database has outstanding locks, clear them first, or use force to drop with locks.

Minimum permission required: Database Manager.

Syntax

You can delete databases using drop database.

```
> drop database DBS-NAME
```

DBS-NAME

Keywords

force
Delete a database that may have locked objects.

Example

drop database Sample.Basic force;

Deletes the database Sample.Basic, even if client users have outstanding locks on Sample.Basic.

Drop Drillthrough

Delete a drill-through URL definition used to link to content hosted on Oracle ERP and EPM applications.

Syntax

```
> drop drillthrough URL-NAME
```
URL-NAME

Example

drop drillthrough sample.basic.myURL;

Drop Filter

Delete a security filter from the database.
Minimum permission required: Database Manager.

Syntax

drop filter FILTER-NAME

FILTER-NAME

You can delete filters using drop filter.

Keywords

drop filter <filter-name>
Delete a filter by name.

Example

drop filter sample.basic.filter1;

Deletes the filter called filter1 from the sample.basic database.

Drop Location Alias

Delete from the database a location alias identifying a host name, application, database, user name, and password.
Minimum permission required: Database Manager.

Syntax

drop location alias LOCATION-Alias-NAME

LOCATION-Alias-NAME

You can delete location aliases using drop location alias.
Keywords

```markdown
**drop location alias <location-alias-name>**
Delete a location-alias definition.
```

Example

```sql
drop location alias Main.Sales.EasternDB;
```

Drops the location alias called EasternDB in the Main.Sales database.

### Drop Lock

Remove locks acquired through a grid client operation.

**Note:**

Data locks do not apply to aggregate storage applications.

### Syntax

```sql
drop lock [on system | on application APP-NAME | on database DBS-NAME] [all] [held by USER-NAME]
```

- **APP-NAME**
- **USER-NAME**
- **DBS-NAME**

Keywords

```markdown
**drop lock on system all**
Drops all locks by all users, for all databases on the system.
```

```markdown
**drop lock all**
Same as "drop lock on system all"
```

```markdown
**drop lock on system**
Same as "drop lock on system all"
```

```markdown
**drop lock**
Same as "drop lock on system all"
```
**drop lock on application APP-NAME**
Drops all locks on the application, for all users.

**drop lock on application APP-NAME held by USER-NAME**
Drops locks on the application which are held by a specific user.

**drop lock on database DBS-NAME**
Drops all locks on the database, for all users.

**drop lock on database DBS-NAME held by USER-NAME**
Drops locks on the database which are held by a specific user.

**drop lock held by USER-NAME**
Drops all locks held by a specific user, on any application or database.

**Drop Object**
Remove database-related file objects stored in database directories.

**Syntax**

```
drop object OBJ-NAME of type <OBJ-TYPE> force
```

**Keywords**

- calc_script
- report_file
- rules_file
- text
- lro
- selection
- wizard
- eqp
- outline_paging_file
- worksheet
- alias_table

**OBJ-NAME**

**Keywords**

**...force**
If the object is locked by a user or process, unlock it and delete it.

**Notes**
To drop a partition, use **drop partition**.

**Drop Partition**
Delete from the system a partition definition between two cubes. Database Manager permission for each cube is required.
Syntax

```
drop [transparent] replicated partition DBS-NAME [from] DBS-NAME
at HOST-NAME [force]
```

- **DBS-NAME**
- **HOST-NAME**

You can delete partition definitions in the following ways using `drop partition`.

**Keywords**

- **drop...partition...from**
  Remove a partition definition between the current target cube and a source cube.

- **drop...partition...to**
  Remove a partition definition between the current source cube and a target cube.

- **at <host-name>**
  Optionally specify the host location, if removing a partition definition associated with a remote instance.
  Use the discovery URL to indicate the location. For example, "https://myEssbase-myDomain.analytics.us2.example.com/essbase/agent".

- **force**
  Specify that the source half of a partition definition should be dropped regardless of whether the target half is missing or invalid. For more information, see Forcing Deletion of Partitions.

**Notes**

If the `create partition` statement used was of the format:

```
create partition SOURCE to TARGET;
```

Then the only permutations of the `drop partition` statement that will have effect are:

```
drop partition SOURCE to TARGET;
drop partition TARGET from SOURCE;
```

**Example**

```sql
create or replace replicated partition sampeast.east area
'@DESCENDANTS("Eastern Region"), @DESCENDANTS(Qtr1)' to samppart.company
at "https://myEssbase-myDomain.analytics.us2.example.com/essbase/agent";

drop replicated partition Samppart.Company from Sampeast.East;
```
Drop Trigger

Remove a trigger created to track state changes over a selected cube area.

Syntax

\[ \text{drop trigger TRIGGER-NAME} \]

Example

drop trigger Sample.Basic.WatchCosts ;

Drop Trigger Spool

Delete the log file created by a trigger. Triggers track state changes over a selected cube area. For more information about triggers, see Examples of Triggers.

Syntax

\[ \text{drop trigger spool SPOOL-NAME [all on database DBS-NAME]} \]

• SPOOL-NAME
• DBS-NAME

Execute Calculation

Click here for aggregate storage version

Execute a stored calculation, the stored default calculation (determined by alter database), or an anonymous (non-stored) calculation string.

Minimum permissions required:

• For stored calculations (CALC-NAME): Granted access to the calculation.
• For anonymous calculations (CALC-STRING) and the default calculation: Execute

Syntax

\[ \text{execute calculation CALC-NAME [on database DBS-STRING with runtime subvars RSV-LIST] on DBS-NAME [default]} \]
You can run calculations in the following ways using execute calculation.

**Keywords**

- **execute calculation `<calc-name>`**
  Run the specified stored calculation script.

- **<calc-name> on database**
  Run the specified stored calculation script against the specified database.

- **<calc-string> on <dbs-name>**
  Run an anonymous calculation, whose body is contained in `<calc-string>`, against the specified database.

- **default on <dbs-name>**
  Run the default calculation against the specified database.

- **<calc-name> with runtimesubvars <rtsv-list>**
  Run the specified stored calculation script with the runtime substitution variables specified in `RTSV-LIST`, which is a string of runtime substitution variables specified as key/value pairs. The string must be enclosed with single quotation marks, and the key/value pairs must be separated by a semicolon, including a semicolon after the last runtime substitution variable in the string and before the terminal single quotation mark. In this example of a runtime substitution variable string, the name and value of four runtime substitution variables are specified (for example, the value of the runtime substitution variable named “a” is 100):

  `'a=100;b=@CHILDREN("100");c="Actual"->"Final";d="New York";'`

  The string of runtime substitution variables cannot exceed 64 KB.
Note:

Runtime substitution variables used in a calculation script must be declared in the SET RUNITMESUBVARS calculation command, with a name and default value. If a different value is declared in the RTSV-LIST, the default value is overwritten at runtime. If you include a runtime substitution variable in RSV-LIST that has not been declared in SET RUNITMESUBVARS, Essbase ignores the undeclared runtime substitution variable (no warnings or exceptions are generated). Runtime substitution variables that are used in a calculation script can be logged in the application log file, using the ENABLERTSLOGGING configuration setting. See “Logging Runtime Substitution Variables” in the Designing and Maintaining Essbase Cubes. If the name of a runtime substitution variable that is declared in the SET RUNITMESUBVARS calculation command is the same as a runtime substitution variable declared in RSV-LIST, the value specified in RSV-LIST overwrites the default value in SET RUNITMESUBVARS.

Notes

- A stored calculation can be associated with a specific database in an application (database level), or with an application only (application level). To execute a calculation stored at the application level, you must specify which database in the application to calculate using the on database STRING grammar.
- A calculation script can reference runtime substitution variables using the with runtimesubvars grammar.

Example

execute calculation Sample.Basic.Calc1;

Calculates the Sample.Basic database using the stored calculation script file named Calc1, which is associated with the database.

execute calculation Sample.Calc2 on database Basic;

Calculates the Sample.Basic database using the stored calculation script file named Calc2, which is associated with the Sample application.

execute calculation
'SET MSG ERROR;
CALC ALL;'
on Sample.basic;
Calculates the Sample.Basic database using an anonymous (unstored) calculation string.

execute calculation Sample.Basic.Calc3 with runtimesubvars ‘a=100;b=50;’;

Calculates the Sample.Basic database using the stored calculation script file named Calc3, which is associated with the database, and the specified runtime substitution variables, in which the value of the runtime substitution variable named "a" is 100 and the value of "b" is 50.

Related Topics
• SET RUNTIMESUBVARS
• ENABLERTSVLOGGING

Execute Aggregate Process (Aggregate Storage)

Perform an aggregation, optionally specifying the maximum disk space for the resulting files, and optionally basing the view selection on user querying patterns.

This statement is applies to aggregate storage databases only.

This statement enables you to build aggregate views with a minimum of settings. If greater control is needed, you can combine the following statements:
• Execute Aggregate Selection
• Execute Aggregate Build

This statement causes Essbase to:
1. Select 0 or more aggregate views based on the stopping value and/or on querying patterns, if given.
2. Build the views that were selected.

For more information about aggregate views, see Designing and Maintaining Essbase Cubes.

Syntax

execute aggregate process on database DBS-NAME

| stopping when total_size exceeds STOPPING-VAL based on query_data |
| enable alternate_rollups | disable |

• DBS-NAME
• STOPPING-VAL
You can aggregate an aggregate storage database in the following ways using **execute aggregate process**.

**Keywords**

**stopping when total_size exceeds...**
Aggregate whichever views Essbase selects, with the exception that the maximum growth of the aggregated database must not exceed the given ratio. For example, if the size of a database is 1 GB, specifying the total size as 1.2 means that the size of the resulting data cannot exceed 20% of 1 GB, for a total size of 1.2 GB.

**based on query_data**
Aggregate whichever views Essbase selects, based on collected user querying patterns. This option is only available if query tracking is turned on, using `alter database` with the **enable query_tracking** grammar.

**enable|disable alternate_rollups**
If enabled, secondary hierarchies (with default level usage) are considered for view selection. Default: disabled (no secondary hierarchies are considered).

**Notes**

- View selection (step 1) can be performed independently of aggregation by using `execute aggregate selection`. Aggregation (step 2) can be performed without built-in view selection by using `execute aggregate build`.
- For small databases, the performance of building aggregate views in Essbase 9.3.1 and later versions may be slower than Essbase versions earlier than 9.3.1. However, Essbase 9.3.1 should perform better for databases larger than a few hundred million cells, especially on computers with more than two processors and where the CALCPARALLEL configuration setting has been chosen appropriately.

**Example**

```maxl
execute aggregate process on database ASOsamp.Sample
stopping when total_size exceeds 1.3;
```

Selects and builds an aggregation of the ASOsamp.Sample database that permits the database to grow by no more than 30% as a result of the aggregation.

```maxl
execute aggregate process on database ASOsamp.Sample based on query_data;
```

Selects and builds an aggregation of the ASOsamp.Sample database, where the views that Essbase selects for aggregation are based on the most frequently queried areas of the database.

**Related Topics**

- Execute Aggregate Build
- Execute Aggregate Selection
**Execute Aggregate Build**

Performs an aggregation based on the views selected by the `execute aggregate selection` statement.

The views to build must either be identified by their view IDs, obtained previously using `execute aggregate selection`, or by a view selection saved in an aggregation script.

For more information about aggregate views, see *Designing and Maintaining Essbase Cubes*.

**Syntax**

```
execute aggregate build on database DBS-NAME

using views VIEW-ID VIEW-SIZE with outline_id OUTLINE-ID

view_file VIEW-FILE-NAME
```

- **DBS-NAME**
- **VIEW-ID**
- **VIEW-SIZE**
- **OUTLINE-ID**
- **VIEW-FILE-NAME**

You can materialize aggregations in the following ways using `execute aggregate build`.

**Keywords**

**using views**...  
Builds an aggregation based on a previously selected view (or views) and the associated outline ID.

**using view_file**...  
Builds an aggregation based on a saved view selection stored in an aggregation script.  
Omit the `.csc` file extension from the view file name when you issue the `execute aggregate build` statement.

**Notes**

- Although it is possible to pass arbitrary view-id and view-size arguments, this practice is not supported.
- Passing view-size arguments other than those returned by the `execute aggregate selection` command may cause unpredictable results.
- For small databases, the performance of building aggregate views in Essbase 9.3.1 and later versions may be slower than Essbase versions earlier than 9.3.1. However, Essbase 9.3.1 should perform better for databases larger than a few
hundred million cells, especially on computers with more than two processors and
where the CALCPARALLEL configuration setting has been chosen appropriately.

Example

execute aggregate build on database Sample.Basic using views 711 0.00375
with outline_ID 4142187876;

Builds an aggregation of the Sample.Basic database. The build is based on the view of an aggregate storage outline
(identified as 4142187876) having the view ID 711, and a view
size of 0.00375.

execute aggregate build on database Sample.Basic using view_file myView;

Builds an aggregation of the Sample.Basic database based on the view saved in the aggregation script myView.csc.

Related Topics
• Execute Aggregate Process (Aggregate Storage)
• Execute Aggregate Selection

Execute Aggregate Selection

Select views of an aggregate storage database based on various selection criteria, and return the results in the form of a table or aggregation script. Next, use the tabular information or aggregation script to build an aggregation (materialize a view) using execute aggregate build.

Note:
View selection and aggregation can be performed by Essbase in a single step by using execute aggregate process. However, the use of the two separate statements execute aggregate selection and execute aggregate build enables you more control of the selection criteria.

For more information about aggregate views, see Designing and Maintaining Essbase Cubes.

Syntax
You can select views in the following ways using **execute aggregate selection**.

**Keywords**

- **using views...with outline_ID**
  Selects views based on pre-selected view IDs. The view IDs are obtained from previous executions of the statement.

- **using views...with outline_ID...force display**
  Selects views based on pre-selected view IDs, including the pre-selected views IDs themselves.

- **using views...with outline_ID...suppress display**
  Selects views based on pre-selected view IDs, skipping the pre-selected views IDs themselves. This is the default behavior even if the **suppress** keyword is omitted.

- **selecting <INTEGER> views**
  Selects the number of views based on whether the number of views specified in **<INTEGER>** is greater than or equal to, or less than, the recommended number of default views that are returned by the **execute aggregate selection** statement. By default, Essbase determines the recommended number of default views.

  Assume that **<RECNUM>** represents the recommended number of default views:

  - If the value of **<INTEGER>** is greater than or equal to the value of **<RECNUM>**, the selected number of views equals **<RECNUM>**.
  - For example, if **<INTEGER>** equals 20 and **<RECNUM>** equals 15, the number of selected number of views equals 15.
• If the value of <INTEGER> is less than the value of <RECNUM>, the number of views that are selected equals <INTEGER>. If you want the number of views that are selected to equal the value of <INTEGER>, use the stopping when total_size exceeds <STOPPING-VAL> grammar to change the number of recommended default views that are returned by the execute aggregate selection statement. Define the <STOPPING-VAL> factor large enough so that the number of default views that are returned by execute aggregate selection is greater than the value of <INTEGER>. For example, if <INTEGER> equals 20 and <RECNUM> equals 50, the number of selected number of views equals 20.

Note:
This parameter does not create views.

stopping when total_size exceeds <STOPPING-VAL>
Selects views, specifying a storage stopping value in terms of a factor times the size of the unaggregated input (level 0) values. For example, a stopping value of 1.5 means that the view selection should permit the database to grow by no more than 50% as a result of the aggregation.

based on query_data
Selects views based on previously collected query-tracking data. You must have already enabled query tracking. After enabling query tracking, allow sufficient time to collect user data-retrieval patterns before performing an aggregate selection based on query data. Query tracking records information about every query executed on the database, so that it can be used as a basis for view selection. Query-based view selection helps to improve query performance when the distribution of user queries is skewed. For every level combination, the cost of retrieving cells is recorded. The recording continues until the application is shut down or until the recording is explicitly turned off using alter database <dbs-name> disable query_tracking. In both cases, all the query cost data is discarded, and the recording stops (and will not continue when the application starts again). All query cost data becomes invalid when additional views are built.
To create views based on tracked query patterns,
1. Enable query tracking using alter database <dbs-name> enable query_tracking.
2. Run all production queries once, and then select the first set of views based on the query cost data. To select the views, run this MaxL statement (execute aggregate selection...based on query_data...).
3. Build the selected aggregate view using execute aggregate build.
4. Repeat the previous two steps at least twice. Selecting and building multiple views iteratively helps ensure there are enough usage-tracking data to form a pattern. Each new view you build decreases the rate at which query costs grow.

dump to view_file
Saves the view selection to an aggregation script. If the specified script name already exists, an error is returned. To overwrite an existing script, use the force_dump keyword.
The aggregation script contains information derived during the aggregate view selection. You can materialize the aggregation at a different time by running the aggregation script. For example:

```
execute aggregate build on database <dbs-name>
using view_file <view-file-name>
```

**force_dump to view_file**
Saves the view selection to an aggregation script. If the specified script name already exists, the `force_dump` keyword causes it to be overwritten.

**enable|disable alternate_rollups**
If enabled, secondary hierarchies (with default level usage) are considered for view selection. Default: disabled (no secondary hierarchies are considered).

**Example**

```
execute aggregate selection on database ASOsamp.Sample;
```

Performs the default view selection for ASOsamp Sample. This statement selects the same views as `execute aggregate process on database ASOsamp.Sample` would build.

```
execute aggregate selection on database ASOsamp.Sample using views 711, 8941 with outline_ID 4142187876;
```

Selects views based on the pre-selected view IDs. The view IDs are obtained from previous executions of the statement.

```
execute aggregate selection on database ASOsamp.Sample using views 711, 8941 with outline_ID 4142187876 force display;
```

Selects views based on the pre-selected view IDs. `force display` is used to include the pre-selected views (711 and 8941) in the new selection.

```
execute aggregate selection on database ASOsamp.Sample stopping when total_size exceeds 1.2;
```

Selects an aggregation of the ASOsamp Sample database that, when built, would permit the database to grow by no more than 20% as a result of the aggregation.

```
execute aggregate selection on database ASOsamp.Sample based on query_data;
```
Selects views based on previously collected query-tracking data. You must have enabled query tracking using `alter database <dbs-name> enable query_tracking`.

```plaintext
execute aggregate selection on database ASOsamp.Sample dump to view_file myView;
```

Selects a default aggregation of the ASOsamp Sample database, saving the selection to `APP\DB\myView.csc`. You can materialize the view later by running the aggregation script `myView.csc`. For example:

```plaintext
execute aggregate build on database ASOsamp.Sample using view_file 'myView.csc';
```

Related Topics

- Execute Aggregate Build
- Execute Aggregate Process (Aggregate Storage)

Export Data

Click here for aggregate storage version

Export all data, level-0 data, or input-level data, which does not include calculated values. Export data files are written to the application/cube directory. To use Report Writer, export the data using a report file.

Minimum permission required: Read.

**Syntax**

```plaintext
export database DBS-NAME
```

```plaintext
data anonymous in columns to server
```

```plaintext
using report file FILE-NAME to data_file FILE-NAME
```

- **DBS-NAME**
- **FILE-NAME**

You can export data from a database in the following ways using `export data`.
Keywords

**export database <dbs-name> all data...**
Export all data in the specified cube to the application/cube directory.

**Note:**
Exporting data does not clear the data from the database.

**export database <dbs-name> level0 data...**
Export level-0 data blocks only (blocks containing only level-0 sparse member combinations. Note that these blocks may contain data for upper level dense dimension members.) A level-0 block is created for sparse member combinations when all of the members of the sparse combination are at the bottom of dimension branches.

**Note:**
Exporting data does not clear the data from the cube.

**export database <dbs-name> input data...**
Export only blocks of data where the block contains at least one data value that was loaded (imported), rather than created as the result of a calculation.

**export database <dbs-name> ... data in columns**
Export data in columns, to facilitate loading the exported data into a relational database. In each row, the columnar format displays a member name from every dimension. Names can be repeated from row to row. Columnar format provides a structure to the exported data, so that it can be used for further data processing by applications other than Essbase tools. In non-columnar format, sparse members identifying a data block are included only once for the block. Because the export file in non-columnar format is smaller than in columnar format, reloading a file in non-columnar format is faster.

**export database <dbs-name> ... data anonymous**
Export data in anonymized format. Anonymization removes the risk of sensitive data disclosure, and can be used in case sample data needs to be provided for technical support. Essbase replaces real data values with incremental values beginning with 0, increasing by 1 for each value in the block.

**export database <dbs-name> ...using...report_file...**
Run a stored report script, exporting a subset of the database.

Notes

- This statement requires the database to be started.
- To export data in parallel, specify a comma-separated list of export files, up to a maximum of 1024 file names. The number of file names determines the number of export threads. The number of available block-address ranges limits the number of export threads that Essbase actually uses. Essbase divides the number of actual data blocks by the specified number of file names (export threads). If there are
fewer actual data blocks than the specified number of export threads, the number of export threads that are created is based on the number of actual data blocks. For example, if the block storage database is very small, with only 100 data blocks, Essbase will use only 100 threads, even if you specify a higher number. This approach results in a more even distribution of data blocks between export threads.

**Note:**

In specifying the number of export files, it is important to consider the number of available CPU cores and I/O bandwidth on the computer on which Essbase Server runs. Specifying too large a number can result in poor performance.

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 exportfile.txt, the next additional file is exportfile_1.txt.

- To export data in column format, use the optional "in columns" grammar.
- During a data export, the export process allows users to connect and perform read-only operations.
- When MaxL exports data from a Unicode-mode application, the export file is encoded in UTF-8. You cannot use UTF-8-encoded export files from a Unicode-mode application to import data to a non-Unicode-mode application.
- MaxL cannot export databases with names containing hyphens (-).

**Example**

export database Sample.Basic data to server data_file 'myfilesamp.txt'

Exports data to Sample/Basic/myfilesamp.txt.

**Export LRO**

Export linked-reporting-object information, and binary files if the database has file-type LROs, to a directory.

**Syntax**

```
export database DBS-NAME lro to directory DBS-EXPORT-DIR
```
You can export LRO information from a database in the following ways using `export lro`.

**Keywords**

- **to server directory**
  Export the LRO information to a directory you specify on the Essbase Server to which you are connected.

- **to local directory**
  Export the LRO information to a directory you specify.

**Notes**

- This statement requires the database to be started.
- MaxL creates exactly one export directory; it does not create a directory structure. For example, if `c:\temp` exists, MaxL will create `c:\temp\exports`, but not `c:\temp\exports\to\this\long\path`.
- If the specified export directory already exists, the export LRO statement will fail. This is a safeguard against overwriting existing export directories.
- If you do not specify a full path for an export directory to be created on the client or server, MaxL uses your short directory specification (DBS-EXPORT-DIR) as a suffix, and creates the destination export-directory in the `ARBORPATH\app` directory with a prefix of `appname-dbname-`. If you do specify a full path, MaxL creates whatever directory you specify.
- When MaxL exports LROs from a database, if the database is from a Unicode-mode application, the exported LRO-catalog file is encoded in UTF-8. You cannot use UTF-8-encoded export files from a Unicode-mode application to import LROs to a non-Unicode mode application.

**Example**

```sql
export database sample.basic lro to server directory 'C:/home/temp/lros';
```

Exports LRO-catalog information, and binary files if the database has file-type LROs, to a server directory called `home/temp/lros`. The directory contains file-type LROs, if applicable, and the LRO-catalog export file `lros.exp`. These can be brought back into a database using `import lro`.

```sql
export database sample.basic lro to server directory 'exportedLROs';
```

Exports LRO-catalog information, and binary files if the database has file-type LROs, to a server directory `$ARBORPATH/app/sample-basic-exportedLROs`. The
directory contains file-type LROs, if applicable, and the LRO-catalog export file named sample-basic-exportedLROs.exp. These can be brought back into a database using import lro.

export database sample.basic lro to server directory 'D:\MaxL\LROexports\dir';

On Windows, exports LRO-catalog information to a new directory dir under the existing directory structure D:\MaxL\LROexports. The double backslashes (\) must be used because a single backslash is an escape character to MaxL.

Export Outline

Export metadata, either from the active database outline or an input outline file, to a specified XML file. Export outline files must be written to a location on the Essbase Server or client computer on which the export outline MaxL statement is run.

Permission required: Database Manager.

Syntax

```
export outline [DBS-NAME] [FILE-NAME] [DIM-NAME] [ALT-NAME-SINGLE] to xml_file [FILE-NAME] [all dimensions] [tree] [with aliasable [ALT-NAME-SINGLE]]
```

• **DBS-NAME**
• **FILE-NAME**
• **DIM-NAME**
• **ALT-NAME-SINGLE**

You can export metadata information from a database in the following ways using export outline.

Keywords

**DBS-NAME**
Specify the database name instead of the outline file path.

**FILE-NAME**
Specify the outline file path instead of the database name.

**all dimensions**
Export information about all dimensions in the database.
**list dimensions**
Export information about only the listed dimensions. Specify each dimension name within curly braces, and separated by commas.

**tree**
Export only the member names in the hierarchy, omitting full metadata details.

**with alias_table**
Export using only the member names indicated in the specified alias table.

**to xml_file**
Specify the full path to the output XML file.

**Notes**
- This statement requires the database to be started.
- The following general outline information is included in the XML export:
  - Case sensitiveness
  - Outline Type
  - Duplicate Member Names allowed
  - Typed Measures Enabled
  - Date Format
  - Varying Attributes Enabled
  - Alias Table count and list
  - Active Alias Table
  - Attribute information
  - Auto configure
  - Text list definitions
  - Universal member comments
  - Locale, if it exists
  - Query hint list (if aggregate storage)
- The following dimension information is included in the XML export:
  - Name
  - Two pass calc
  - Type
  - Text list, if text typed
  - Formula
  - Format String
  - Comment
  - Extended member comment
  - Dimension category
  - Attribute type
- Data Storage
- Dimension Storage
- Alias Names, if any
- UDAs, if any
- Consolidation
- Attribute dimension associated
- Independent dimensions, if any
- Time balance
- Skip options
- Variance reporting
- Currency conversion
- Currency conversion member
- Dynamic Time Series enabled list
- Attachment level, if linked attribute dimension
- Dimension solve order
- Is Non Unique dimension?
- Hierarchy type
- Level usage for aggregation (for aggregate storage hierarchies)
- Is Compression dimension? (if aggregate storage)
- Storage category

• The following member information is included in the XML export:
  - Name
  - Two pass calc
  - Type
  - Text list, if text typed
  - Is shared?
  - Shared member name, if shared
  - Formula
  - Format string
  - Comment
  - Extended member comment
  - Attribute type
  - Data storage
  - Dimension storage
  - Alias names, if any
  - UDAs, if any
  - Consolidation
– Attribute member associated
– Validity sets, if any
– Time balance
– Skip options
– Variance reporting
– Currency conversion
– Currency conversion member
– Member solve order (if aggregate storage)
– Level usage for aggregation (for aggregate storage hierarchy members)

Example

export outline sample.basic all dimensions to xml_file "c:/temp/basic.xml";

Exports all outline information from Sample.Basic to the specified XML file, basic.xml.

export outline sample.basic list dimensions ("Product", "Market") tree to xml_file "c:/temp/basic.xml";

Exports information about Product and Market dimensions from Sample.Basic to the XML file.

Export outline "c:/temp/basic.otl" all dimensions with alias_table "Default" to xml_file "c:/temp/basic.xml";

Exports information about all dimensions in Sample.Basic from the specified outline file to the XML file, using only default alias names.

Grant

Grant a a filter or a stored calculation to a user or a group.

Syntax

```plaintext
grant filter FILTER-NAME execute CALC-NAME any default on system on application APP-NAME on database DBS-NAME to USER-NAME GROUP-NAME
```

* FILTER-NAME
You can grant permissions to users and groups in the following ways using `grant`.

**Keywords**

- **filter `<filter-name>` to...**
  Assign a filter to a user or group that grants or denies permissions to the specified database at a data-value level of detail.

- **execute `<calc-name>` to...**
  Grant the user or group permission to run the specified stored calculation script.

- **execute any on system to...**
  Grant the user or group permission to run any calculation against any database on the Essbase Server.

- **execute any on application...to...**
  Grant the user or group permission to run any calculation against any databases in the specified application.

- **execute any on database...to...**
  Grant the user or group permission to run any calculation against the specified database.

- **execute default on system to...**
  Grant the user or group permission to run the default calculation against any database on the Essbase Server.

- **execute default on application...to...**
  Grant the user or group permission to run the default calculation against any databases in the specified application.

- **execute default on database...to...**
  Grant the user or group permission to run the default calculation against the specified database. The default calculation is typically 'CALC ALL;', but it can be changed using `alter application set default calculation`.

**Notes**

- **Granting filters:**
  Users may be granted multiple filters per database.

- **Granting calculations:**
  A user or group may have any number of calculations per database. Therefore, granting a calculation adds it to the user or group's list of calculations. **Grant execute any** gives the user or group permission to execute all calculations, including the default calculation.

**Example**

```plaintext
grant filter Sample.basic.filter8 to Fiona;
```
Import Data

Click here for aggregate storage version

Import data from data files, with or without a rules file.

Minimum permission required: Write.

Syntax

- `import database DBS-NAME` using max_threads INTEGER data →
- `<data-file-spec>` using max_threads INTEGER data →
  - `<data-record-spec>` using max_threads INTEGER data →
  - `<SQL-connect-spec>` using max_threads INTEGER data →
- `<data-file-spec>` ::= from local text data file IMP-FILE
  - server
- `<data-record-spec>` ::= from data_string STRING
- `<SQL-connect-spec>` ::= connect as SQL-USR identified by SQL-PASS using rules_file IMP-FILE
  - local
  - server
- `<data-error-spec>` ::= on error write append abort to FILE-NAME

- **DBS-NAME**
- **INTEGER**
- **IMP-FILE**
- **FILE-NAME**

You can import data to a database in the following ways using `import data`. 
Keywords

...using max_threads INTEGER
Optionally specify a maximum number of threads to use, if this is a parallel data load. If this clause is omitted for a parallel data load, Essbase uses a number of pipelines equal to the lesser of number of files, or half the number of CPU cores.

import database <dbs-name> data from...
Specify whether the data import file(s) are local or on the server, and specify the type of import file(s).
To import from multiple files in parallel, use the wildcard characters * and/or ? in the IMP-FILE name so that all intended import files are matched.

• * substitutes any number of characters, and can be used anywhere in the pattern. For example, day*.txt matches an entire set of import files ranging from day1.txt - day9.txt.

• ?* substitutes one occurrence of any character, and can be used anywhere in the pattern. For example, 0?*-2011.txt matches data source files named by date, for the single-digit months (Jan to Sept).

...using ... rules_file
Import data into the database using a specified rules file. If you are using a rules file for a parallel data load, all the data files in the load must be able to use the same rules file.

...<data error spec> (on error...)
Required. Tell Essbase what to do in case of errors during the data load: abort the operation, or write or append to a specified error log.

...<data record spec> from data_string
Load a single data record into the selected database. The string following data_string must be a contiguous line, without newline characters.

...<SQL connect spec> (connect as...)
If you are importing data from an SQL source, provide your SQL user name and password. You must always use a rules file when you load SQL data sources.

Notes

• This statement requires the database to be started.

• When using the import statement, you must specify what should happen in case of an error.

• To import from a SQL data source, you must connect as the relational user name, and use a rules file.

Example

import database Sample.Basic data from server data_file 'expsamp.txt' on error abort;

import database Sample.Basic data from server data_file '/Sample/Basic/expsamp.txt' on error abort;
Import Dimensions

Import dimensions from data files, using a rules file.

Minimum permission required: Write.

Syntax

```
import database DBS-NAME dimensions
from <SQL-connect-spec> data_file IMP-FILE using rules_file IMP-FILE
```

- **DBS-NAME**
- **IMP-FILE**
- **FILE-NAME**

You can import dimensions to a database in the following ways using `import` dimensions.

Keywords

import database `<dbs-name>` dimensions from...
Specify whether the dimension import is from a local or server file, and what type of file to import the dimension from.

...using ... rules_file
Import dimensions into the database outline using a specified rules file.

...enforce verification
Verify the outline resulting from the dimension build. This is the default behavior.

...suppress verification
Do not verify the outline resulting from the dimension build.
Caution:
Using this option defers restructuring.

...preserve all data
If you need to preserve all data when importing dimensions, specify that here.

...on error...
Tell Essbase what to do in case of errors during the dimension build: abort the operation, or write or append to an error log.

...<SQL connect spec> (connect as...)
If you are importing dimensions from an SQL source, provide your SQL user name and password. You must always use a rules file when you load SQL data sources.

...<preserve spec alt> (preserve...data)
If you need to preserve level-0 or input data when importing dimensions, specify that here.

Notes

- This statement requires the database to be started.
- When using the import statement, you must specify how error logs should be handled.
- When multiple files are included in the same statement, restructure is deferred until all files have been processed. The deferred-restructure type of dimension build has been called an incremental dimension build.
- When the suppress verification option is used, restructure is deferred.
- When multiple files are included in the same statement, be sure verification is enforced for the last file.
- To import from a SQL data source, you must connect as the relational user name, and use a rules file.

Example

```
import database sample.basic dimensions
from data_file 'dims.txt' using rules_file 'rulesfile.rul'
on error append to 'dimbuild.log';
```

Import LRO

Import Linked Reporting Objects (LROs) from the specified output directory created by export lro. The directory contains an ASCII .exp file containing LRO-catalog information, and LRO binary files (if the database from which LROs were exported contained file-type LROs).

Minimum permission required: Write.

Syntax
You can import exported LRO information to a database using `import lro`.

**Keywords**

`import database <dbs-name> lro...`

Import Linked Reporting Objects (LROs) from the specified export directory on the local computer or on a remote server where the Essbase Server resides.

**Notes**

- This statement requires the database to be started.
- The specified import directory must come from the results of the `export lro` operation. The exported LRO-catalog file contains a record of the LRO file locations, cell notes, or URL text, and database index locations to use for re-importing to the correct data blocks.
- In the paths in the second two examples, double quotation marks are used to allow variable expansion in the string `IMPORT-DIR`, and single quotation marks are required because there are special characters (see MaxL Syntax Notes) in the path name.

**Example**

**Windows Example**

```maxl
import database sample.basic lro
from server directory 'C:\\Hyperion\\products\\Essbase\\EssbaseServer\\app\\sample-basic-lros';
```

```maxl
import database sample.basic lro
from directory "'$ARBORPATH/app/sample-basic-lros'";
```

**UNIX Example**

```maxl
import database sample.basic lro
from server directory "'$ARBORPATH/app/sample-basic-lros'";
```

From the subdirectory created by `export lro` in the app directory on the server, both the Windows and UNIX example statements above re-import the LRO-catalog information (and file-type LROs if applicable) that were exported to that location.

**Query Application**

Click here for aggregate storage version
Get information about the current state of the application.
This statement requires the application to be started.

Syntax

query application APP-NAME get cache_size

APP-NAME

You can query application state information using keywords.

Keywords

get cache_size
Check the current maximum size setting to which the application cache may grow.
The application cache grows dynamically until it reaches this limit. The application
cache is used for hybrid aggregation in block storage databases, and can help you
manage memory usage for retrievals.

Example

The following MaxL statement:

query application sample get cache_size;

returns the maximum size (in kilobytes) to which the application cache may grow.

Query Archive_File

Retrieve information about the database backup archive file.
Minimum permission required: Read.
The database must be running.

Syntax

query archive_file FILE-NAME

FILE-NAME

You can query archive file information using keywords.

Keywords

get overview
Retrieve the following overview information:

• Application name
• Database name
• Time when the archive was performed

**list disk volume**
Retrieve a list of disk volume names.
On Windows, Essbase adds the default ARBORPATH drive (for example, the C: drive) as a disk volume, even if the database that you backed up does not store data on that disk volume.

**Example**

```maxl
query archive_file /Hyperion/samplebasic.arc get overview;
```

Retrieves overview information about the samplebasic.arc backup archive file.

```maxl
query archive_file /Hyperion/samplebasic.arc list disk volume;
```

Retrieves disk volume information about the samplebasic.arc backup archive file.

**Query Database**

*Click here for aggregate storage version*

Get advanced information about the current state of the database.

Minimum permission required: Read.

This statement requires the database to be started.

**Syntax**
You can query for database information in the following ways using `query database`.

**Keywords**

- `get active alias_table`
  Display the active alias table for the user issuing the statement.
**get attribute_info**  
Get attribute member, dimension, and name information for the specified attribute member.

**get attribute_spec**  
Display the current attribute specifications for the database. These specifications include attribute member name format, Attribute Calculation dimension member names, Boolean and date member names, and numeric range specifications.

**get currency_rate**  
Display the currency rate for every currency partition.

**get dbstats dimension**  
Get information about dimensions.  
**Output**  
The `index_type` field values are numeric, and translate as follows:

0               Dense  
1               Sparse  
3               None (database is aggregate storage)

**get dbstats data_block**  
Get information about data blocks. The information returned has little relevance to aggregate storage databases.  
**Output**  
The type field values are numeric, and translate as follows:

0               Array  
1               AVL (or "B+ Tree")

**get default calculation**  
View the contents of the calculation designated as default for the database. The default calculation refers to either the relations defined in the database outline (CALC ALL) or to the set of calculation strings defined as the default database calculation.

**get member_info MEMBER-NAME**  
Get information on a specific member.  
**Output**  
The `unary_type` field values are numeric, and translate as follows:

0               Add  
1               Subtract  
2               Multiply  
3               Divide  
4               Percent  
5               NoRollUp  

The `member_tag_type` field values translate as follows:

0               SkipNone  
16384           SkipMissing  
32768           SkipZero
Variations are possible. The field value consists of one of the first four "skip" values plus any/all/none of the last five values. Some examples:

0               SkipNone
77              SkipNone, BalFirst, TwoPass, Average, Expense
16385           SkipMissing and BalFirst

The first four "skip" values are base values, and added to them are combinations of 1, 2, 4, 8, and 64.

The **status** field values are hexadecimal, and translate as follows:

0               Normal
1               Never Share
2               Label
4               Refer Share
8               Refer Share (with different name)
16              Implicit share
32              Virtual Member (stored)
64              Virtual Member (not stored)
2048            Attribute
32768           Referred

**get member_calculation MEMBER-NAME**
View the formula associated with the selected member.

**get estimated size**
Display an estimate of the number of blocks a database will create after full calculation (CALC ALL), based on the number of blocks that exist before calculation. The database can have all data loaded, or it can have a random sampling of data loaded. Outlines that contain sparse formulas of any type or top-down formulas are not supported. Results of the estimation on such databases may be invalid.

**performance statistics...table**
Display one of several choices of performance statistics tables. Before you can use this statement, you must enable performance statistics gathering, using `alter database DBS-NAME set performance statistics enabled`.

**list alias_table**
Get a list of alias tables that are defined for the database.

**list alias_names in alias_table**
List the alias names defined in an alias table. Alias tables contain sets of aliases for member names and are stored in the database outline. Use this grammar to see a list of alias names defined in the specified table.
list lro
Get information about linked objects, including the object type, name, and description, based on criteria you specify. If you specify both a user name and modification date, objects matching both criteria are listed. If you specify no user name or date, a list of all linked objects in the database is displayed.

list...file information
Get accurate index and data file information. Provides index and data file names, counts, sizes, and totals, and indicates whether or not each file is presently opened by Essbase. The file size information is accurate. Note that the file size information provided by the Windows operating system for index and data files that reside on NTFS volumes may not be accurate.

list transactions
Display, in the MaxL Shell window, database transactions that were logged after the time when the last replay request was originally executed or after the last restored backup's time (which ever occurred later).

list transactions after LOG-TIME
Display, in the MaxL Shell window, database transactions that were logged after the specified time. Enclose the TIME value in quotation marks; for example: '11_20_2007:12:20:00'

list transactions after LOG-TIME write to file PATHNAME_FILENAME
Write the list of database transactions to the specified file. The list output is written to a comma-separated file on the Essbase Server computer. Provide the full pathname to an existing directory and the name of the output file. If only the output file name is provided, Essbase writes the file to the ARBORPATH/app directory. When writing to an output file that already exists, you must use the force grammar to overwrite the file.

list transactions force write to file PATHNAME_FILENAME
Overwrite the contents of an existing output file.

list transactions after TIME...write to file PATHNAME_FILENAME
Write the list of database transactions that were logged after the specified time to the specified file.

Example

Example 1

query database Sample.Basic list transactions;

Displays, in the MaxL Shell window, Sample.Basic database transactions that were logged after the time when the last replay request was originally executed or after the last restored backup's time (which ever occurred later).
Example 2

query database Sample.Basic list transactions after '11_20_2007:12:20:00'
write to file 'C:\Hyperion\products\Essbase\EssbaseServer\app\Sample\Basic\listoutput.csv';

Writes the transactions in the Sample.Basic database that
were logged after November 20, 2007 at 12:20:00 to a CSV
file in the Sample.Basic database directory.

Example 3

query database sample.basic get member_calculation 'Profit per Ounce';

Displays the formula associated with the 'Profit per Ounce'
member.

Example 4

query database sample.basic list lro before '06_16_2008';

Displays information about linked objects, in the Sample.Basic
database, that were modified before the specified time.

Refresh Outline

Synchronize the outlines between partitioned databases. Use this in the event that one
outline has undergone changes to dimensions, members, or member properties, and
you wish to propagate those changes to the partitioned database.

Outline synchronization is not currently enabled for partitions that involve aggregate
storage databases.

Syntax
You can synchronize the outlines between partitioned databases using `refresh outline`.

**Keywords**

...to...
Use the current source outline to refresh the remote target outline.

...from...
Refresh the current target outline using the remote source outline.

**purge outline change_file**
Clear any source outline changes that have already been applied to the target outline or have been rejected. Source outline changes that have not been applied or rejected are not deleted from the outline change file.

**apply all**
Refresh all aspects of the target outline, including dimension changes, member changes, and member property changes made to the source outline. This is the
recommended method for refreshing outlines, because if you choose to omit some changes, those changes cannot be applied later.

**apply nothing**
Do not apply source outline changes to any aspects of the target outline. The target outline will be considered synchronized to the source, and the timestamp will be updated, although source changes were not actually applied to the target.

**apply on dimension...**
Refresh the target outline with all or some dimension changes made to the source outline.

- **add**: Refresh with added dimensions.
- **delete**: Refresh by deleting dimensions.
- **rename**: Refresh with renamed dimensions.
- **update**: Refresh with dimensions that have member updates (required if the statement will also use **apply on member**).
- **move**: Refresh the order of dimensions in the outline.

Use commas to separate the types of source dimension changes to refresh on the target. For example, to refresh only with added or moved dimensions, use the following phrase: `apply on dimension add, move`.

**apply on member...**
Refresh the target outline with all or some physical member changes made to the source outline. Requires **apply on dimension update**.

- **add**: Refresh dimensions with added members.
- **delete**: Refresh dimensions by deleting members.
- **rename**: Refresh dimensions with renamed members.
- **move**: Refresh the order or hierarchy of members in the dimension.

Use commas to separate the types of source member changes to refresh on the target. For example, to refresh only with added or moved members, use the following phrase: `apply on dimension update, apply on member add, move`.

**apply on member_property...**
Refresh the target outline with all or some member property changes made to the source outline. Requires **apply on dimension update**.

- **account_type**: Refresh with changes in account type.
- **alias**: Refresh with changes to aliases.
- **calc_formula**: Refresh with changes to member formulas.
- **consolidation**: Refresh with changes to consolidation tags.
- **currency_conversion**: Refresh with changes to currency conversion flags.
- **currency_category**: Refresh with changes to currency categories.
- **data_storage**: Refresh with changes to data storage tags.
- **uda**: Refresh with changes to UDAs.

Use commas to separate the types of source member-property changes to refresh on the target. For example, to refresh only with updated member formulas, use the
following phrase: apply on dimension update, apply on member_property calc_formula.

Example

refresh outline on replicated partition sampeast.east to samppart.company apply all;

Refresh Replicated Partition

Refresh the current replicated-partition target cube from the remote (second DBS-NAME) source partition. Database Manager permission for each cube is required.

Syntax

```
refresh replicated partition DBS-NAME to DBS-NAME
from HOST-NAME all updated data
```

- DBS-NAME
- HOST-NAME

You can update a replicated-partition using `refresh replicated partition`.

Keywords

...to...

Use the current replicated-partition source cube to refresh the remote target partition.
refresh replicated partition sampeast.east to samppart.company at "https://myEssbase-myDomain.analytics.us2.example.com/essbase/agent" all data;

MaxL Definitions

This section contains the following topics:

- MaxL Syntax Notes
- Numbers in MaxL Syntax
- Terminals
- Privileges and Roles
- Quoting and Special Characters Rules for MaxL Language

MaxL Syntax Notes

The following syntax scheme applies to the creation of MaxL statements.

A MaxL statement corresponds to a sentence telling Essbase what to do with users and database objects. In this documentation, the grammar of MaxL statements is illustrated using railroad diagrams.

When issued via the MaxL Shell (essmsh), statements must be terminated by semicolons. Semicolons are used only to tell the shell when to terminate the statement; semicolons are not part of the MaxL language itself. Therefore, when issuing MaxL statements programmatically external programs, do not terminate with a semicolon.

A token is a delimited sequence of characters recognized by MaxL as a single readable unit. Tokens may be singleton names, keywords, strings, or numbers. Names can have one, two, or three tokens, delimited by periods. The space delimiting tokens can be any white space: spaces, tabs, new lines, or blank lines.

A keyword is a sequence of alphabetic characters that is part of the MaxL grammar. Each keyword is recognized as one token. To be recognized as keywords, keywords cannot be enclosed in quotation marks. However, if you wish to use MaxL keywords outside of the grammar as terminals (for example, as database names or passwords), they must be enclosed in single or double quotation marks.

A terminal is something referenced in the grammar for which you provide the correct name or definition. Terminals can be names, numbers, or strings. Examples: username, filter-name, size-string.
A **name** is a string which can be quoted or unquoted. Unquoted names must begin with an alphabetic character. Quoted names can consist of any sequence of characters. Names in MaxL are used to uniquely identify databases and database objects, such as users, applications, or filters.

Names in MaxL may be one of three types:

- **Singletons**, which are names with one token (example: `Sample`). Use a singleton name for objects that have a system-wide context: for example, applications.
- **Doubles**, which are names with two tokens. A double is two names connected by a period (example: `Sample.basic`). Use doubles to name objects with application-wide contexts, such as databases.
- **Triples**, which are names with three tokens. A triple is three names connected by two periods (example: `Sample.Basic.Calcname`). Use triples to name objects having database-wide contexts, such as filters.

A **string** is unquoted or quoted. An unquoted string can be any sequence of non-special characters. A quoted string can be any sequence of characters (special, alphabetic, or numeric) in the MaxL Alphabet, enclosed in single or double quotation marks.

A **number** is one kind of token which may be passed to Essbase by MaxL. To have meaning, the number must be in the correct format for the Essbase value it represents. In the MaxL grammar documentation, labels for numbers indicate whether the allowed number is positive, negative, an integer, or a real. See **Numbers in MaxL Syntax**.

The MaxL **alphabet** consists of the following elements:

### Table 5-11 MaxL Alphabet Elements

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Special characters</td>
<td>Valid special characters: . , ; % $ ’ ‘ SPACE</td>
</tr>
<tr>
<td></td>
<td>TAB * + - = &lt; &gt; [ ] { } ( ) ? ! / \ ~ ` # &amp; @ ^</td>
</tr>
<tr>
<td></td>
<td>When using special characters in MaxL terminals, note the quoting rules (see Quoting and Special Characters Rules for MaxL Language).</td>
</tr>
<tr>
<td>Non-special characters</td>
<td>Alphabetic characters and numbers.</td>
</tr>
<tr>
<td>Alphabetic characters</td>
<td>Letters of the alphabet, and the underscore.</td>
</tr>
<tr>
<td></td>
<td>[a-z, A-Z, _]</td>
</tr>
<tr>
<td>Numbers</td>
<td>See Numbers in MaxL Syntax</td>
</tr>
</tbody>
</table>

### Numbers in MaxL Syntax

Numbers in MaxL statements fit into one of the following categories.

- `<NUMBER>` ::= ![INTEGER](chart) ![REAL](chart)
- **INTEGER**—Zero or a positive integer. Decimals and scientific notation are permitted. Examples: 0, 1, 1000, 1.3e4
• REAL—Zero or a positive real number. Decimals and scientific notation are permitted. Examples: 0.0, 1, 1000, 1000.4, 13.1e-4

Terminals

The following sections describe terminals in alphabetical order.

ACTION

The required action if a data-monitoring trigger is activated.

Syntax

mail [smtp],[sender],[receiver1,receiver2,...],[subject]
spool FILE-NAME

• mail - sends an email from the specified sender, to a specified email address or addresses, with the specified subject line (optional). Enclose email addresses containing special characters in square brackets ([ ]). The mail action is not supported for after-update triggers, which are the only triggers available for use with aggregate storage cubes.
• spool - logs a message in a specified file in the \trig folder under the cube directory.

Type

string (see MaxL Syntax Notes)

Example

mail manager.sales.com, [mktdir@CC.com, Monitor@acnts.com]
spool "trgmonitor"

Referenced By

create trigger
drop trigger

ALT-NAME-SINGLE

The name of an alias table. If the name contains special characters (see MaxL Syntax Notes), it must be enclosed in single or double quotation marks.

Type

name (see MaxL Syntax Notes)

Example

Region
'Long Names'
Referenced By

alter database
query database

APP-NAME

The name of the application.

The application name must not exceed 8 bytes (non-Unicode-mode applications) or 30 characters (Unicode-mode applications). Avoid using spaces. Application names are not case-sensitive.
If the name contains any allowed special characters, it must be enclosed in single or double quotation marks. Only the following special characters are allowed by Essbase within application names:

- % (percent sign)
- $ (dollar sign)
- - (minus sign)
- { (open brace)
- } (close brace)
- ( (open parenthesis)
- ) (close parenthesis)
- ! (exclamation mark)
- ~ (tilde)
- ` (accent mark)
- # (pound sign)
- & (ampersand)
- @ (at sign)
- ^ (caret)

Type name (see MaxL Syntax Notes)

Example

Sample
Referenced By
alter application
alter partition
alter system
create application
display application
display calculation
display database
display location alias
display lock
display object
display session
display trigger spool
drop application
drop lock
grant
query application

AREA-ALIAS

A shorthand name used in the the create partition statement for referring to an already-specified member expression that designates which areas of the databases should be partitioned.

Type
name (see MaxL Syntax Notes)

Example
In the create partition statement below, "foo" is an area-alias for the member expression specified in the area specification. To create area-aliases, enter the alias names after the member expression in each area specification. To specify which area is relevant when mapping members (if applicable), refer to its alias name in the mapped phrase.

In the example below, the alias name as created is shown in this color, and it specifies which area (in other words, it refers to the entire member expression string, '@DESCENDANTS(East) @DESCENDANTS(Qtr1)'). The alias name as referenced is shown in this color.

create or replace replicated partition sampeast.east
    area '@DESCENDANTS("Eastern Region"), @DESCENDANTS(Qtr1)'
to samppart.company at aspen
as admin identified by 'password'
area '@IDESCENDANTS(East) @IDESCENDANTS(Qtr1)' foo
mapped foo (Year) to (Yr)
update allow validate only;

[Note:]
All area aliases used in a mapping should be associated with the target (as in the example above), and the direction of member names listed in the mapped clause should go from source to target.

Referenced By
create partition

BUFFER-ID
A number between 1 and 999,999 inclusive. To destroy a buffer before a data load is complete, you must use the same BUFFER-ID number that was used to initialize the buffer.

Type
number (see MaxL Syntax Notes)

Referenced By
alter database

CALC-NAME
A stored calculation.

Syntax
- Syntax for database-level calculation:
  \(name1.name2.name3\)
- Syntax for application-level calculation:
  \(name1.name3\)

- name1—Application name.
- name2—Database name (not required for application-level calcs).
- name3—Calculation script name.

Type
name (see MaxL Syntax Notes)
For calculations associated with databases, three tokens are required, to indicate application and database context and the calculation name.

**Example**

```plaintext
Sample.basic.'alloc.csc'
```

For application-level calculations, two tokens are required, indicating application context and the calculation name. When executing application-level calculations, you must specify which database to calculate using the syntax 'on database STRING.'

**Example**

- `Sample.'alloc.csc'` is the application-level CALC-NAME.
- `execute calculation Sample.'alloc.csc' on database Basic;` is a way to execute the application-level calculation on a database.

If any part of the name contains special characters (see MaxL Syntax Notes), it must be enclosed in single or double quotation marks.

**Referenced By**
- create calculation
- display calculation
- drop calculation
- execute calculation
- grant

**CALC-NAME-SINGLE**

A stored calculation name that is the third token of a database-level CALC-NAME.

If any part of the name contains special characters (see MaxL Syntax Notes), it must be enclosed in single or double quotation marks.

**Type**

name (see MaxL Syntax Notes)

**Example**

If the full database-level calc name is `sample.basic.'alloc.csc'`, then CALC-NAME-SINGLE is 'alloc.csc'.

**Referenced By**
- alter database

**CALC-STRING**

A calculation string. The body of an anonymous (unstored) calculation, or the string used to specify the body of a stored calculation at create time.
Because calculations are terminated with a semicolon, and semicolons are special characters to MaxL, CALC-STRING should be enclosed in single or double quotation marks.

**Type**

string (see MaxL Syntax Notes)

**Example**

CALC DIM(Year, Measures, Product);

**Referenced By**

alter database

execute calculation

**COLUMN-WIDTH**

A number (at least 8) representing character-width of columns; or, the keyword default, representing 20 characters wide.

**Type**

number (see MaxL Syntax Notes) or default

**Example**

set display column width 80

set display column width default

**Referenced By**

Set Display Column Width

**COMMENT-STRING**

A string of user-defined informational text. If the string contains special characters (see MaxL Syntax Notes), it must be enclosed in single or double quotation marks.

**Type**

string (see MaxL Syntax Notes)

**Example**

'This is a comment.'

**Referenced By**

alter application

alter database

create application

create database
create partition

CONDITION

A numeric-value-expression developed in MDX. Must be enclosed in double quotation marks. Enclose strings containing special characters in square brackets ([ ]).

Type

string (see MaxL Syntax Notes)

Example

"Jan>20"

Referenced By

create trigger

CUBE-AREA or MDX-SET

A cube area or other specification developed in MDX as a symmetric, syntactically-valid set. The area specification must be static, for example it cannot contain Dynamic Calc members or runtime functions such as Filter, TopSum, or BottomSum. Enclose strings containing special characters in square brackets ([ ]). For complete information about defining MDX sets, see MDX Set Specification in the MDX section.

Type

string (see MaxL Syntax Notes)

Examples

The following is a set of siblings.

'([Jan 2000], [Feb 2000], [Mar 2000])'

The following is a crossjoined set.

'((Qtr1, [New York]), ([Qtr1], [California]),
 (Qtr2, [New York]), ([Qtr2], [California]))'

The following set is also a tuple.

'((Jun, FY2011, Actual))'

The following statement clears data from a region of ASO samp.Sample. The region is defined using a CUBE-AREA expressed in MDX.

alter database ASO samp.sample clear data in region '((Coupon, [Prev Year],
South))' physical;
DATE

A valid date string formatted according to these rules:

- MM/DD/YYYY or MM/DD/YY
- Any character can be used as a separator; for example, MM-DD-YY is valid.

If the string contains special characters (see MaxL Syntax Notes), it must be enclosed in single or double quotation marks.

Type

string (see MaxL Syntax Notes)

Example

'04/16/03'
'04.16.2003'
04_16_2003

DBS-EXPORT-DIR

Suffix for the name of a cube directory to contain export files, to be created (upon export lro) in the application directory as appname-dbname-suffix.

After LRO export, the directory contains file-type LRO binary files (if applicable to the database), and the LRO-catalog export file with file-extension .exp.

If for a Sample.Basic export, DBS-EXPORT-DIR is given as lros, then the sample-basic-lros directory is created in the application directory. The sample-basic-lros directory contains file-type LRO binary files and the LRO-catalog export file 'sample-basic-lros.exp'.

Notes:

- MaxL creates exactly one export directory; it does not create a directory structure.
- If the specified export directory already exists, the export LRO statement fails, as a safeguard against overwriting.
Type
string (see MaxL Syntax Notes)

Referenced By
export lro

**DBS-NAME**

The name of a database. Two tokens are required, to indicate application context.

**Syntax**

\texttt{name1.name2}

- **name1**—The name of the application containing the database.
  
  The application name must not exceed 8 bytes (non-Unicode-mode applications) or 30 characters (Unicode-mode applications). Avoid using spaces.

- **name2**—The name of the database.
  
  The database name must not exceed 8 bytes (non-Unicode-mode applications) or 30 characters (Unicode-mode applications). Avoid using spaces.

Database names are not case-sensitive.

If the name contains any allowed special characters, it must be enclosed in single or double quotation marks. Only following special characters are allowed by Essbase within database names:

- `%` (percent sign)
- `$` (dollar sign)
- `-` (minus sign)
- `{` (open brace)
- `}` (close brace)
- `(` (open parenthesis)
- `)` (close parenthesis)
- `!` (exclamation mark)
- `~` (tilde)
- `\` (accent mark)
- `#` (pound sign)
- `&` (ampersand)
- `@` (at sign)
- `^` (caret)

**Type**

name (see MaxL Syntax Notes)

**Example**

Sample.basic
Referenced By

alter database
alter partition
alter system
alter trigger
create database
create location alias
create outline
create partition
display database
display filter
display filter row
display location alias
display lock
display object
display partition
display session
display trigger spool
display variable
drop database
drop lock
drop partition
drop trigger spool
execute aggregate build
execute aggregate process
execute aggregate selection
export data
grant
import data
import dimensions
import lro
query database
refresh outline
refresh replicated partition

**DBS-STRING**

The second token of `DBS-NAME`. Limit 8 characters.

If the name contains special characters (see MaxL Syntax Notes), it must be enclosed in single or double quotation marks.

**Type**

string (see MaxL Syntax Notes)

**Example**

basic

**Referenced By**

alter application
alter database
alter partition
execute calculation

**DIM-NAME**

The name of a database dimension.

If the string contains special characters (see MaxL Syntax Notes), it must be enclosed in single or double quotation marks.

**Type**

string (see MaxL Syntax Notes)

**Example**

Year
Market

**Referenced By**

query database

**EXPORT-DIR**

**Type**

string (see MaxL Syntax Notes)
Example
'sample-basic-out'

Referenced By
alter system

FILE-NAME
A file name or path. If the string contains special characters, it must be enclosed in single or double quotation marks.

Type
string (see MaxL Syntax Notes)

Example
- file01
- "errors.txt"
- '/Sample/Basic/expsamp.txt'

Referenced By
alter database
export data
import data
import dimensions

FILE-NAME-PREFIX
Prefix for one or more file names to be created (upon display drillthrough DBS-NAME to FILE-NAME-PREFIX) on the client in the working directory of MaxL execution.

These display output files contain the URL XML content of URL drill-through definitions used to link to content hosted on ERP and EPM applications.

If the string contains special characters (see MaxL Syntax Notes), it must be enclosed in single or double quotation marks.

Type
string (see MaxL Syntax Notes)

Example
urlxmls
FILTER-NAME

The name of a security filter. Three tokens are required, to indicate application and database context.

Syntax

\[ \text{name1}.	ext{name2}.	ext{name3} \]

- \text{name1}—Application name.
- \text{name2}—Database name.
- \text{name3}—Filter name.

Type

name (see MaxL Syntax Notes)

Example

Sample.basic.filt1

FULL-EXPORT-DIR

Full path for the name of a directory for LRO export files, to be created (upon export lro) anywhere on the client or server.

After export lro, the directory contains file-type LRO binary files (if applicable to the database), and the LRO-catalog export file named in the format \text{directoryname}.exp.

For example, if for a Sample.Basic export, FULL-EXPORT-DIR is given as \text{home/temp/lros}, then the lros directory structure is created under \text{home/temp} if \text{home/temp} exists. The lros subdirectory contains file-type LRO binary files and the LRO-catalog export file \text{'lros.exp'}.

Notes:
• MaxL creates exactly one export directory; it does not create a directory structure. In the above example, if the home/temp directory structure exists, MaxL creates the lros directory as a subdirectory of home/temp, but if home/temp does not exist, MaxL will not create home/temp/lros.

• If the specified export directory already exists, the export LRO statement will fail. This is a safeguard against overwriting existing export directories.

**Type**

string (see MaxL Syntax Notes)

**Referenced By**

Export LRO

**GROUP-NAME**

The name of the Essbase security group.

Group name guidelines:

• Non-Unicode application limit: 256 bytes
• Unicode-mode application limit: 256 characters
• Group names must start with a letter or a number
• If the group name contains any special characters (see MaxL Syntax Notes), the name must be enclosed in single or double quotation marks.

**Types**

• name (see MaxL Syntax Notes)
• name@provider
• WITH IDENTITY ID-STRING

**Note:**

If a user or group name includes the @ character, you must specify the provider as well. For example, if you want to log in user admin@msad which is on a Native Directory provider, you must specify 'admin@msad@Native Directory'.
HOST-NAME

Use the discovery URL instead of a host name. A discovery URL is the URL provided by your Service Administrator, with /agent appended to the end. For example, https://myEssbase2.oraclecloud.com/essbase/agent.

Leading or trailing spaces will be trimmed off. Maximum length is 1024 bytes (non-Unicode application) or characters (Unicode application).

ID-RANGE

A comma-separated list of sequence ID ranges for logged sequential transactions. A range can consist of:

- A single transaction: n to n; for example, 1 to 1
- Multiple transactions: x to y; for example, 20 to 100

Type

string (see MaxL Syntax Notes)

Example

1 to 10, 20 to 100

ID-STRING

Unique identity attribute identifying a user or group in a directory.

To find the identities of existing users or groups, use display user or display group.

Example

native://nvid=f0ed2a6d7fb07688:5a342200:1265973105c:-7f46?USER

Referenced By

alter database
GROUP-NAME

IMPORT-DIR

A string representing the full path to the directory used in the `export lro` statement.

**Note:**
If importing lros from a server directory (using `from server` syntax of `import lro`), you can give just the full directory name instead of the full path, as specified by `EXPORT-DIR`.

The string must be enclosed in single or double quotation marks.

**Type**
string (see MaxL Syntax Notes)

**Example**
'home/exports/temp/sample-basic-lros'

For information about how IMPORT-DIR is created, see the grammar and definitions for `export lro`.

**Referenced By**
import lro

IMP-FILE

A name or path to a server-side rules file or data file, used for `import data` and `import dimension` statements.

If the data or rules file is specified to be on the server, the following rules apply. If the data or rules file is specified to be local (or left unspecified, in which case it is also local), skip the following and use `FILE-NAME`.

If you are using `server data_file` or `server rules_file`, you can get the file from any application (not just the current application) by starting the IMP-FILE string using the following pattern:

```
FILE_SEP AppName FILE_SEP DbName FILE_SEP rest_of_file_name
```

where `FILE_SEP` must be `/`.

**Type**
name (see MaxL Syntax Notes)
Examples

Consider the MaxL statement:

```maxl
import database demo.basic data
from server rules_file 'IMP-FILE'
on error abort;
```

If IMP-FILE is 'calcdat.txt' or '/Demo/Basic/calcdat.txt', the file will be looked for in the Demo.Basic cube directory.

```maxl
import database demo.basic data
from server file '/Sample/Basic/Calcdat.txt'
on error abort;
```

Essbase looks for calcdat.txt inside the Sample.Basic cube directory, and loads the data to Demo.Basic.

Referenced By

- import data
- import dimensions

LOCATION-ALIAS-NAME

The name of a location alias referencing another database.

Syntax

```maxl
name1.name2.name3
```

- `name1`—Application name.
- `name2`—Database name.
- `name3`—Location alias name.

Type

name (see MaxL Syntax Notes)

Example

```maxl
Sample.Basic.EasternDB
```

Referenced By

- create location alias
- display location alias
LOC-ALIAS-SINGLE

The single form of a location alias name. Use if you are creating a new location alias.

Type
name (see MaxL Syntax Notes)

Example

EasternDB

Referenced By

alter database
create location alias

LOG-TIME

A specific log time after which to replay subsequent transactions. Enclose the value in quotation marks.

Type
string (see MaxL Syntax Notes)

Example

'11_20_2007:12:20:00'

Referenced By

alter database

ALLOC-NUMERIC

An MDX numeric value expression used to specify the amount for an allocation source. The amount value is allocated to cells in the target region. The allocation numeric is one of the following:

• An MDX tuple
• A number
• An arithmetic expression using member names, with the following restrictions:
  – All members in the expression must be from the same dimension.
  – Tuples cannot be used.
  – Only arithmetic operators (+, -, /, and *) can be used.
  – MDX functions (such as Avg and Parent) are not allowed.
Type

string (see MaxL Syntax Notes)

Examples

- (Acc_1000, Jan_2009)
- 100.00
- (Acc_1000 + Acc_2000)/2
- AcctA + AcctB
- Balance * 1.1

Referenced By
execute allocation

MEMBER-EXPRESSION

Outline member specification of members from one or more dimensions, member combinations separated by commas, or member sets defined with functions. Must be enclosed in single or double quotation marks.

Type

string (see MaxL Syntax Notes)

Example

'@ANCESTORS(Qtr2)'

If MEMBER-EXPRESSION contains MEMBER-NAMES that begin with numbers or contain special characters, enclose those member names in double quotation marks, and the entire MEMBER EXPRESSION in single quotation marks. For example:

- create or replace filter demo.basic.numfilt no_access on '@"2"';
- '@DESCENDANTS("Eastern Region"), @CHILDREN(Qtr1)'

The following example shows how create drillthrough uses a member expression to define the list of drillable regions.

create drillthrough sample.basic.myURL from xml_file "temp.xml" on
('@Ichildren("Qtr1"), '@Ichildren("Qtr2")') level0 only;

Referenced By
alter filter
create filter
create partition
create drillthrough
alter drillthrough
MEMBER-NAME

The name of a database outline member.

If the name contains special characters (see MaxL Syntax Notes), it must be enclosed in single quotation marks.

Type
name (see MaxL Syntax Notes)

Example

Jan

'New York'

If MEMBER-NAME is part of MEMBER-EXPRESSION and MEMBER-NAME begins with a number or contains special characters (see MaxL Syntax Notes), enclose MEMBER-NAME in double quotation marks and enclose MEMBER-EXPRESSION in single quotation marks.

Referenced By
alter database
create partition
query database

OBJ-NAME

The name of a database object. Three tokens are required, to indicate application and database context.

Syntax

name1.name2.name3

- name1—Application name.
- name2—Database name.
- name3—Object name.

Type
name (see MaxL Syntax Notes)

Example

Sample.basic.Calcdat
OBJ-NAME-SINGLE

A stored database object name that is the third token of a database-level OBJ-NAME. If any part of the name contains special characters (see MaxL Syntax Notes), it must be enclosed in single or double quotation marks.

Type

name (see MaxL Syntax Notes)

Example

If the full database object name is sample.basic.calcdat, then OBJ-NAME-SINGLE is calcdat.

OUTLINE-ID

The numeric identification of an aggregate storage outline associated with a view. The outline ID is returned by the execute aggregate selection statement. The execute aggregate selection statement returns a set of views, including the outline ID for the views it returns.

Type

number (see MaxL Syntax Notes)

Example

4142187876

PASSWORD

A user’s password. Not applicable for externally authenticated users.

Password guidelines:

- Non-Unicode application limit: 100 bytes
- Unicode-mode application limit: 100 characters
• If the string contains special characters (see MaxL Syntax Notes), the password must be enclosed in single or double quotation marks
• Leading or trailing spaces are illegal and will be trimmed off

Type
string (see MaxL Syntax Notes)

Referenced By
alter partition
create location alias
create outline
create partition
Login

PATHNAME_FILENAME
A path to a file. If the string contains special characters (see MaxL Syntax Notes), it must be enclosed in single or double quotation marks.

Type
string (see MaxL Syntax Notes)

Referenced By
query database

PRECISION-DIGITS
An integer between 0 and 15, inclusive.

Type
number (see MaxL Syntax Notes)

Referenced By
alter session

PROPS
Aggregate storage data load properties that determine how missing and zero values, duplicate values, and multiple values for the same cell in the data source are processed.

• ignore_missing_values: Ignore missing values in the data source.
• ignore_zero_values: Ignore zeros in the data source.
• aggregate_use_last: Combine duplicate cells by using the value of the cell that was loaded last into the data load buffer. When using this option, data loads are significantly slower, even if there are not any duplicate values.
Caution:
The `aggregate_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 `aggregate_sum` instead.

- `aggregate_sum`: (Default) Add values when the buffer contains multiple values for the same cell.

If you use multiple properties and any conflict occurs, the last property listed takes precedence.

**Type**

string (see MaxL Syntax Notes)

**Referenced By**

`alter database` (aggregate storage)

---

**RNUM**

Resource usage specification for temporary aggregate storage data load buffer.

Must be a number between .01 and 1.0 inclusive. If not specified, the default value is 1.0. Only two digits after the decimal point are significant (for example, 0.029 is interpreted as 0.02). The total resource usage of all load buffers created on a database cannot exceed 1.0 (for example, if a buffer of size 0.9 exists, you cannot create another buffer of a size greater than 0.1). Send operations internally create load buffers of size 0.2; therefore, a load buffer of the default size of 1.0 will cause send operations to fail because of insufficient load buffer resources.

**Type**

number (see MaxL Syntax Notes)

**Example**

0.02

**Referenced By**

`alter database` (aggregate storage)

---

**RTSV-LIST**

A string of runtime substitution variables that can be used in calculation scripts. Runtime substitution variables are specified as key/value pairs. The string must be enclosed with single quotation marks, and key/value pairs must be separated by a semicolon, including a semicolon after the last runtime substitution variable in the string and before the terminal single quotation mark.
Runtime substitution variables—name and default value—must be declared in the SET RUNTIMESUBVARS calculation command. If you include a runtime substitution variable in RTSV-LIST that has not been declared in SET RUNTIMESUBVARS, Essbase ignores the undeclared runtime substitution variable (no warnings or exceptions are generated).

**Type**

string (see MaxL Syntax Notes)

**Example**

In this example of a runtime substitution variable string, the name and value of four runtime substitution variables are specified (for example, the value of the runtime substitution variable named "a" is 100):

'\(a=100; b=@\text{CHILDREN}("100"); c=\text{Actual} \rightarrow \text{Final}; d=\text{New York}\); '

**Referenced By**

execute calculation (block storage only)

**See Also**

SET RUNTIMESUBVARS calculation command

**RULE-FILE-NAME**

A comma separated list of strings of rules-file names. Each rules-file name should be an 8-character object file name with no extension. The rule files must reside on the Essbase server.

**Type**

string (see MaxL Syntax Notes)

**Example**

'h1h1h1', 'h1h1h2'

**Referenced By**

import data (aggregate storage)

**SESSION-ID**

The unique session ID. This ID can be used to logout a user session, or kill the current request in that session.

**Type**

number (see MaxL Syntax Notes)
Example

3310545319

Referenced By

alter system
display session
query database

SIZE-STRING

Syntax

number units

OR

number

- number—Any positive number. Decimals and scientific notation are permitted. Whitespace between number and units is optional.
- units—One of the following: b, kb, mb, gb, tb (case-insensitive). If units are unspecified, bytes are assumed.

Type

number (see MaxL Syntax Notes)

Examples

51040b
51040 b
1M
11000kb
12.34gb
1234e-2gb

Referenced By

alter application
alter database
alter tablespace

SPOOL-NAME

The name of a trigger's output file, as specified in the THEN or ELSE section of the create trigger statement.
Syntax

name1.name2.name3

Type

name (see MaxL Syntax Notes)

Example

In the following create trigger statement, the **bold** section is the spool name.

create or replace trigger Sample.Basic.Trigger_Jan_20
where "(Jan,Sales,[100],East,Actual)"
when Jan > 20 and is(Year.currentmember,Jan) then
spool **Trigger_Jan_20**
end;

Referenced By

display trigger spool
drop trigger spool

STOPTING-VAL

Optional stopping value for the **execute aggregate process** statement. Use this value to give the ratio of the growth size you want to allow during the materialization of an aggregate storage database, versus the pre-aggregation size of the database (Before an aggregation is materialized, the database contains only level 0 input-level data.)

Type

number (see MaxL Syntax Notes)

Example

A stopping value of 1.5 means that during the materialization of the aggregation, the aggregate cells are allowed to occupy up to 50% of the disk space occupied by the level-0 data.

Referenced By

**execute aggregate selection**

**execute aggregate process**

TABLSP-NAME

The name of a tablespace. Tablespaces are applicable only to aggregate storage databases. For this release, possible names for tablespaces you can alter are **default** and **temp**. Other tablespace names reserved by the system are **metadata** and **log**.
Syntax

\texttt{name1.name2}

- \texttt{name1}—Application name.
- \texttt{name2}—Tablespace name.

Type

name (see MaxL Syntax Notes)

Example

temp

Referenced By

\texttt{alter tablespace}

\texttt{display tablespace}

\section*{TRIGGER-NAME}

The name of the trigger device created to track and respond to database updates. Trigger names must be triple names, specifying application name, database name, and trigger name (if you rename the application or database, the trigger is invalidated). Trigger names are case-insensitive, are a maximum of 30 bytes, and cannot contain special characters.

Syntax

\texttt{name1.name2.name3}

- \texttt{name1}—Application name.
- \texttt{name2}—Database name.
- \texttt{name3}—The name of the trigger.

Type

name (see MaxL Syntax Notes)

Example

Sample.Basic.MyTrigger

Referenced By

\texttt{alter trigger}

\texttt{create trigger}

\texttt{display trigger}

\texttt{drop trigger}
URL-NAME

The name of a drill-through URL definition used to link to content hosted on Oracle ERP and EPM applications.

Syntax

\[ \text{name}_1.\text{name}_2.\text{name}_3 \]

- \text{name}_1—Application name
- \text{name}_2—Database name
- \text{name}_3—URL name

Type

name (see MaxL Syntax Notes)

Example

Sample.basic.MyURL

If any part of the name contains special characters (see MaxL Syntax Notes), the name must be enclosed in single or double quotation marks.

Referenced By

- create drillthrough
- alter drillthrough
- display drillthrough
- drop drillthrough

USER-NAME

The name of the user.

User name guidelines:

- Non-Unicode application limit: 256 bytes
- Unicode-mode application limit: 256 characters
- If the user name contains any special characters (see MaxL Syntax Notes), the name must be enclosed in single or double quotation marks.

Types

- name (see MaxL Syntax Notes)
- name@provider
- WITH IDENTITY ID-STRING
Note:

If a user or group name includes the @ character, you must specify the provider as well. For example, if you want to log in user admin@msad which is on a Native Directory provider, you must specify 'admin@msad@Native Directory'.

Examples

JWSmith

JWSmith@Native Directory

with identity "native://nvid=f0ed2a6d7fb07688:5a342200:1265973105c:-7f46?USER"

Referenced By
alter application
alter database
alter partition
alter system
create location alias
create outline
create partition
display privilege
display user
drop lock
grant
query database
Login

VARIABLE-NAME

The name of the substitution variable. The name can only contain alphanumeric characters and the underscore: (a-z A-Z 0-9 _).

Type
name (see MaxL Syntax Notes)
Example
curmonth

Referenced By
alter application
alter database
alter system
display variable

VIEW-FILE-NAME

An aggregation script containing information derived during aggregate view selection.
The file is created in the cube directory, with a .csc extension.
Aggregation scripts are valid as long as the dimension level structure in the outline has
not changed.
Executing an aggregation script (using execute aggregate build) materializes the
aggregate views specified within it.
The .csc extension is optional when executing the script.
The file name can be a maximum of 8 characters in length (excluding the extension)
and must not contain any of the following characters, or whitespace: ;,.=+*?[ ]
<>*'\/

Type
string (see MaxL Syntax Notes)

Referenced By
execute aggregate selection
execute aggregate build
query database

VIEW-ID

The numeric identification of an aggregate view, returned by the execute aggregate
selection statement. The concept of views applies only to aggregate storage
databases.
VIEW-IDs persist only as long as their associated OUTLINE-IDs. OUTLINE-IDs change when changes are made to the outline.

Type
number (see MaxL Syntax Notes)
Example

8941

Referenced By

execute aggregate selection
execute aggregate build

VIEW-SIZE

Approximate view size as a fraction of input data size. For example, a view size of \(0.5\) means that the view is 2X smaller than the input-level view. The concept of views applies only to aggregate storage databases.

Type

number (see MaxL Syntax Notes)

Referenced By

execute aggregate build

Privileges and Roles

Essbase system privileges are indivisible database access types. In MaxL, privileges are grouped together to form permission-sets called roles. Privileges themselves are not grantable using MaxL; you typically grant roles, which are the equivalent of privilege levels. The scope of a role can be the system, the application, or the database.

While one privilege does not imply another, roles are hierarchical. The following table illustrates the Essbase system privileges that are contained in each MaxL system role.

Table 5-12 Privileges and Roles

<table>
<thead>
<tr>
<th>Privileges and Roles</th>
<th>read</th>
<th>write</th>
<th>calculate</th>
<th>manage database</th>
<th>create database</th>
<th>start application</th>
<th>manage application</th>
<th>create/drop application</th>
</tr>
</thead>
<tbody>
<tr>
<td>no access</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>read</td>
<td>☐</td>
<td>☐</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>write</td>
<td>☐</td>
<td>☐</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>execute</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>manager (database)</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
</tbody>
</table>

Application-Level System Roles

Application-level system roles are applicable to an application. The following roles may have an application-wide scope:
Database-Level System Roles

Database-level system roles are minimum access permissions you can set for databases. The following roles have a database-wide scope and are available when assigning minimum database permissions:

- **no_access**—No access to the database (if assigned using `alter database`) or to any databases in the application (if assigned using `alter application`).
- **read**—Read-only access to the database (if assigned using `alter database`) or to all databases in the application (if assigned using `alter application`). Read access means ability to view files, retrieve data values, and run report scripts.
- **write**—Write access to the database (if assigned using `alter database`) or to all databases in the application (if assigned using `alter application`). Write access means ability to update data values, in addition to having Read access.
- **execute**—Calculate access to the database (if assigned using `alter database`) or to all databases in the application (if assigned using `alter application`). Calculate access means ability to update data values, in addition to having Read and Write access.
- **manager**—Manager access to the database (if assigned using `alter database`) or to all databases in the application (if assigned using `alter application`). Manager access means ability to modify database outlines, in addition to having Read and Write access.

Quoting and Special Characters Rules for MaxL Language

These rules apply to terminals of MaxL statements; for example, USER-NAME or FILE-NAME. Rules for MaxL Shell also apply (see MaxL Shell Syntax Rules and Variables).
Tokens enclosed in Single Quotation Marks

Contents are preserved as literal, with the following exceptions:

- One backslash is ignored; two are treated as one.
- Apostrophe must be escaped using one backslash (\').

Example: export database sample.basic data to data_file 'D:\export.txt';
Result: Exports data to D:\export.txt.

Example: display user 'O\'Brien';
Result: Error.

Example: display user 'O\'Brien';
Result: User O’Brien is displayed.

Tokens Enclosed in Double Quotation Marks

Contents are preserved as literal, with the following exceptions:

- Variables are expanded.
- One backslash is ignored; two are treated as one.
- Apostrophe must be escaped using one backslash (\').

Example: export database sample.basic data to data_file "D:\export.txt";
Result: Exports data to D:\export.txt.

Example: export database sample.basic data to data_file "$ARBORPATH\App\Sample\Basic\export.txt";
Result: Exports data to C:\Hyperion\products\Essbase\EssbaseServer\App\Sample\Basic\export.txt.

Example: display user "O\'Brien";
Result: Error.

Example: display user "O\'Brien";
Result: User O’Brien is displayed.

Use of Backslashes in MaxL

Ignored unless preceded by another backslash (the escape character). Must use single or double quotation marks around the token containing the two backslashes.

create application 'finance\budget';
Result: Application finance\budget is created.
Example (Windows):

```maxl
display user 'O\'Brien';
```

Result: User O'Brien is displayed.

Use sparingly. Apostrophes are permitted by Essbase in user and group names, but not in application or database names.

Use of Dollar Signs

Syntax error returned, unless preceded by a backslash (the escape character) and enclosed in single quotation marks. Dollar signs ($) intended literally need to be escaped by the backslash so that they are not considered variable indicators.

Example:

```maxl
create application '\$App1';
```

Result: Application $App1 is created.

MaxL Shell Commands

The MaxL Shell (essmsh) is a pre-parser mechanism for entering MaxL statements. The MaxL Shell has a separate set of useful commands, independent of the MaxL language itself. Before using any of the following MaxL Shell commands, you need to log in (see Login).

- Spool on/off
- Set Display Column Width
- Set Message Level
- Set Timestamp
- Echo
- Nesting
- Error Checking and Branching
- Version
Overview of MaxL Shell

The MaxL Client is a utility through which you execute MaxL statements or scripts.

This section contains the following topics:

- Invocation and Login
- Syntax Rules and Variables
- Shell Commands

MaxL Shell Invocation

The MaxL Shell (essmsh) is a pre-parser mechanism for entering MaxL statements.

You can start the shell to be used interactively, to read input from a file, or to read stream-oriented input (standard input from another process). You can log in after you start the shell, interactively or using a login statement in the input file. You can also log in at invocation time, by using the -l flag (see -l Flag: Login).

To start the essmsh shell, do not invoke it directly. In order for the environment to be set correctly, you must start essmsh using startMAXL.bat (Windows) or startMAXL.sh (UNIX).

- Prerequisites for Using MaxL
- MaxL Invocation Summary
- Interactive Input
- File Input
- Standard Input
- Login
- LoginAs
- Encryption
- Query Cancellation

Prerequisites for Using MaxL

Before the Essbase Server can receive MaxL statements,

1. The Essbase Server must be running.
2. The MaxL Shell (essmsh) must be invoked (see MaxL Invocation Summary), if you are using the shell.
3. You must log in (see Login) to the Essbase Server from the MaxL Shell. If you are running a MaxL script, the first line of your script must be a login statement.

You must use a semicolon (;) to terminate each MaxL statement.
MaxL Invocation Summary

The following MaxL Shell help page summarizes invocation options. This help is also available at the operating-system command prompt if you type `startMAXL.bat -h | more`.

Note:
The following help text is for `essmsh` shell; however, in order for the environment to be set correctly, you must start `essmsh` using `startMAXL.bat` (Windows) or `startMAXL.sh` (UNIX). You can pass the same arguments to `startMAXL` as you would formerly pass to `essmsh`. For example, instead of `essmsh -l username password`, you should now use `startMAXL.bat -l username password`.

esmsh(1)

NAME
   essmsh -- MaxL Shell

SYNOPSIS
   essmsh [-hlsmp] [-a | -l | file] [arguments...]

DESCRIPTION
   This document describes ways to invoke the MaxL Shell.
   The shell, invoked and nicknamed `essmsh`, takes input in the following ways: interactively (from the keyboard), standard input (piped from another program), or file input (taken from file specified on the command line).
   The MaxL Shell also accepts any number of command-line arguments, which can be used to represent any name.

OPTIONS
   essmsh accepts the following options on the command line:

   -h
      Prints this help.

   -l <user> <pwd>
      Logs in a user name and password to the local Essbase Server instance.

   -u <user>
      Specifies a user to be logged in to an Essbase Server instance. If omitted but the '-p' or '-s' flags are used, essmsh will prompt for the username.

   -p <pwd>
      Specifies a password of the user set by the '-u' option to
be logged in to an Essbase Server instance. If omitted, essmsh will prompt for the password, and the password will be hidden on the screen.

-s <server>
  Used after -l, or with [-u -p], logs the specified user into a named server. When omitted, localhost is implied.

-m <msglevel>
  Sets the level of messages returned by the shell. Values for <msglevel> are: all (the default), warning, error, and fatal.

-i
  Starts a MaxL session which reads from <STDIN>, piped in from another program. The end of the session is signalled by the EOF character in that program.

-a
  Allows a string of command-line arguments to be referenced from within the subsequent INTERACTIVE session. These arguments can be referenced with positional parameters, such as $1, $2, $3, etc. Note: omit the -a when using arguments with a file-input session.

NOTES

No option is required to pass a filename to essmsh.

Arguments passed to essmsh can represent anything: for example, a user name, an application name, or a filter name. Arguments must appear at the end of the invocation line, following `-a', `-i', or filename.

EXAMPLES

Interactive session, simplest case:
  essmsh

Interactive session, logging in a user:
  essmsh -l user pwd

Interactive session, logging user in to a server:
  essmsh -l user pwd -s server

Interactive session, logging in with two command-line arguments (referenced thereafter at the keyboard as $1 and $2):
  essmsh -l user pwd -a argument1 argument2
Interactive session, with setting the message level:
  essmsh -m error

Interactive session, hiding the password:
  essmsh -u user1
  Enter Password > *****

File-input session, simplest case:
  essmsh filename

File-input session, with three command-line arguments
(referenced anonymously in the file as $1, $2, and $3):
  essmsh filename argument1 argument2 argument3

Session reading from <STDIN>, logging into a server with two
command-line arguments:
  essmsh -l user pwd -s server -i argument1 argument2

Interactive Input

You can log into the MaxL Shell for interactive use (typing statements at the keyboard) in the following ways. See MaxL Invocation Summary for more descriptions of login flags.

No Flag

-a Flag: Arguments
-l Flag: Login
-u, -p, and -s Flags: Login Prompts and Hostname Selection
-m Flag: Message Level

No Flag

Invoked without a flag, file name, or arguments, the MaxL Shell starts in interactive mode and waits for you to log in. Note to UNIX users: In the following examples, replace startMAXL.bat with startMAXL.sh.

Example:

startMAXL.bat

login Fiona identified by sunflower;
49 - User logged in: [Fiona].

-a Flag: Arguments

With the -a flag, the MaxL Shell starts in interactive mode and accepts space-separated arguments to be referenced at the keyboard with positional parameters.

**Note:**

If interactive arguments are used with spooling turned on, variables are recorded in the log file just as you typed them (for example, $1, $2, $ARBORPATH).

Example:

```
startMAXL.bat -a Fiona sunflower appname dbsname
```

Contents of logfile createapp.out:

```
MAXL> login $1 identified by $2;
   49 - User logged in: [Fiona].

MAXL> create application $3;
   30 - Application created: ['appname'].

MAXL> create database $3.$4 as Sample.Basic;
   36 - Database created: ['appname'. 'dbsname'].

MAXL> echo $ARBORPATH;
   C:\Hyperion\products\Essbase\EssbaseClient

MAXL> spool off;
```
MAXL> create application $3;

OK/INFO - 1051061 - Application appname loaded - connection established.
OK/INFO - 1054027 - Application [appname] started with process id [404].
OK/INFO - 1056010 - Application appname created.

MAXL> create database $3.$4 as Sample.Basic;

OK/INFO - 1056020 - Database appname.dbname created.

MAXL> echo $ARBORPATH;

C:\Hyperion\products\Essbase\EssbaseClient

MAXL> spool off;

-I Flag: Login

When the -l flag is used followed by a user name and password, the MaxL Shell logs in the given user name and password and starts in interactive or non-interactive mode. The user name and password must immediately follow the -l, and be separated from it by a space.

Example:

startMAXL.bat -l Fiona sunflower

Entered at the command prompt, this starts the MaxL Shell in interactive mode and logs in user Fiona, who can henceforth issue MaxL statements at the keyboard.

-u, -p, and -s Flags: Login Prompts and Hostname Selection

The MaxL Shell can be invoked using -u and -p options in interactive mode, for passing the user name and password to the shell upon startup. To be prompted for both username and password, use the -s option with the host name of the Essbase Server.

• If -s <host-name> is passed to the shell, MaxL will prompt for the user name and password, and the password will be hidden.

startMAXL.bat -s localhost
Enter UserName> admin
Enter Password> ********

OK/INFO - 1051034 - Logging in user admin.
OK/INFO - 1051035 - Last login on Monday, January 28, 2003 10:06:16 AM.
OK/INFO - 1241001 - Logged in to Essbase.
• If -u <username> is passed to the shell and -p <password> is omitted, MaxL Shell will prompt for the password, and the password will be hidden.

```
startMAXL.bat -u smith
Enter Password > ******
```

• If -p <password> is passed to the shell and -u <username> is omitted, MaxL Shell will prompt for the user name.

```
startMAXL.bat -p password
Enter Username > smith
```

• If -m <messageLevel> is passed to the shell, only the specified level of messages will be returned by the shell.

```
startMAXL.bat -m error
```

Values for <messageLevel> include: default, all, warning, error, and fatal. The default value is all (same as specifying default).

**-m Flag: Message Level**

If -m <messageLevel> is passed to the shell, only the specified level of messages will be returned by the shell.

**Example:** `startMAXL.bat -m error`

Values for the <messageLevel> include: default, all, warning, error, and fatal. The default value is all (same as specifying default).

**File Input**

You invoke the MaxL Shell to run scripts (instead of typing statements at the keyboard) in the following ways. See MaxL Invocation Summary for a complete description of login flags.

**File Only**

If you type `startMAXL.bat` followed by a file name or path, the shell takes input from the specified file.

**Examples:**

```
startMAXL.bat C:\Hyperion\products\Essbase\EssbaseClient\scripts \filename.msh
```

Entered at the command prompt, this starts the shell, tells it to read MaxL statements from a file, and terminates the session when it is finished.

```
startMAXL.bat filename
```
Starts the shell to read MaxL statements from filename, located in the current directory (the directory from which the MaxL Shell was invoked).

**File with Arguments**

If you type `startMAXL.bat` followed by a file name followed by an argument or list of space-separated arguments, essmsh remembers the command-line arguments, which can be referenced as $1, $2, etc. in the specified file. If spooling is turned on, all variables are expanded in the log file.

**Example:**

```
D:\Scripts> startMAXL.bat filename.msh Fiona sunflower localhost
```

Starts the shell to read MaxL statements from `filename.msh`, located in the current directory.

**Contents of script filename.msh:**

```
spool on to $HOME\output\filename.out;
login $1 $2 on $3;
echo "Essbase is installed in $ESSBASEPATH";
spool off;
exit;
```

**Contents of logfile filename.out:**

```
MAXL> login Fiona sunflower on localhost;

        49 - User logged in: [Fiona].

Essbase is installed in C:\Hyperion\products\Essbase\EssbaseClient
```

**Standard Input**

With the -i flag, essmsh uses standard input, which could be input from another process. For example,

```
program.sh | startMAXL.bat -i
```

When `program.sh` generates MaxL statements as output, you can pipe `program.sh` to `startMAXL.bat` -i to use the standard output of `program.sh` as standard input for essmsh. Essmsh receives input as `program.sh` generates output, allowing for efficient co-execution of scripts.

**Example:**

```
echo login Fiona sunflower on localhost; display privilege user; | startMAXL.bat -i
```
The MaxL Shell takes input from the echo command’s output. User Fiona is logged in, and user privileges are displayed.

Login

Before you can send MaxL statements from the MaxL Shell to Essbase Server, you must log in to an Essbase Server session.

Note:

Before logging in to an Essbase Server session, you must start the MaxL Shell (see MaxL Invocation Summary). Or, you can start the MaxL Shell and log in (see -l Flag: Login) at the same time.

• USER-NAME
• PASSWORD
• HOST-NAME

Example

login admin pa5sw0rd on "https://myEssbase-myDomain.analytics.us2.example.com/essbase/agent";

Establishes a connection for user Admin.

LoginAs

To facilitate creating scheduled reports with user-appropriate permissions, administrators can log in as another user from MaxL.

Example of "log in as" statement:

loginas USER-NAME PASSWORD MIMICKED-USER-NAME [on HOST-NAME];

Example of "log in as" invocation method:

essmsh -la USER-NAME PASSWORD MIMICKED-USER-NAME [-s HOST-NAME]
Interactive example:

MAXL>loginas;
Enter UserName> username
Enter Password> password
Enter Host> machine_name
Enter UserName to Login As> mimicked_user_name

Encryption

You can encrypt user and password information stored in MaxL scripts.

The following MaxL Shell invocation generates a public-private key pair that you can use to encrypt a MaxL script.

essmsh -gk

The following MaxL Shell invocation encrypts the input MaxL script, obscuring user name and password, and changing the file extension to .mxls.

essmsh -E scriptname.mxl PUBLIC-KEY

Nested scripts are also encrypted. To avoid this and encrypt only the base script, use -Em.

The following MaxL Shell invocation decrypts and executes the MaxL script.

essmsh -D scriptname.mxls PRIVATE-KEY

The following invocation encrypts input data and returns it in encrypted form. This is useful if there is a need to manually prepare secure scripts.

essmsh -ep DATA PUBLIC-KEY

The following invocation enables you to encrypt the base script while saving any nested scripts for manual encryption.

essmsh -Em scriptname.mxl PUBLIC-KEY

Query Cancellation

You can use the Esc key to cancel a query running from MaxL Shell.

MaxL Shell Syntax Rules and Variables

The MaxL Shell (essmsh) is a pre-parser mechanism for entering MaxL statements. The following syntax information can help you use the MaxL Shell successfully.

Semicolons

Variables
Quoting and Special Characters Rules for MaxL Language

Semicolons

When a MaxL statement is passed to Essbase Server interactively or in batch mode via the MaxL Shell (essmsh), it must be terminated by a semicolon. Semicolons are used only to tell essmsh when to terminate the statement; semicolons are not part of the MaxL language itself. Therefore, when issuing MaxL statements programmatically, do not use semicolons.

Examples

Table 5-13  Semicolon Usage Examples in MaxL

<table>
<thead>
<tr>
<th>Program</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interactive MaxL Shell</td>
<td>create application Sample;</td>
</tr>
<tr>
<td>MaxL Shell script:</td>
<td>login $1 identified by $2;</td>
</tr>
<tr>
<td></td>
<td>create application Sample;</td>
</tr>
<tr>
<td></td>
<td>create currency database Sample.Interntl;</td>
</tr>
<tr>
<td></td>
<td>display database Sample.Interntl;</td>
</tr>
<tr>
<td></td>
<td>exit;</td>
</tr>
</tbody>
</table>

Variables

Overview of MaxL Shell

Environment Variables

Positional Parameters

Locally Defined Shell Variables

Quotation Marks and Variable Expansion

Exit Status Variable

Overview of MaxL Shell Variables

In the MaxL Shell, you can use variables as placeholders for any data that is subject to change or that you refer to often; for example, the name of a computer, user names, and passwords. You can use variables in MaxL scripts as well as during interactive use of the shell. Using variables in MaxL scripts eliminates the need to create many customized scripts for each user, database, or host.

Variables can be environment variables (for example, $ESSBASEPATH, which references the directory Essbase is installed to), positional parameters (for example, $1, $2, etc.), or locally defined shell variables.
All variables must begin with a $ (dollar sign). Locally defined shell variables should be set without the dollar sign, but should be referenced with the dollar sign. Example:

```
set A = val_1;
echo $A;
val_1
```

Note:

Variables can be in parentheses. Example: if $1 = arg1, then $(1)23 = arg123.

Use double quotation marks around a string when you want the string interpreted as a single token with the variables recognized and expanded. For example, "$ESSBASEPATH" is interpreted as C:\Hyperion\products\Essbase\EssbaseServer.

Use single quotation marks around a string to tell essmsh to recognize the string as a single token, without expanding variables. For example, "$ESSBASEPATH" is interpreted as $ESSBASEPATH, not C:\Hyperion\products\Essbase\EssbaseServer.

Environment Variables

You can reference any environment variable in the MaxL Shell.

Example (Windows): spool on to "$ESSBASEPATH\out.txt";

Result: MaxL Shell session is recorded to C:\Hyperion\products\Essbase\EssbaseServer\out.txt.

Example (UNIX): spool on to "$HOME/output.txt";

Result: MaxL Shell session is recorded to output.txt in the directory referenced by the $HOME environment variable.

Positional Parameters

Positional parameter variables are passed in to the shell at invocation time as arguments, and can be referred to generically by the subsequent script or interactive MaxL Shell session using $n, where n is the number representing the order in which the argument was passed on the command line.

For example, given the following invocation of the MaxL Shell,

```
essmsh filename Fiona sunflower
```

and the following subsequent login statement in that session,

```
login $1 identified by $2 on $COMPUTERNAME;
```

- $COMPUTERNAME is a Windows environment variable.
• $1 and $2 refer to the user name and password passed in as arguments at invocation time.

The values of positional parameters can be changed within a session. For example, if the value of $1 was originally Fiona (because essmsh was invoked with Fiona as the first argument), you can change it using the following syntax:

```plaintext
set 1 = arg_new;
```

**Note:**

If you nest MaxL Shell scripts or interactive sessions, the nested shell does not recognize positional parameters of the parent shell. The nested shell should be passed separate arguments, if positional parameters are to be used.

The file or process that the MaxL Shell reads from can be referred to with the positional parameter $0. Examples:

1) Invocation: essmsh filename
   $0 = filename
2) Invocation: program.sh | essmsh -i
   $0 = stdin
3) Invocation: essmsh
   $0 = null

**Locally Defined Shell Variables**

You can create variables of any name in the MaxL Shell without the use of arguments or positional parameters. These variables persist for the duration of the shell session, including in any nested shell sessions.

Example:

```plaintext
MaxL>login user1 identified by password1;
MaxL>set var1 = sample;
MaxL>echo $var1; /* see what the value of $var1 is */
  sample
MaxL>display application $var1; /* MaxL displays application "sample" */
```

**Note:**

Locally defined variables can be named using alphabetic characters, numbers, and the underscore (_). Variable values can be any characters, but take note of the usual quoting and syntax rules that apply for the MaxL Shell (see MaxL Shell Syntax Rules and Variables).
Note:

Variables defined or changed in a nested script persist into the parent script after the nested script executes.

Quotation Marks and Variable Expansion

In the following examples, assume you logged in to the MaxL Shell interactively with arguments, as follows. In addition to these examples, see Quoting and Special Characters Rules for MaxL Shell.

```
  essmsh -a Fiona sunflower sample basic login $1 $2;
```

<table>
<thead>
<tr>
<th>Example</th>
<th>Return Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>echo $1;</code></td>
<td>Fiona</td>
<td>$1 is expanded as the first invocation argument.</td>
</tr>
<tr>
<td><code>echo &quot;$1's hat&quot;;</code></td>
<td>Fiona's hat</td>
<td>$1 is expanded as the first invocation argument, and the special character ' is allowed because double quotation marks are used.</td>
</tr>
<tr>
<td><code>echo $3;</code></td>
<td>sample</td>
<td>$3 is expanded as the third invocation argument.</td>
</tr>
<tr>
<td><code>echo &quot;$3&quot;;</code></td>
<td>$3</td>
<td>$3 is taken literally and not expanded, because it is protected by single quotation marks.</td>
</tr>
<tr>
<td><code>display database $3.$4;</code></td>
<td>Database sample.basic is displayed.</td>
<td>$3 and $4 are expanded as the third and fourth invocation arguments. $3.$4 is interpreted as two tokens, which makes it suitable for DBS-NAME.</td>
</tr>
<tr>
<td><code>echo &quot;$3.$4&quot;;</code></td>
<td>sample.basic, but interpreted as one token (NOT suitable for DBS-NAME, which requires two tokens).</td>
<td>$3 and $4 are expanded as the third and fourth invocation arguments, but the entire string is interpreted as a single token, because of the double quotation marks.</td>
</tr>
</tbody>
</table>

Exit Status Variable

A successful MaxL Shell operation should have an exit status of zero. Most unsuccessful MaxL Shell operations have an exit status number, usually 1. Exit status can be referred to from within the shell, using `?`. For example,

```
MAXL> create application test1;
OK/INFO - 1051061 - Application test1 loaded - connection established.
OK/INFO - 1054027 - Application [test1] started with process id [234].
OK/INFO - 1056010 - Application test1 created.
MAXL> echo $?;
0
```
MAXL> drop application no_such;
      ERROR - 1051030 - Application no_such does not exist.
MAXL> echo $?;
2

Quoting and Special Characters Rules for MaxL Shell

These rules are for MaxL Shell commands. Applicable MaxL Shell commands include Spool on/off, Echo, and Nesting.

See Also

Quoting and Special Characters Rules for MaxL Language
Tokens enclosed in Single Quotation Marks
Tokens Enclosed in Double Quotation Marks
Use of Backslashes in MaxL
Use of Apostrophes (Single Quotation Marks)

Tokens enclosed in single quotation marks

Contents within single quotation marks are preserved as literal, without variable expansion.

Example: `echo '$3';`
Result: $3

Tokens enclosed in double quotation marks

Contents of double quotation marks are treated as a single token, and the contents are perceived as literal except that variables are expanded.

Example: `spool on to "$ESSBASEPATH\out.txt";`
Result: MaxL Shell session is recorded to C:\Hyperion\products\Essbase\EssbaseServer\out.txt.

Example: `spool on to "Ten o'clock.txt"`
Result: MaxL Shell session is recorded to a file named Ten o'clock.txt

Use of apostrophes (single quotation marks)

Preserved if enclosed in double quotation marks. Otherwise, causes a syntax error.

Example: `spool on to "Ten o'clock.txt"`
Result: MaxL Shell session is recorded to a file named Ten o'clock.txt

Use of Backslashes

Backslashes must be enclosed in single or double quotation marks because they are special characters.
One backslash is treated as one backslash by the shell, but is ignored or treated as an escape character by MaxL. Two backslashes are treated as one backslash by the shell and MaxL.

- `\" = \` (MaxL Shell)
- `\" = (nothing) (MaxL)
- `\" = \ (MaxL Shell)
- `\" = \ (MaxL)

Example: `spool on to 'D:\output.txt'`

Result: MaxL Shell records output to `D:\output.txt`.

Example: `spool on to 'D:\\output.txt'`

Result: MaxL Shell records output to `D:\output.txt`.

Example: `import database sample.basic lro from directory "$ARBORPATH\app\sample-basic-lros";`

Result: Error. Import is a MaxL statement, and for MaxL, `\"` is ignored.

Example: `import database sample.basic lro from directory "$ARBORPATH,:,:app\sample-basic-lros";`

Result: MaxL imports LRO information to Sample.Basic from `$ARBORPATH\app\sample-basic-lros`.

MaxL Shell and Unicode

MaxL Shell is in native mode when started in interactive mode.

MaxL Shell is in native mode when processing a script without a UTF8 byte header.

MaxL Shell is in UTF8 mode when processing a script with the UTF8 byte header.

MaxL Shell Command Reference

The following topics describe the MaxL Shell commands.

- Spool on/off
- Set display column width
- Set message level
- Set Timestamp
- Echo
- Nesting
- Error Checking and Branching
- Version
- Logout
- Exit
Spool on/off

Log the output of a MaxL Shell session to a file. Send standard output, informational messages, error messages, and/or warning messages generated by the execution of MaxL statements to a file.

If FILE-NAME does not exist, it is created. If FILE-NAME already exists, it is overwritten. If a directory path is not specified for FILE-NAME, FILE-NAME is created in the current directory of the MaxL Shell. Directories cannot be created using the spool command.

Message logging begins with `spool on` and ends with `spool off`.

```
spool on to 'FILE-NAME';
{MaxL statements}
spool off;
```

**Example**

```
spool on to 'output.txt';
{MaxL statements}
spool off;
```

Sends output of MaxL statements to a file called output.txt, located in the current directory where the MaxL Shell was invoked.

```
spool on to 'c:\hyperion\output.txt';
```

Sends output of MaxL statements to a file called output.txt, located in the pre-existing directory specified by an absolute path.

```
spool on to '../..../output.txt';
```

Sends output of MaxL statements to a file called output.txt, located in the pre-existing directory specified by a relative path. The file would be located three directories above the current directory.

**Description**

Most operating systems support three channels for input/output:

- STDIN (standard input channel)
- stdout (standard output channel)
- stderr (standard error channel)
Most operating systems also provide command-line options for re-directing data generated by applications, depending on which of the above channels the data is piped through.

Errors in MaxL are flagged as STDERR, allowing command-line redirection of errors using operating-system redirection handles. Non errors are flagged as STDOUT; thus normal output may be logged separately from error output. Here is an example of redirecting error-output at invocation time:

`essmsh script.mxl 2>errorfile.err`

**Note:**
Operating-system redirection handles vary; check the platform documentation.

You can also redirect STDERR and STDOUT independently to different MaxL output logs, using the corresponding options in the `spool` command. For example, you can direct errors to one file and output to another by placing the following lines in your script:

```plaintext
spool stdout on to 'output.txt';
spool stderr on to 'errors.txt';
```

or you can direct errors only:

```plaintext
spool stderr on to 'errors.txt';
```

or you can direct output only:

```plaintext
spool stdout on to 'output.txt';
```

**Note:**
You cannot use the generic spool and the special output-channel spools in the same script. For example, the following is not valid:

```plaintext
spool on to 'session.txt';
spool stderr on to 'errors.txt';
```

**Set Display Column Width**

Set the width of the columns that appear in MaxL display output tables, for the current MaxL Shell session.

- Default: 20 characters
• Minimum: 8 characters
• Maximum: None.

```plaintext
set column_width default;
```

**COLUMN-WIDTH**

**Example**

```plaintext
set column_width 10;
```

Sets the column width to 10 characters.

```plaintext
set column_width default;
```

Sets the column width back to 20 characters.

**Set Message Level**

Set the level of messaging you want returned from MaxL Shell sessions. By default, all messages are returned.

```plaintext
set message level all;
```

**Table 5-15  MaxL Shell Message Levels**

<table>
<thead>
<tr>
<th>Message level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>all</td>
<td>Errors, warnings, status reporting, and informational messages. This is the default message level.</td>
</tr>
<tr>
<td>error</td>
<td>Essbase and MaxL Shell error messages.</td>
</tr>
<tr>
<td>warning</td>
<td>Essbase warning messages.</td>
</tr>
<tr>
<td>fatal</td>
<td>Only errors which cause the shell to disconnect from Essbase.</td>
</tr>
</tbody>
</table>

**Example**

```plaintext
set message level all;
```
Set Timestamp

Enable or disable the display of a timestamp after execution of each MaxL statement. By default, no timestamps are returned.

```
set timestamp on
off
```

Notes

The timestamp information does not display after the error-control shell statements goto, iferror, and define.

Example

```
set timestamp on;
```

Echo

Display text or expand variables to the screen or to a log file. When used in scripts with spooling (log-file generation) turned on, echo expands variables in the log file. For interactive sessions, variables are not expanded in the log file; instead, the variable name you typed is recorded (for example, $1).

Syntax

```
echo <text> | <variablename>
```

Example

See examples of echo under the discussion of variables (Quotation Marks and Variable Expansion).

Nesting

Reference (include) a MaxL script from within another MaxL script. You might use this if variables are defined in the referenced MaxL script which are useful to the current MaxL script.

Syntax

```
msh <scriptfile>;
```

Example

```
login fiona sunflower;
alter database sample.basic end archive;
msh calculate.msh;
alter database sample.basic
begin archive to file bak;
```
logout;

Note:
Variables defined or changed in a nested script persist into the parent script after the nested script executes.

Note:
Because msh is a shell command, it is limited to the originating session. Therefore, you should not reference MaxL scripts that contain new login statements.

Error Checking and Branching

IfError instructs the MaxL Shell to respond to an error in the previous statement by skipping subsequent statements, up to a certain location in the script that is defined by a label name.

IfError checks the presence of errors only in the precedent statement. IfError checks for:

- Errors in MaxL statement execution
- Errors in MaxL Shell command execution, including:
  - Errors in spool on/off, such as permission errors
  - Errors in set column_width, such as invalid widths
  - Errors in script nesting, such as permission errors or nonexistent include files

Goto forces the MaxL Shell to branch to a certain location in the script defined by a label name; goto is not dependent on the occurrence of an error.

Syntax

iferror LABELNAME
goto LABELNAME
define label LABELNAME

Example: Iferror (MaxL)

The following example script contains a dimension build statement and a data load statement. If the dimension build fails, the data load is skipped.

login $1 $2;

import database sample.basic dimensions
from data_file 'C:\data\dimensions.txt'
using rules_file 'C:\data\rulesfile.rul'
Example: Iferror (MaxL Shell)

The following example script tests various errors including MaxL Shell errors, and demonstrates how you can set the exit status variable to a nonzero argument to return an exit status to the MaxL Shell.

```maxl
### Begin Script ###

login $1 $2;
echo "Testing syntactic errors...";

spool on to spool.out;
set timestampTypo on;
iferror 'End';

msh "doesnotexistlerr.mxl";
iferror 'FileDoesNotExistError';

echo "Script completed successfully...";
spool off;
logout;
exit 0;

define label 'FileDoesNotExistError';
echo "Error detected: Script file does not exist";
spool off;
logout;
exit 1;

define label 'ShellError';
echo ' Shell error detected...';
spool off;
logout;
exit 2;

define label 'End';
echo ' Syntax error detected...';
spool off;
logout;
exit 3;
```

---

Chapter 5
MaxL Shell Commands

5-193
### Example: Goto

The following example script contains a dimension build statement and a data load statement. Goto is used to skip the data load.

```plaintext
login $1 $2;
import database sample.basic dimensions from data_file 'C:\data\dimensions.txt' using rules_file 'C:\\data\\rulesfile.rul' on error append to 'C:\\logs\dimbuild.log';
goto 'Finished';
import database sample.basic data from data_file "$ARBORPATH\app\sample\basic\calcdat.txt" on error abort;
define label 'Finished';
exit;
```

**Notes**

The MaxL Shell will skip forward in the script to LABELNAME but not backwards.

### Version

To see which version of MaxL you are using, type `version`.

**Example**

```plaintext
version;
```

**Returns**

```plaintext
Essbase MaxL Shell - Release 11.1.2
Copyright (c) 2000, 2010, Oracle and/or its affiliates.
All rights reserved.
MAXL>
```

### Logout

Log out from Essbase without exiting the interactive MaxL Shell.

**Syntax**

```plaintext
logout;
```
Example

logout;

Exit

Exit from the \texttt{MAXL}> prompt after using interactive mode. You can optionally set the exit status variable to a non zero argument to return an exit status to the parent shell.

\begin{quote}
\textbf{Note:}

It is not necessary to exit at the end of MaxL script files or stream-oriented input (using the \texttt{-i} switch).
\end{quote}

Syntax

exit;

Example

exit;

closes the MaxL Shell window or terminal.

exit 10;

closes the MaxL Shell window or terminal with a return status of 10. You can use this in combination with \texttt{IfError} to return a non zero error status to the parent shell.

ESSCMD Script Conversion

\texttt{cmd2mxl} is a fully supported utility for converting existing ESSCMD shell scripts to their corresponding MaxL scripts. To convert an ESSCMD shell script to a MaxL script, go to the operating-system command prompt and enter the executable name, the ESSCMD shell script name, the desired MaxL script name, and the name of a logfile to write to in case of errors.

- ESSCMD Script Utility Usage
- Things to Note About the ESSCMD shell Script Utility
- ESSCMD to MaxL Mapping

ESSCMD Script Utility Usage

\texttt{cmd2mxl esscmd\_script maxl\_output logfile}

For example, if the ESSCMD shell script name is \texttt{%ARBORPATH%\dailyupd.scr}, the command issued on the operating-system command line would be:
Subsequently, the MaxL script can be executed using the MaxL Shell by the following command:

\texttt{essmsh \%ARBORPATH\%\dailyupd.mxl}

**Things to Note About the ESSCMD shell Script Utility**

1. The utility will only translate syntactically and semantically valid ESSCMD shell scripts.
2. For invalid ESSCMD shell scripts, the resulting MaxL script is undefined.
3. All ESSCMD shell statements in the scripts should end with a semicolon ( ; ) statement terminator.
4. This utility will only work on Windows platforms.
5. Although most ESSCMD shell commands have corresponding MaxL statements, there are exceptions. For such exceptions, a comment will be generated in the logfile, and the resulting MaxL script will have to be modified to work correctly. Note that if an ESSCMD shell command is still needed, it can be invoked from a MaxL script using `shell esscmd <scriptname>`.
6. All strings in the ESSCMD shell scripts should be surrounded by double quotation marks ("").

**ESSCMD to MaxL Mapping**

The following table compares ESSCMD shell usage to MaxL usage, and the following conversions are supported by \texttt{cmd2mxl}.

<table>
<thead>
<tr>
<th>ESSCMD shell Command</th>
<th>ESSCMD shell Usage Example</th>
<th>MaxL Equivalent Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDUSER</td>
<td>ADDUSER finance essexer1;</td>
<td>N/A. User management statements no longer supported in MaxL.</td>
</tr>
<tr>
<td>BEGINARCHIVE</td>
<td>beginarchive sample basic &quot;test.txt&quot;;</td>
<td>alter database Sample.Basic begin archive to file &quot;test.txt&quot;;</td>
</tr>
<tr>
<td>BEGININCBUILDDIM</td>
<td>beginincbuilddim;</td>
<td>import database Sample.Basic dimensions from local text data_file 'c:\data.txt' using local rules_file 'c:\data_rule.rul' on error write to 'c:\error.log';</td>
</tr>
<tr>
<td>BUILDDIM</td>
<td>builddim 1 &quot;c:\data_rul.rul&quot; 3 &quot;c:\data.txt&quot; 4 &quot;c:\error.log&quot;;</td>
<td>Same as BEGININCBUILDDIM</td>
</tr>
<tr>
<td>CALC</td>
<td>calc &quot;CALC ALL;&quot;;</td>
<td>execute calculation 'CALC ALL' on sample.basic;</td>
</tr>
<tr>
<td>CALCDEFAULT</td>
<td>calcdefault;</td>
<td>execute calculation default on Sample.Basic;</td>
</tr>
<tr>
<td>CALCLINE</td>
<td>calcline &quot;CALC ALL;&quot;;</td>
<td>execute calculation 'CALC ALL;' on sample.basic;</td>
</tr>
<tr>
<td>COPYAPP</td>
<td>copyapp sample sampnew;</td>
<td>create application sampnew as sample;</td>
</tr>
<tr>
<td>ESSCMD shell Command</td>
<td>ESSCMD shell Usage Example</td>
<td>MaxL Equivalent Example</td>
</tr>
<tr>
<td>----------------------</td>
<td>---------------------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>COPYDB</td>
<td><code>copydb sample basic sample basic2;</code></td>
<td>create or replace database sample.basic2 as sample.basic;</td>
</tr>
<tr>
<td>COPYFILTER</td>
<td><code>copyfilter sample basic westwrite sample basic westmgr;</code></td>
<td>create filter sample.basic.westmgr as sample.basic.westwrite;</td>
</tr>
<tr>
<td>COPYOBJECT</td>
<td><code>copyobject &quot;9&quot; &quot;sample&quot; &quot;basic&quot; &quot;calcdat&quot; &quot;sample&quot; &quot;basic&quot; &quot;calcdat2&quot;;</code></td>
<td>alter object sample.basic.calcdat of type text copy to 'sample.basic.calcdat2';</td>
</tr>
<tr>
<td>CREATEAPP</td>
<td><code>createapp finance;</code></td>
<td>create or replace application finance;</td>
</tr>
<tr>
<td>CREATEDB</td>
<td><code>createdb finance investor;</code></td>
<td>create or replace database finance.investor;</td>
</tr>
<tr>
<td>CREATEGROUP</td>
<td><code>creategroup managers;</code></td>
<td>N/A. User management statements no longer supported in MaxL.</td>
</tr>
<tr>
<td>CREATELOCATION</td>
<td><code>select sample basic;</code></td>
<td>alter system load application sample;</td>
</tr>
<tr>
<td></td>
<td><code>createlocation hq hqserver finance investor admin password;</code></td>
<td>alter application sample load database basic;</td>
</tr>
<tr>
<td></td>
<td><code>alter system load application sample;</code></td>
<td>create location alias hq from sample.basic to finance.investor at hqserver as admin identified by 'password';</td>
</tr>
<tr>
<td>CREATEUSER</td>
<td><code>createuser karen password;</code></td>
<td>N/A. User management statements no longer supported in MaxL.</td>
</tr>
<tr>
<td>CREATEVARIABLE</td>
<td><code>createvariable CurMnth localhost sample basic Jan;</code></td>
<td>alter database sample.basic add variable CurMnth 'Jan';</td>
</tr>
<tr>
<td></td>
<td><code>alter application sample add variable CurMnth 'Jan';</code></td>
<td>alter system add variable CurMnth 'Jan';</td>
</tr>
<tr>
<td>DELETEAPP</td>
<td><code>deleteapp sampnew;</code></td>
<td>drop application sampnew cascade;</td>
</tr>
<tr>
<td>DELETEDB</td>
<td><code>deletedb demo basic;</code></td>
<td>drop database demo.basic;</td>
</tr>
<tr>
<td>DELETEGROUP</td>
<td><code>deletegroup engg;</code></td>
<td>N/A. User management statements no longer supported in MaxL.</td>
</tr>
<tr>
<td>DELETELOCATION</td>
<td><code>select finance investor;</code></td>
<td>alter system load application finance;</td>
</tr>
<tr>
<td></td>
<td><code>deletelocation hq1;</code></td>
<td>alter application finance load database investor;</td>
</tr>
<tr>
<td></td>
<td><code>drop location alias finance.investor.hq1;</code></td>
<td></td>
</tr>
<tr>
<td>DELETELOG</td>
<td><code>deletelog sample;</code></td>
<td>alter application sample clear logfile;</td>
</tr>
<tr>
<td>DELETEUSER</td>
<td><code>deleteuser rob;</code></td>
<td>N/A. User management statements no longer supported in MaxL.</td>
</tr>
<tr>
<td>DELETEVARIABLE</td>
<td><code>select sample basic;</code></td>
<td>alter system load application sample;</td>
</tr>
<tr>
<td></td>
<td><code>deletevariable CurMnth &quot;localhost&quot;;</code></td>
<td>alter application sample load database basic;</td>
</tr>
<tr>
<td></td>
<td><code>alter database sample.basic drop variable CurMnth;</code></td>
<td>alter application sample drop variable CurMnth;</td>
</tr>
<tr>
<td></td>
<td><code>alter system drop variable CurMnth;</code></td>
<td></td>
</tr>
<tr>
<td>ESSCMD shell Command</td>
<td>ESSCMD shell Usage Example</td>
<td>MaxL Equivalent Example</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------</td>
<td>---------------------------------------------</td>
</tr>
<tr>
<td>DISABLELOGIN</td>
<td>disablelogin demo;</td>
<td>alter application demo disable connects;</td>
</tr>
<tr>
<td>DISPLAYALIAS</td>
<td>select sample basic; displayalias &quot;default&quot;;</td>
<td>query database sample.basic list alias_names in alias_table 'Default';</td>
</tr>
<tr>
<td>ENABLELOGIN</td>
<td>enablelogin demo;</td>
<td>alter application demo enable connects;</td>
</tr>
<tr>
<td>ENDARCHIVE</td>
<td>endarchive sample basic;</td>
<td>alter database sample.basic end archive;</td>
</tr>
<tr>
<td>ENDINCBUILDDIM</td>
<td>ENDINCBUILDDIM;</td>
<td>See BEGININCBUILDDIM</td>
</tr>
<tr>
<td>ESTIMATEFULDBSIZE</td>
<td>select sample basic; estimatefulldbsize;</td>
<td>query database sample.basic get estimated size;</td>
</tr>
<tr>
<td>EXIT</td>
<td>exit;</td>
<td>exit;</td>
</tr>
<tr>
<td>EXPORT</td>
<td>select sample basic; export &quot;c:\data.txt&quot; 1;</td>
<td>alter system load application sample; alter application sample load database basic; export database Sample.Basic all data to data_file 'c:\data.txt';</td>
</tr>
<tr>
<td>GETALLREPLCELLS</td>
<td>select samppart company; getallreplcells &quot;svr2&quot; &quot;sampeast&quot; &quot;east&quot;;</td>
<td>alter system load application samppart; alter application samppart load database company; refresh replicated partition samppart.company from sampeast.east at svr2;</td>
</tr>
<tr>
<td>GETAPPINFO</td>
<td>getappinfo &quot;demo&quot;;</td>
<td>display application demo;</td>
</tr>
<tr>
<td>GETAPPSTATE</td>
<td>getappstate demo;</td>
<td>display application demo;</td>
</tr>
<tr>
<td>GETATTRIBUTESPECS</td>
<td>select sample basic; getattributespecs;</td>
<td>query database sample.basic get attribute_spec;</td>
</tr>
<tr>
<td>GETATTRINFO</td>
<td>select sample basic; getattrinfo &quot;Caffeinated_True&quot;;</td>
<td>query database sample.basic get attribute_info 'Caffeinated_True';</td>
</tr>
<tr>
<td>GETDBINFO</td>
<td>select sample basic; getdbinfo;</td>
<td>display database sample.basic request_history;</td>
</tr>
<tr>
<td>GETDBSTATE</td>
<td>getdbstate sample basic;</td>
<td>display database sample.basic;</td>
</tr>
<tr>
<td>GETDBSTATS</td>
<td>select sample basic; getdbstats;</td>
<td>query database sample.basic get dbstats data_block;</td>
</tr>
<tr>
<td>GETCRRATE</td>
<td>getcrrate;</td>
<td>query database sample.basic get currency_rate;</td>
</tr>
<tr>
<td>GETDEFAULTCALC</td>
<td>select sample basic; getdefaultcalc;</td>
<td>query database sample.basic get default calculation;</td>
</tr>
<tr>
<td>GETMBRCALC</td>
<td>select sample basic; getmbrcalc &quot;Profit %&quot;;</td>
<td>query database sample.basic get member_calculation 'Profit %';</td>
</tr>
<tr>
<td>GETMBRINFO</td>
<td>select sample basic; getmbrinfo &quot;Ounces_20&quot;;</td>
<td>query database sample.basic get member_info 'Ounces_20';</td>
</tr>
</tbody>
</table>
Table 5-16  (Cont.) ESSCMD shell to MaxL Mapping

<table>
<thead>
<tr>
<th>ESSCMD shell Command</th>
<th>ESSCMD shell Usage Example</th>
<th>MaxL Equivalent Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>GETPERFSTATS</td>
<td>select sample basic; getperfstats;</td>
<td>query database sample.basic get performance statistics kernel_cache table;</td>
</tr>
<tr>
<td>GETUPDATEDREPLCELLS</td>
<td>See GETALLREPLCELLS</td>
<td>See GETALLREPLCELLS</td>
</tr>
<tr>
<td>GETUSERINFO</td>
<td>getuserinfo admin;</td>
<td>display user admin;</td>
</tr>
<tr>
<td>GETVERSION</td>
<td>getversion;</td>
<td>version;</td>
</tr>
<tr>
<td>IMPORT</td>
<td>select sample basic; import 1 &quot;c:\data.txt&quot; 4 y 3 &quot;c:\import.rul&quot; n &quot;c:\data_load.err&quot;;</td>
<td>alter system load application sample;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>alter application sample load database basic;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>import database sample.basic data from local text data_file 'c:\data.txt' using local_rules_file 'c:\data_rule.rul' on error write to 'c:\data_load.err';</td>
</tr>
<tr>
<td>INCBUILD DIM</td>
<td>See BEGININCBUILDDIM</td>
<td>See BEGININCBUILDDIM</td>
</tr>
<tr>
<td>LISTALIASES</td>
<td>select sample basic; listaliases;</td>
<td>query database sample.basic list alias_table;</td>
</tr>
<tr>
<td>LISTAPP</td>
<td>listapp;</td>
<td>display application all;</td>
</tr>
<tr>
<td>LISTDB</td>
<td>listdb;</td>
<td>display database all;</td>
</tr>
<tr>
<td>LISTFILES</td>
<td>listfiles &quot;&quot;&quot;sample&quot;&quot; basic&quot;&quot;</td>
<td>query database sample.basic list all file information;</td>
</tr>
<tr>
<td>LISTFILTERS</td>
<td>listfilters sample basic;</td>
<td>display filter on database Sample.Basic;</td>
</tr>
<tr>
<td>LISTGROUPS</td>
<td>listgroups;</td>
<td>display group all;</td>
</tr>
<tr>
<td>LISTGROUPUSERS</td>
<td>listgroupusers finance;</td>
<td>display user in group finance;</td>
</tr>
<tr>
<td>LISTLINKEDOBJECTS</td>
<td>select sample basic; listlinkedobjects &quot;Fiona&quot; &quot;07/07/2003&quot;;</td>
<td>query database sample.basic list lro by Fiona before '07/07/2003';</td>
</tr>
<tr>
<td>LISTLOCATIONS</td>
<td>select sample basic; listlocations;</td>
<td>alter system load application sample;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>alter application sample load database basic;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>display location alias on database sample.basic;</td>
</tr>
<tr>
<td>LISTLOCKS</td>
<td>listlocks;</td>
<td>display lock;</td>
</tr>
<tr>
<td>LISTLOGINS</td>
<td>listlogins;</td>
<td>display session all;</td>
</tr>
<tr>
<td>LISTOBJECTS</td>
<td>listobjects &quot;2&quot; &quot;Sample&quot; &quot;Basic&quot;;</td>
<td>display object of type calc_script on database sample.basic;</td>
</tr>
<tr>
<td>LISTUSERS</td>
<td>listusers;</td>
<td>display user all;</td>
</tr>
<tr>
<td>LISTVARIABLES</td>
<td>listvariables localhost sample basic;</td>
<td>display variable on database sample.basic;</td>
</tr>
<tr>
<td>LOADALIAS</td>
<td>loadalians &quot;special_flavors&quot; &quot;C:\Hyperion\products\Essbase \EssbaseServer\apps\sample\basic \seasonal.txt&quot;;</td>
<td>alter database sample.basic load alias_table 'special_flavors' from data_file &quot;$ARBORPATH\app\sample\basic\seasonal.txt&quot;;</td>
</tr>
<tr>
<td>LOADAPP</td>
<td>loadapp sample;</td>
<td>alter system load application sample;</td>
</tr>
<tr>
<td>ESSCMD shell Command</td>
<td>ESSCMD shell Usage Example</td>
<td>MaxL Equivalent Example</td>
</tr>
<tr>
<td>-------------------------</td>
<td>--------------------------------------------------------</td>
<td>----------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>LOADDDB</td>
<td>loaddb sample basic;</td>
<td>alter application sample load database basic;</td>
</tr>
<tr>
<td>LOADDATA</td>
<td>select sample basic;</td>
<td>alter system load application sample;</td>
</tr>
<tr>
<td></td>
<td>loaddata 3 &quot;c:\data.txt&quot;;</td>
<td>alter application sample load database basic;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>import database sample.basic data from local text data_file 'c:\data.txt' on error abort;</td>
</tr>
<tr>
<td>LOGIN</td>
<td>login local admin password;</td>
<td>login admin 'password' on local;</td>
</tr>
<tr>
<td>LOGOUT</td>
<td>logout;</td>
<td>logout;</td>
</tr>
<tr>
<td>LOGOUTALLUSERS</td>
<td>logoutallusers y;</td>
<td>alter system logout session all;</td>
</tr>
<tr>
<td>LOGOUTUSER</td>
<td>Available only in interactive ESSCMD shell sessions.</td>
<td>alter system logout session 4294967295;</td>
</tr>
<tr>
<td>OUTPUT</td>
<td>output 1 c:\test.log;</td>
<td>spool on to 'c:\test.log';</td>
</tr>
<tr>
<td></td>
<td>output 4;</td>
<td>spool off;</td>
</tr>
<tr>
<td>PURGELINKEDOBJECTS</td>
<td>purgelinkedobjects &quot;Fiona&quot; &quot;07/07/2002&quot;;</td>
<td>alter database sample.basic delete lro by 'fiona' before '07/07/2002';</td>
</tr>
<tr>
<td>PUTALLREPLCELLS</td>
<td>select sampeast east;</td>
<td>alter system load application sampeast;</td>
</tr>
<tr>
<td></td>
<td>putallreplcells svr1 samppart company;</td>
<td>alter application sampeast load database east;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>refresh replicated partition sampeast.east from samppart.company at svr1 updated data;</td>
</tr>
<tr>
<td>PUTUPDATEDREPLCELLS</td>
<td>See PUTALLREPLCELLS</td>
<td>See PUTALLREPLCELLS</td>
</tr>
<tr>
<td>REMOVELOCKS</td>
<td>removelocks &quot;2&quot;;</td>
<td>drop lock held by Fiona;</td>
</tr>
<tr>
<td>REMOVEUSER</td>
<td>removeuser finance steve;</td>
<td>N/A. User management statements no longer supported in MaxL.</td>
</tr>
<tr>
<td>RENAMEAPP</td>
<td>renameapp sample newsamp1;</td>
<td>alter application sample rename to newsamp1;</td>
</tr>
<tr>
<td>RENAMEDB</td>
<td>renamedb sample basic newbasic;</td>
<td>alter database sample.basic rename to newbasic;</td>
</tr>
<tr>
<td>RENAMEFILTER</td>
<td>renamefilter sample basic westmgr allwest;</td>
<td>create or replace filter sample.basic.westmgr as sample.basic.allwest;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>drop filter sample.basic.westmgr;</td>
</tr>
<tr>
<td>RENAMEOBJECT</td>
<td>RENAMEOBJECT &quot;9&quot; &quot;sample&quot; &quot;basic&quot; &quot;calcdat&quot; &quot;calcdat2&quot;;</td>
<td>alter object sample.basic.calcdat of type text rename to 'calcdat2';</td>
</tr>
<tr>
<td>RENAMEUSER</td>
<td>renameuser steve_m m_steve;</td>
<td>N/A. User management statements no longer supported in MaxL.</td>
</tr>
<tr>
<td>RESETDB</td>
<td>select sample basic;</td>
<td>alter database sample.basic reset;</td>
</tr>
<tr>
<td>RESETPERFSTATS</td>
<td>resetperfstats enable;</td>
<td>alter database sample.basic set performance statistics enabled;</td>
</tr>
</tbody>
</table>
### Table 5-16  (Cont.) ESSCMD shell to MaxL Mapping

<table>
<thead>
<tr>
<th>ESSCMD shell Command</th>
<th>ESSCMD shell Usage Example</th>
<th>MaxL Equivalent Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>RUNCALC</td>
<td>The only command supported is the server based calc script execution. Select Sample.Basic; Runcalc 2 one;</td>
<td>execute calculation Sample.Basic.one;</td>
</tr>
<tr>
<td>RUNREPT</td>
<td>select sample basic; runrept 2 complex &quot;c:\complex.out&quot;;</td>
<td>alter system load application sample; alter application load database basic; export database sample.basic using server report_file 'complex' to data_file 'c:\complex.out';</td>
</tr>
<tr>
<td>SELECT</td>
<td>select sample basic;</td>
<td>alter system load application sample; alter application load database basic;</td>
</tr>
<tr>
<td>SETALIAS</td>
<td>select sample basic; setalias &quot;long names&quot;;</td>
<td>alter database sample.basic set active alias_table 'Long Names';</td>
</tr>
<tr>
<td>SETAPPSTATE</td>
<td>setappstate sample &quot;&quot; &quot;&quot; y y y y y y 1000 1000;</td>
<td>alter application sample enable startup; alter application sample enable autostartup; alter application sample set minimum permission manager; alter application sample enable connects; alter application sample enable commands; alter application sample enable updates; alter application sample enable security; alter application sample set lock_timeout after 1000 seconds; alter application sample set max_lro_file_size 1000 kb;</td>
</tr>
<tr>
<td>ESSCMD shell Command</td>
<td>ESSCMD shell Usage Example</td>
<td>MaxL Equivalent Example</td>
</tr>
<tr>
<td>-----------------------</td>
<td>---------------------------</td>
<td>-------------------------</td>
</tr>
<tr>
<td>SETDBSTATE</td>
<td><code>setdbstate &quot;Y&quot; &quot;Y&quot; 4 3145728 &quot;Y&quot; &quot;Y&quot; &quot;Y&quot; 0 1048576 1025 &quot;Y&quot;;</code></td>
<td><code>alter database sample.basic enable startup;</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>alter database sample.basic enable autostartup;</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>alter database sample.basic set minimum permission manager;</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>alter database sample.basic set data_cache_size 3145728;</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>alter database sample.basic enable aggregate_missing;</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>alter database sample.basic enable two_pass_calc;</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>alter database sample.basic enable create_blocks;</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>alter database sample.basic set currency_conversion division;</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>alter database sample.basic set index_cache_size 1048576;</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>alter database sample.basic enable compression;</code></td>
</tr>
<tr>
<td>SETDBSTATEITEM</td>
<td>.</td>
<td>See the <code>alter database</code> statement.</td>
</tr>
<tr>
<td>SETDEFAULTCALC</td>
<td><code>select sample basic;</code></td>
<td><code>alter database sample.basic set default calculation as 'CALC ALL';</code></td>
</tr>
<tr>
<td></td>
<td><code>setdefaultcalc &quot;CALC ALL;&quot;;</code></td>
<td></td>
</tr>
<tr>
<td>SETDEFAULTCALCFIELD</td>
<td><code>select sample basic;</code></td>
<td>Create a calculation file in the server containing the calculation string. Then, <code>alter database sample.sasic set default calculation sample.basic.defcalc;</code> will set the default calculation.</td>
</tr>
<tr>
<td></td>
<td><code>setdefaultcalcfile defcalc;</code></td>
<td></td>
</tr>
<tr>
<td>SETMSGLEVEL</td>
<td><code>setmsglevel 2;</code></td>
<td><code>set message level all;</code></td>
</tr>
</tbody>
</table>
### Table 5-16  (Cont.) ESSCMD shell to MaxL Mapping

<table>
<thead>
<tr>
<th>ESSCMD shell Command</th>
<th>ESSCMD shell Usage Example</th>
<th>MaxL Equivalent Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>SETPASSWORD</td>
<td>setpassword steve newpass;</td>
<td>N/A. User management statements no longer supported in MaxL.</td>
</tr>
<tr>
<td>SHUTDOWNSERVER</td>
<td>shutdownserver local admin password;</td>
<td>login admin 'password' on local; alter system shutdown;</td>
</tr>
<tr>
<td>SLEEP</td>
<td>sleep 10;</td>
<td>shell sleep 10;</td>
</tr>
<tr>
<td>UNLOADALIAS</td>
<td>select sample basic; unloadalias &quot;flavors&quot;;</td>
<td>alter database sample.basic unload alias_table 'flavors';</td>
</tr>
<tr>
<td>UNLOADAPP</td>
<td>unloadapp sample;</td>
<td>alter system unload application sample;</td>
</tr>
<tr>
<td>UNLOADDB</td>
<td>unloaddb sample basic;</td>
<td>alter application sample unload database basic;</td>
</tr>
<tr>
<td>UNLOCKOBJECT</td>
<td>unlockobject &quot;1&quot; &quot;sample&quot; &quot;basic&quot; &quot;basic&quot;;</td>
<td>alter object 'sample.basic.basic' of type outline unlock;</td>
</tr>
<tr>
<td>UPDATE</td>
<td>select sample.basic update &quot;Jan Sales '100-10' Florida Actual 220&quot;;</td>
<td>import database sample.basic from data_string 'Jan Sales 100-10 Florida Actual 220';</td>
</tr>
<tr>
<td>UPDATEFILE</td>
<td>updatefile 3 &quot;c:\data.txt&quot; 1;</td>
<td>same as LOADDATA;</td>
</tr>
<tr>
<td>UPDATEVARIABLE</td>
<td>updatevariable hot_product local sample basic &quot;100-10&quot;;</td>
<td>alter system set variable 'hot_product' '100-10'; alter application sample set variable 'hot_product' '&quot;100-10&quot;'; alter database Sample.Basic set variable 'hot_product' '&quot;100-10&quot;';</td>
</tr>
<tr>
<td>VALIDATE</td>
<td>validate;</td>
<td>alter database sample.basic validate data to local logfile 'validation.txt';</td>
</tr>
</tbody>
</table>

### Reserved Words List

The following keywords are part of the MaxL DDL grammar, and are reserved. If you intend to use any of these words as names or passwords, you must enclose the word in single quotation marks.

- `abort`
- `absolute_value`
- `account_type`
- `active`
- `add`
- `administrator`
- `advanced`
- `after`
- `aggregate`
- `aggregates`
- `aggregate_assume_equal`
- `aggregate_missing`
- `aggregate_storage`
- `aggregate_sum`
aggregate_view
aggregate_use_last
algorithm
alias
alias_names
alias_table
all
all_users_groups
allocation
alloc_rule
allow
allow_merge
alter
alternate_rollups
amount
amountcontext
amounttimespan
any
append
application
application_access_type
apply
archive
archive_file
area
as
aso_level_info
at
attribute
attribute_calc
attribute_info
attribute_spec
attribute_to_base_member_association
auto_password
autostartup
b
backup_file
based
basis
basistimespan
basistimespanoptions
before
begin
bitmap
blocks
buffer_id
buffered
build
by
cache_pinning
cache_size
calc_formula
calc_script
calc_string
calculation
cascade
cell_status
change_file
clear
client
cnt_sempaphore
column_width
columns
combinebasis
commands
comment
commitblock
committed_mode
compact
compression
compression_info
config_values
connect
connects
consolidation
copy
copy_subvar
copy_useraccess
create
create_application
create_blocks
create_user
creation
creation_user
creditmember
cube_size_info
currency
currency_category
currency_conversion
currency_database
currency_member
currency_rate
custom
data
data_block
data_cache_size
data_file
data_file_cache_size
data_storage
data_string
database
database_synch
database_asynch
days
dbstats
debitmember
debug
default
definition_only
definitions
delete
designer
destroy
dimension
dimensions
direct
direction
directory
disable
disabled
disallow
discard_errors
disk
display
divideamount
division
drillthrough
dml_output
drop
dump
dynamic_calc
eas_loc
enable
enabled
encrypted
end
end_transaction
enforce
eqd
error
error_file
errors_to_highest
errors_to_location
errors_to_lowest
estimated
event
exact
excel
exceeds
excludedrange
execute
existing_views
export
export_directory
external
failed_sss_migration
fragmentation_percent
freespace
from
file
file_location
file_size
file_type
filter
filter_access
fixed_decimal
for
force
force_dump
formatted_value
function
gb
get
get_missing_cells
get_meaningless_cells
global
grant
group
group_id
ha_trace
held
high
hostname
identified
identify
ignore_missing_values
ignore_zero_values
immediate
implicit_commit
import
in
inactive
inactive_user_days
including
incremental
index
index_cache_size
index_data
index_page_size
information
initialize
input
instead
invalid_block_headers
invalid_login_limit
io_access_mode
kb
kernel_io
kernel_cache
kill
level
level0
license_info
linked
list
load
load_buffer
load_buffers
load_buffer_block
local
location
lock
lock_timeout
locked
log_level
logfile
login
logout
long
lotus_2
lotus_3
lotus_4
low
lro
macro
manager
mapped
max_disk_size
max_file_size
max_lro_file_size
mb
medium
member
member_alias_namespace
member_calculation
member_comment
member_data
member_fixed_length_data
member_formula
member_info
member_name_namespace
member_property
member uda
member uda namespace
member_variable_length_data
merge
meta_read
metadata_only
migr_modified_access
miner
minimum
mining
minutes
missing_value
mode
model
move
multiple
multiplication
mutex
name
negativebasisoptions
never
no_access
none
non_unique_members
nonunicode_mode
note
nothing
numerical_display
object
objects
of
off
offset
on
only
opg_cache
opg_state
optional
optional_group
options
or
outline
outline_id
outline_paging_file
output
override
overview
partition
partition_file
partition_size
passive
password
password_reset_days
performance
permission
persistence
perspective
physical
pmml_file
ports
pov
pre_image_access
precision
preserve
preserve_groups
private
privilege
process
project
property
protocol
purge
query
query_data
query_tracking
range
read
recover
reference_cube
reference_cube_reg
refresh
region
registration
reregister
remote
remove
remove_zero_cells
rename
repair
repeatamount
replace
replay
replicated
replication_assume_identical
report_file
request
request_history
request_id
reset
resource_usage
restore
restructure
result
resync
retrieve_buffer_size
retrieve_sort_buffer_size
reverse
revoke
rle
row
rows
rules_file
runtime
runtime_info
save
scientific_notation
scope
score
script_file
seconds
security
security_backup
select
selecting
selection
self_session_info
semaphore
sequence_id_range
server
server_port
session
session_idle_limit
session_idle_poll
set
shared_services_native
short
shutdown
single
singlecell
size
size_limit
skip_to_next_amount
skip_missing
skip_negative
skip_zero
slice
sourceregion
spec
spinlock
splitbasis
spread
SSL
sss
sss_mode
sss_name
starting
startup
statistics
status
stop
stopping
storage
storage_info
structure_file
subtract
supervisor
suppress
sync
system
table
tablespace
target
targettimespan
targettimespanoptions
task
tb
template
text
thread
to
total_size
transactions
transformation
transparent
trigger
trigger_func
trigger_spool
two_pass_calc
type
uda
unicode
unicode_mode
unlimited
unload
unlock
update
updated
updates
use
user
username_as_password
using
validate
values
variable
vector
verification
version
view_file
views
volume
wait_for_resources
warn
when
with
wizard
worksheet
write
xml_file
zero_value
zeroamountoptions
zerobasisoptions
zlib

MaxL BNF

MaxL BNF diagrams are an optional alternative to railroad diagrams, for reading MaxL syntax.

Key

{}    Alternatives (at least one required)
[]     Options (none required)
!!     Default option if none indicated
|       Separates options (OR)
[,]     Comma-separated list (of previous item) allowed
[ ... ] Whitespace-separated list (of previous item) allowed
' '    Literal
::=    "is defined as." Symbol to the left is to be replaced with expression on the right
TERMINAL
%NON-TERMINAL%

alter application

alter application
{APP-Name
    {set
        {lock_timeout after INTEGER[!seconds!|minutes]
            max_lro_file_size {unlimited|SIZE-STRING}
            minimum permission %DBS-SYSTEM-ROLE%
            variable VARIABLE-NAME STRING
            cache_size SIZE-STRING
            type unicode_mode
        }
        {load|unload} database DBS-STRING
        {enable|disable} {startup|autostartup|commands|updates|connects|security}
        comment COMMENT-STRING
        clear logfile
        add variable VARIABLE-NAME [STRING]
        drop variable VARIABLE-NAME
        rename to APP-NAME
    }

DBS-SYSTEM-ROLE::=
{no_access|read|write|execute|manager}

alter application (aggregate storage)

alter application
{APP-Name
    {set
        minimum permission %DBS-SYSTEM-ROLE%
        variable VARIABLE-NAME STRING
        cache_size SIZE-STRING
        type unicode_mode
    }
    {load|unload} database DBS-STRING
    {enable|disable} {startup|autostartup|commands|updates|connects|security}
    comment COMMENT-STRING
    clear logfile
    add variable VARIABLE-NAME [STRING]
    drop variable VARIABLE-NAME
    rename to APP-NAME
}

DBS-SYSTEM-ROLE::=
{no_access|read|write|execute|manager}
**alter database enable|disable**

```
alter database DBS-NAME
    {enable|disable}
    {
        two_pass_calc
        aggregate_missing
        startup
        autostartup
        compression
        create_blocks
        committed_mode
        pre_image_access
    }
```

**alter database set**

```
alter database DBS-NAME
    set
    {
        retrieve_buffer_size SIZE-STRING
        retrieve_sort_buffer_size SIZE-STRING
        data_cache_size SIZE-STRING
        index_cache_size SIZE-STRING
        currency_database DBS-STRING
        currency_member MEMBER-NAME
        currency_conversion {division|multiplication}
        minimum permission %DBS-SYSTEM-ROLE%
        compression {rle|bitmap}
        lock_timeout
            {immediate
             never
             |after INTEGER {[!seconds!]|minutes}}
        |implicit_commit after INTEGER {blocks|rows}
        |variable VARIABLE-NAME STRING
        |default calculation {CALC-NAME-SINGLE|as calc_string CALC-STRING}
        |active alias_table ALT-NAME-SINGLE
        |performance statistics {enabled|disabled|mode to %PST-SPEC%}
        |note COMMENT-STRING
    }
```

**DBS-SYSTEM-ROLE::=**

```
(no_access|read|write|execute|manager)
```

**PST-SPEC::=**

```
{
    default
    |(medium|long) persistence {all|database|server} scope
}
```
alter database misc

alter database DBS-NAME
{
  reset [(all|data)]
  validate
  {
    data to local logfile FILE-NAME
    using (error_file FILE-NAME|default error_file)
  }
  force restructure
  load alias_table ALT-NAME-SINGLE from data_file FILE-NAME
  unload alias_table ALT-NAME-SINGLE
  add variable VARIABLE-NAME [STRING]
  drop variable VARIABLE-NAME
  delete lro
  {
    all
    by USER-NAME
    before DATE
    by USER-NAME before DATE
  }
  unlock all objects
  begin archive to file FILE-NAME
  end archive
  [force] archive to file FILE-NAME
  [force] restore from file FILE-NAME
  replay transactions
  {
    after LOG-TIME
    using sequence_id_range ID-RANGE
  }
  rename to DBS-STRING
  comment COMMENT-STRING
}

alter database disk volumes

alter database DBS-NAME
{
  {add|drop} disk volume VOLUME-NAME
  set disk volume VOLUME-NAME
  {
    file_type {data|index|index_data}
    file_size SIZE-STRING
    partition_size {SIZE-STRING|unlimited}
  }
}

alter database (aggregate storage)

alter database DBS-NAME
{

\{(enable|disable)\\n\{\n    startup\n    autostartup\n    query_tracking\n    replication_assume_identical_outline\n\}\n\{set\n\{\n    retrieve_buffer_size SIZE-STRING\n    retrieve_sort_buffer_size SIZE-STRING\n    minimum permission %DBS-SYSTEM-ROLE%\n    variable VARIABLE-NAME STRING\n    active alias_table ALT-NAME-SINGLE\n\}\nreset [{all|data}]\ncompact outline\nadd variable VARIABLE-NAME [STRING]\ndrop variable VARIABLE-NAME\n%LOAD-BUFFER-INIT%\ndestroy load_buffer with buffer_id BUFFER-ID[,....]\nunlock all objects\nrename to DBS-STRING\ncomment COMMENT-STRING\nmerge {all|incremental} data\nbegi archive to file FILE-NAME\nenend archive\\n\}

DBS-SYSTEM-ROLE::=\n\{no_access|read|write|execute|manager\n
LOAD-BUFFER-INIT::=\ninitialize load_buffer with buffer_id BUFFER-ID[,....]\n[resource_usage RNUM][property PROPS][wait_for_resources]

\textbf{alter drillthrough}

alter drillthrough\nURL-NAME from xml_file FILE-NAME\non '{"MEMBER-EXPRESSION"}'[,....]\n[allow_merge]

\textbf{alter filter}

alter filter FILTER-NAME\nadd \{no_access|read|write|meta_read\} on MEMBER-EXPRESSION [,....]

\textbf{alter object}

alter object OBJ-NAME of type %OBJ-TYPE%\n\{rename to OBJ-NAME-SINGLE|unlock|[force]copy to OBJ-NAME\}
OBJ-TYPE::=
  outline
  |calc_script
  |report_file
  |rules_file
  |text
  |partition_file
  |lro
  |selection
  |wizard
  |eqd
  |outline_paging_file
  |worksheet
  |alias_table

alter partition

alter {transparent|replicated} partition DBS-NAME
  {to|from} DBS-NAME [at HOST-NAME]
set{
  connect as USER-NAME identified by PASSWORD
  |hostname as HOST-NAME instead of HOST-NAME direction {single|all}
  |application as APP-NAME instead of APP-NAME direction {single|all}
  |database as DSB-STRING instead of DBS-STRING
}

alter session

alter session set dml_output
{
  |
  !default!
  |alias {on|off}
  |metadata_only {on|off}
  |cell_status {on|off}
  |numerical_display {!default!|fixed_decimal|scientific_notation}
  |precision PRECISION-DIGITS
  |formatted_value {on|off}
  |get_missing_cells {on|off}
  |get_meaningless_cells {on|off}
  [,....]
}

alter system

alter system
{
  load application {all|APP-NAME}
  |unload application {all|APP-NAME} [no_force]
  set
  {
    session_idle_limit {INTEGER![seconds!|minutes]|none}
    |session_idle_poll {INTEGER![seconds!|minutes]|none}
|invalid_login_limit {INTEGER|none}| |
inactive_user_days {INTEGER[days]|none} |
password_reset_days {INTEGER[days]|none} |
variable VARIABLE-NAME STRING |
server_port begin at INTEGER end at INTEGER |
|
delete export_directory EXPORT-DIR |
add variable VARIABLE-NAME[STRING] |
drop variable VARIABLE-NAME |
logout session %SESSION-SPEC% [force] |
shutdown |
kill request %SESSION-SPEC% |
(enable|disable) unicode |
reconcile [force] |
|
SESSION SPEC::= |
all |
SESSION-ID |
by user USER-NAME |
[ |
on application APP-NAME |
on database DBS-NAME |
] |
on application APP-NAME |
on database DBS-NAME |

alter system (aggregate storage)

alter system |
{ |
load application {all|APP-NAME} |
unload application {all|APP-NAME} [no_force] |
set |
{ |
session_idle_limit {INTEGER[!seconds!|minutes]|none} |
session_idle_poll {INTEGER[!seconds!|minutes]|none} |
invalid_login_limit {INTEGER|none} |
inactive_user_days {INTEGER[days]|none} |
password_reset_days {INTEGER[days]|none} |
variable VARIABLE-NAME STRING |
server_port begin at INTEGER end at INTEGER |
} |
add variable VARIABLE-NAME[STRING] |
drop variable VARIABLE-NAME |
logout session %SESSION-SPEC% [force] |
shutdown |
kill request %SESSION-SPEC% |
reconcile [force] |
}
SESSION SPEC::=
   all
   |SESSION-ID
   |by user USER-NAME
   [ application APP-NAME
     [ database DBS-NAME
     ]
   ] application APP-NAME
   [ database DBS-NAME
   ]

alter tablespace (aggregate storage)

alter tablespace TABLSP-NAME
{ add file_location FILE-NAME
  [ set max_file_size SIZE-STRING
    |set max_disk_size SIZE-STRING
    [,....]
  ]
  alter file_location FILE-NAME
  [ set max_file_size SIZE-STRING
    |set max_disk_size SIZE-STRING
    [,....]
  ]
  drop file_location FILE-NAME
}

alter trigger

alter trigger
{ TRIGGER-NAME {enable|disable}
  [ on database DBS-NAME disable
  ]
}

create application

create [or replace] application APP-NAME
[ type {!nonunicode_mode!|unicode_mode}] [as APP-NAME]
[ comment COMMENT-STRING]
create application (aggregate storage)

create [or replace] application APP-NAME
[type {!nonunicode_mode!|unicode_mode}]
[using aggregate_storage]
[as APP-NAME]
[comment COMMENT-STRING]

create calculation

create [or replace] calculation CALC-NAME {CALC-STRING|as CALC-NAME}

create database

create [or replace] [currency] database DBS-NAME
[using non_unique_members]
[as DBS-NAME]
[comment COMMENT-STRING]

create database (aggregate storage)

create [or replace] database DBS-NAME
[using non_unique_members]
[comment COMMENT-STRING]

create drillthrough

create drillthrough URL-NAME from xml_file FILE-NAME
on '{MEMBER-EXPRESSION [,...]}'
[level0 only]

create filter

create [or replace] filter FILTER-NAME
{
    as FILTER-NAME
    |
    |
    no_access
    |read
    |write
    |meta_read
} on MEMBER-EXPRESSION
[,...]
[definition_only]
create location alias

create [or replace] location alias
{  
  LOC-ALIAS-SINGLE from DBS-NAME
  |LOCATION-ALIAS-NAME
}
to DBS-NAME at HOST-NAME as USER-NAME identified by PASSWORD

create replicated partition

create [or replace] replicated partition DBS-NAME
%AREA-SPEC%
{to|from}
DBS-NAME [at HOST-NAME][as USER-NAME identified by PASSWORD]
[using USER-NAME identified by PASSWORD for creation]
[%AREA-SPEC%]
[  
mapped
  {globally|AREA-ALIAS}
  '({'MEMBER-NAME [,..]'}')
  to '({'MEMBER-NAME [,..]'}')
  [,..]
]
[outline {!direct!|reverse}]
[comment COMMENT-STRING]
[remote comment COMMENT-STRING]
[update {allow|disallow}]
[validate only]

AREA-SPEC::=
  area MEMBER-EXPRESSION [AREA-ALIAS] [ ...]

create transparent partition

create [or replace] transparent partition DBS-NAME
%AREA-SPEC%
{to|from}
DBS-NAME [at HOST-NAME][as USER-NAME identified by PASSWORD]
[using USER-NAME identified by PASSWORD for creation]
[%AREA-SPEC%]
[  
mapped
  {globally|AREA-ALIAS}
  '({'MEMBER-NAME [,..]'}')
  to '({'MEMBER-NAME [,..]'}')
  [,..]
]
[outline {!direct!|reverse}]
[comment COMMENT-STRING]
[remote comment COMMENT-STRING]
[validate only]
AREA-SPEC::=
  area MEMBER-EXPRESSION [AREA-ALIAS] [ ...]

create after-update trigger

create [or replace] after update trigger TRIGGER-NAME
  where CUBE-AREA [when CONDITION then ACTION][ ...] end

create on-update trigger

create [or replace] ![on update!] trigger TRIGGER-NAME
  [log_value ![OFF!|ON]]
  where CUBE-AREA
  [when CONDITION then ACTION][ ...]
  [else ACTION]
  end

display application

display application ![all!]APP-NAME [message_level]

display calculation

display calculation
  ![all!]
  CALC-NAME
  on application APP-NAME
  on database DBS-NAME
}

display database

display database
  ![all!]
  DBS-NAME
  on application APP-NAME
}
  [request_history]

display disk volume

display disk volume
  ![all!]UNIQUE-VOL-NAME on database DBS-NAME

display drillthrough

display drillthrough
display filter

display filter [!all!]|FILTER-NAME|on database DBS-NAME

display filter row

display filter row [!all!]|FILTER-NAME|on database DBS-NAME

display group

display group [!all!]|GROUP-NAME|

display location alias

display location alias [!all!]|LOCATION-ALIAS-NAME|on application APP-NAME|on database DBS-NAME

display lock

display lock [!all!]|on system|on application APP-NAME|on database DBS-NAME

display object

display [locked] object

[ [!all!]|of type %OBJ-TYPE%]

[!on system!]|on application APP-NAME|on database DBS-NAME]

|OBJ-NAME of type %OBJ-TYPE%]

OBJ-TYPE::=
outline
calc_script
report_file
rules_file
text
partition_file
lro
selection
wizard
eqd
outline_paging_file
worksheet
alias_table

OBJ-NAME::=
OBJ-TYPE

display partition

display partition ![all!|on database DBS-NAME][advanced]

display privilege

display privilege
{
    user ![all!|USER-NAME]
    |group ![all!|GROUP-NAME]
}

display session

display session
[
    ![all!|SESSION-ID
    by user USER-NAME [on application APP-NAME|on database DBS-NAME]
    on application APP-NAME
    on database DBS-NAME
]

display system

display system
[
    version
    |ports {in use|overview}
    export_directory
    license_info
    security mode
    configuration
    {
        |agent
        |network
        |errors
        |on database DBS-NAME
    }
    |message_level
]

display trigger

display trigger
[
    ![all!|on system
    on application APP-NAME
    on database DBS-NAME
]
display trigger spool

display trigger_spool
[
  !all!
  on application APP-NAME
  on database DBS-NAME
  SPOOL-NAME
]

display user

display user
[
  in group ![all!|GROUP-NAME]
  [USER-NAME
]

display variable

display variable
[
  !all!
  VARIABLE-NAME
  on application APP-NAME
  on database DBS-NAME
  on system
]

drop application

drop application APP-NAME [cascade] [force]

drop calculation

drop calculation CALC-NAME

drop database

drop database DBS-NAME [force]

drop drillthrough

drop drillthrough URL-NAME
drop filter

drop filter FILTER-NAME

drop location alias

drop location alias LOCATION-_ALIAS-NAME

drop lock

drop lock
[
  !all!
  |
    !on system!
    |on application APP-NAME
    |on database DBS-NAME
  ]
  [!all! | held by USER-NAME]
]

drop object

drop object OBJ-NAME of type %OBJ-TYPE% [force]

OBJ-TYPE::=
  outline
  |calc_script
  |report_file
  |rules_file
  |text
  |partition_file
  |lro
  |selection
  |wizard
  |eqd
  |outline_paging_file
  |worksheet
  |alias_table

drop partition

drop
  {transparent|replicated}
  partition DBS-NAME {from|to} DBS-NAME
  [at HOST-NAME][force]

drop trigger

drop trigger TRIGGER-NAME
drop trigger spool

drop trigger_spool {SPOOL-NAME|all on database DBS-NAME}

execute aggregate build

execute aggregate build on database DBS-NAME
using
{
  views VIEW-ID VIEW-SIZE [,...] with outline_id OUTLINE-ID
  |view_file VIEW-FILE-NAME
}

execute aggregate process

execute aggregate process on database DBS-NAME
[stopping when total_size exceeds STOPPING-VAL]
[based on query_data]
[{enable|!disable!} alternate_rollups]

execute aggregate selection

execute aggregate selection on database DBS-NAME
[
  using views VIEW-ID[,....]
  with outline_id OUTLINE-ID
  [[[!suppress!]|force] display]
]
[selecting INTEGER views]
[stopping when total_size exceeds STOPPING-VAL]
[based on query_data]
[{dump|force_dump} to view_file VIEW-FILE-NAME]
[{enable|!disable!} alternate_rollups]

execute allocation (aggregate storage)

execute allocation process on database DBS-NAME with
{
  pov MDX-SET
  amount ALLOC-NUMERIC
  }
{amountcontext MDX-TUPLE}
{amounttimespan MDX-SET }
)
target MDX-TUPLE
{
  [targettimespan MDX-SET]
  [targettimespanoptions {!divideamout!|repeatamount}]
  [offset MDX-TUPLE]
  [debitmember MDX-MBR]
  [creditmember MDX-MBR]
} range MDX-SET
{
  [excludedrange MDX-SET]
  [basis MDX-TUPLE]
  [basistimespan MDX-SET]
  [basistimespanoptions {splitbasis|combinebasis}]
  
  share
  |spread [{skip_missing|skip_zero|skip_negative},...]
}
[zeroamountoptions {skip_to_next_amount|abort}]
[zerobasisoptions
  
  skip_to_next_amount
  |abort
]}
[round
  {INTEGER|MDX-NUMERIC}
  
  discard errors
  |errors_to_lowest
  |errors_to_highest
  |errors_to_location MDX-TUPLE
}
webdriver
{{!override!|add|subtract} values}
}

execute calculation
execute calculation
{
  CALC-NAME
  |CALC-NAME on database DBS-STRING
  |{CALC-STRING|default} on DBS-NAME
}

execute calculation (aggregate storage)
execute calculation on database DBS-NAME with
local script_file FILE-NAME pov MDX-SET sourceregion MDX-SET
{
  [target MDX-TUPLE]
[debitmember MDX-MBR]
[creditmember MDX-MBR]
[offset MDX-TUPLE]
)

[override add subtract] values

export data
export database DBS-NAME
{
[all level0 input]
data [anonymous] [in columns] to [server] data_file FILE-NAME[...]
using [local server] report_file FILE-NAME to data_file FILE-NAME

export data aggregate storage
export database DBS-NAME
{
[level0 input]
data [anonymous] to [server] data_file FILE-NAME[...]
using [local server] report_file FILE-NAME to data_file FILE-NAME

export lro
export database DBS-NAME lro to
[server local] directory
{DBS-EXPORT-DIR FULL-EXPORT-DIR}

export outline
export outline {DBS-NAME FILE-NAME}
{
all dimensions
|list dimensions '{DIM-NAME}'[...]
}
[tree with alias_table ALT-NAME-SINGLE]
to xml_file FILE-NAME

export query_tracking
export query_tracking DBS-NAME to
[server] file FILE-NAME

grant

grant
{
create_application create_user no_access administrator
[on system]
import data

import database DBS-NAME
[using max_threads INTEGER]
data
{
from
[![local|server]
[!text!]
data_file IMP-FILE
[using ![local|server] rules_file IMP-FILE]
from data_string STRING
connect as SQL-USR identified by SQL-PASS
using ![local|server] rules_file IMP-FILE
}
on error {write|append}to FILE-NAME|abort

import data (aggregate storage)

import database DBS-NAME data
{
from
[![local|server]
[!text!]
data_file IMP-FILE
[using ![local|server] rules_file IMP-FILE]
from data_string STRING
connect as SQL-USR identified by SQL-PASS
using
{
[![local|server] rules_file IMP-FILE
multiple rules_file RULE-FILE-NAME[,...]
to load_buffer_block starting with buffer id BUFFER-ID
on error {write to FILE-NAME|abort}
}
from load_buffer with buffer_id BUFFER-ID[,...]
[[!override|add|subtract] values]
[create slice]
override {all|incremental} data
on error {(write|append)to FILE-NAME|abort}

import dimensions

import database DBS-NAME dimensions
{
from
[[!local!|server]
[!text!] data_file IMP-FILE
using[[!local!|server] rules_file IMP-FILE
[[!enforce!|suppress] verification]
connect as SQL-USR identified by SQL-PASS
using[[!local!|server] rules_file IMP-FILE
},...
[
!preserve all data!
!preserve {level0|input} data
]
on error {write|append) to FILE-NAME

import lro

import database DBS-NAME
lro from [[!local!|server] directory IMPORT-DIR

import query_tracking

import query_tracking DBS-NAME from
[[!server!] file FILE-NAME

query application

query application APP-NAME get cache_size

query application (aggregate storage)

query application APP-NAME
{
get cache_size
list aggregate_storage storage_info
}

query archive file

query archive_file FILE-NAME {get overview|list disk volume}
query database

query database DBS-NAME
{
get
{
active_alias_table
(attribute_info MEMBER-NAME
attribute_spec
currency_rate
dbstats (dimension|data_block)
default calculation
member_info MEMBER-NAME
member_calculation MEMBER-NAME
estimated size
performance statistics
{
kernel_io
kernel_cache
end_transaction
database_synch
database_asynch
dynamic_calc
}
table
}
list
{
alias_table
alias_names in alias_table ALT-NAME-SINGLE
lro
[
!all!
|by USER-NAME
|before DATE
|by USER-NAME before DATE
]
|(all|data|index) file information
transactions
[
after LOG-TIME
[[force] write to file PATHNAME_FILENAME]
]
}
}

query database (aggregate storage)

query database DBS-NAME
{
get
{
active_alias_table
attribute_info MEMBER-NAME
|attribute_spec
|cube_size_info
|dbstats {dimension|data_block}
|member_info MEMBER-NAME
|opg_state of %OPG-SECTION% for dimension DIM-NAME
|
|list
|
|aggregate_storage runtime_info
|aggregate_storage compression_info
|aggregate_storage group_id_info
|aggregate_storage slice_info
|aggregate_storage uncommitted_transaction_info
|alias_table
|alias_names in alias_table ALT-NAME-SINGLE
|existing_views {based on query_data
|{!all!|data|index} file information
|load_buffers
|aso_level_info
|
|{dump|force_dump}

existing_views to view_file VIEW-FILE-NAME
[based on query_data]
|

refresh outline

refresh outline on {transparent|replicated}
partition DBS-NAME (to|from) DBS-NAME
[at HOST-NAME]
|
|purge outline change_file
|apply all
|apply nothing
|%OTL-CHANGE-SPEC%
|

OTL-CHANGE-SPEC::=  
apply on dimension
{add|delete|rename|update|move}[...,]
apply on member
{add|delete|rename|move}[...,]
apply on member_property 
|account_type
|alias
|calc_formula
|consolidation
|currency_conversion
|currency_category
|data_storage
|uda
)[...,]
refresh replicated partition

refresh replicated partition DBS-NAME
{to|from} DBS-NAME
[at HOST-NAME]
[[!all!|updated]data]

MaxL Statements (Aggregate Storage)

Click here for non-aggregate storage list

Some MaxL grammar is applicable only to aggregate storage mode, and some standard grammar is not applicable to aggregate storage mode. The following statements support aggregate storage application and database operations.

- alter application
- alter database
- alter filter
- alter object
- alter partition
- alter system
- alter tablespace
- alter trigger
- create application
- create database
- create filter
- create outline
- create partition
- create after-update trigger
- display application
- display calculation
- display database
- display filter
- display filter row
- display group
- display lock
- display object
- display partition
- display privilege
- display session
- display system
• display tablespace
• display trigger
• display user
• display variable
• drop application
• drop calculation
• drop database
• drop filter
• drop lock
• drop object
• drop partition
• drop trigger
• execute aggregate process
• execute aggregate build
• execute aggregate selection
• export data
• grant
• import data
• import dimensions
• login
• query application
• query database
• refresh outline
• refresh replicated partition

The MaxL grammar is case-insensitive. Semicolon statement-terminators are required when using the MaxL Shell. Key words of the MaxL grammar are represented in this document in lower-case. Terminals, represented in upper-case, are to be replaced by the appropriate names, numbers, privileges, or strings. For more information about components of MaxL statements, see MaxL Definitions.

Note:

Login is part of the separate command shell grammar, not the MaxL language itself.

Alter Application (Aggregate Storage)

Click here for non-aggregate storage version

Change application-wide settings.
Permission required: Application Manager.

Syntax

```
alter application APP-NAME
```

### Keywords

- **APP-NAME**
- **Database-Level System Roles**
- **VARIABLE-NAME**
- **SIZE-STRING**
- **DBS-STRING**
- **COMMENT-STRING**
- **VARIABLE-NAME**

You can change the following application-wide settings using `alter application`.

### Keywords

#### set minimum permission
Grant all users a minimum level of permission to all databases in the application. Users with higher permissions than this minimum are not affected.

#### set variable
Assign a string value to an existing substitution-variable name. If the variable does not exist, first create it using `add variable`. Substitution variables may be referenced by calculations in the application.

#### set cache_size
Set the maximum size to which the aggregate storage cache may grow. The aggregate storage cache grows dynamically until it reaches this limit. This setting
takes effect after you restart the application. To check the currently set limit, use the following MaxL statement:

```
query application APP-NAME get cache_size;
```

**set type unicode_mode**
Migrate an application to Unicode mode. Migration to Unicode mode cannot be reversed.

**load database**
Start (by loading into memory) an idle database. The statement will fail if you do not have at least read privilege for the database.

**unload database**
Stop (by unloading from memory) an active database. The statement will fail if you do not have at least read privilege for the database.

**enable startup**
Permit all users to load (start) the application. This only applies to users who have at least read privilege for the application. Startup is enabled by default.

**disable startup**
Prevent all users from loading (starting) the application. Startup is enabled by default.

**enable autostartup**
Start the application automatically when Essbase Server starts. By default, autostartup is disabled.

**disable autostartup**
Do not start the application automatically when Essbase Server starts. By default, autostartup is disabled.

**enable commands**
Allow all users with sufficient permissions to make requests to databases in the application. Use to reverse the effect of **disable commands**. The disable commands setting remains in effect only for the duration of your session. By default, commands are enabled.

**disable commands**
Prevent all requests to databases in the application, including non-data-specific requests, such as viewing database information or changing database settings. All users are affected, including other administrators. Administrators are affected by this setting as a safety mechanism to prevent accidental updates to databases during maintenance operations. This setting remains in effect only for the duration of your session. The setting takes effect immediately, and affects users who are currently logged in, as well as users who log in later during your session.

⚠️ **Caution:**
If performing maintenance operations that require disabling commands, you must make those maintenance operations within the same session and the same script as the one in which commands were disabled.
By default, commands are enabled.

**enable updates**
Allow all users with sufficient permissions to make requests to databases in the application. Use to reverse the effect of disable updates. Disabling updates remains in effect only for the duration of your session. By default, updates are enabled.

**disable updates**
Prevent all users from making requests to databases in the application. Use before performing update and maintenance operations. The disable updates setting remains in effect only for the duration of your session.

### Caution:
If performing maintenance operations that require updates to be disabled, you must make those maintenance operations within the same session and the same script as the one in which updates were disabled. By default, updates are enabled.

**enable connects**
Allow all users with sufficient permissions to make connections to databases in the application. Use to reverse the effect of disable connects. By default, connections are enabled.

**disable connects**
Prevent any user with a permission lower than Application Manager from making connections to the databases that require the databases to be started. Database connections remain disabled for all databases in the application, until the application setting is re-enabled by the administrator. By default, connections are enabled.

**enable security**
When security is disabled, Essbase ignores all security settings in the application and treats all users as Application Managers. By default, security is enabled.

**disable security**
When security is disabled, Essbase ignores all security settings in the application and treats all users as Application Managers. By default, security is enabled.

**comment**
Enter an application description (optional). The description can contain up to 80 characters.

**clear logfile**
Delete the application log located in the application directory. A new log is created for entries recording subsequent application activity.

**add variable**
Create an application-level substitution variable by name, and optionally assign a string value for the variable to represent. You can assign or change the value later using **set variable**. A substitution variable acts as a global placeholder for information that changes regularly. Substitution variables may be referenced by calculations and report scripts.
If substitution variables with the same name exist at server, application, and database levels, the order of precedence for the variables is as follows: a database level substitution variable supersedes an application level variable, which supersedes a server level variable.

**drop variable**
Remove a substitution variable and its corresponding value from the application.

**rename to**
Rename the application. When you rename an application, the application and the application directory (`ARBORPATH\app\appname`) are renamed.

**Example**

```sql
alter application ASOsamp set cache_size 64MB;
```

Sets the maximum size of the aggregate storage cache to 64 MB.

```sql
alter application ASOsamp disable commands;
```

Prevents all users from making requests to the application scope. Use this statement before performing application-wide update and maintenance operations.

```sql
alter application ASOsamp comment 'Aggregate storage application';
```

Attaches a descriptive comment to the ASOsamp application.

---

**Alter Database (Aggregate Storage)**

[Click here for non-aggregate storage version](#)

Change database-wide settings.

Permission required: create_application.

**Syntax**
You can change the following database-wide settings using `alter database`.
Keywords

**enable startup**
Enable users to start the database directly or as a result of requests requiring the database to be started. Startup is enabled by default.

**disable startup**
Prevent all users from starting the database directly or as a result of requests that would start the database. Startup is enabled by default.

**enable autostartup**
Automatically start the database when the application to which it belongs starts. Autostartup is enabled by default. This setting is applicable only when startup is enabled.

**disable autostartup**
Prevent automatic starting of the database when the application to which it belongs starts. Autostartup is enabled by default.

**enable query_tracking**
Begin collecting query data for this database, to be used for query-based view optimization.
To utilize the results of query tracking, use the optional **based on query_data** grammar in any of the following statements:

- `query database <dbs-name> list existing_views`
- `execute aggregate process`
- `execute aggregate selection`

Query tracking is disabled by default.

**disable query_tracking**
Stop collecting query data for query-based view optimization. Query tracking is disabled by default.

**set retrieve_buffer_size**
Change the database retrieval buffer size. This buffer holds extracted row data cells before they are evaluated by the RESTRICT or TOP/BOTTOM Report Writer commands. The default size is 10 KB. The minimum size is 2 KB. Increasing the size may improve retrieval performance.

**set retrieve_sort_buffer_size**
Change the database retrieval sort buffer size. This buffer holds data until it is sorted. The default size is 10 KB. The minimum size is 2 KB. Increasing the size may improve retrieval performance.

**set minimum permission**
Set a level of permission that all users or groups can have to the database. Users or groups with higher granted permissions than the minimum permission are not affected.

**set variable**
Change the value of an existing substitution variable on the database. The value must not exceed 256 bytes. It may contain any character except a leading ampersand (&).
set active alias_table
Set an alias table as the primary table for reporting and any additional alias requests. Only one alias table can be used at a time. This setting is user-specific; it only sets the active alias table for the user issuing the statement.

reset
Clear all data and linked-reporting objects from the database, but preserve the outline.

Note:
If kernel queries are running when a clear data operation starts, the clear data operation waits for the kernel queries to complete and then the clear data operation proceeds. This information also applies to the reset all and reset data grammar.

reset all
Clear all data, Linked Reporting Objects, and the outline.

reset data
Same as using reset.

clear aggregates
Delete all aggregate views.

compact outline
Compact the outline file to decrease the outline file size. Compaction helps keep the outline file at an optimal size. After the outline file is compacted, the file continues to grow as before, when members are added or deleted.

Note:
Compacting the outline does not cause Essbase to clear the data. When a member is deleted from the outline, the corresponding record of that member in the outline file is marked as deleted but the record remains in the outline file. Compacting the outline file does not remove the records of deleted members.

add variable
Create a database-level substitution variable by name, and optionally assign a string value for the variable to represent. You can assign or change the value later using set variable. A substitution variable acts as a global placeholder for information that changes regularly. Substitution variables may be referenced by calculations and report scripts.
If substitution variables with the same name exist at server, application, and database levels, the order of precedence for the variables is as follows: a database level substitution variable supersedes an application level variable, which supersedes a server level variable.

drop variable
Remove a substitution variable and its corresponding value from the database.
initialize load_buffer
Create a temporary buffer in memory for loading data.
Data load buffers are used in aggregate storage databases for allocations, custom
calculations, and lock and send operations. Multiple data load buffers can exist on a
single aggregate storage database.
You can control the share of aggregate storage cache resources the load buffer is
allowed to use and how long to wait for resources to become available before aborting
load buffer operations. You can also set properties that determine how missing and
zero values, duplicate values, and multiple values for the same cell in the data source
are processed.
  • resource_usage
  • property
destroy load_buffer
Destroy the temporary data-load memory buffer.
unlock all objects
Unlock all objects on the database that are in use by a user or process.
rename to
Rename the database. When you rename a database, the database directory is also
renamed.
comment
Create a description of the database. The maximum number of characters is 80. This
description is available to database administrators. To annotate the database for
Smart View or other grid client users, use set note.
merge all|incremental data [remove_zero_cells]
Merge incremental data slices. Use these keywords:
  • all—Merge all incremental data slices into the main database slice.
  • incremental—Merge all incremental data slices into a single data slice. The main
database slice is not changed.
  • (Optional) remove_zero_cells—When merging incremental data slices, remove
cells that have a value of zero (logically clearing data from a region results in cell
with a value of zero).

clear data in region ...
Clear the data in the specified region.
There are two methods for 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. Therefore, you might typically use this method only when you need to remove large slices of data.

Use the MaxL statement with the physical keyword:

```sql
alter database appname.dbname clear data in region 'MDX set expression' physical;
```

- 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. Because compensating cells are created, this option increases the size of the database.

Use the MaxL statement without a keyword:

```sql
alter database appname.dbname clear data in region '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 in alternate hierarchies. (If the region contains upper-level members from alternate hierarchies, you may experience a decrease in performance.) Members cannot be dynamic members (members with implicit or explicit MDX formulas), nor can they be from an attribute dimension.

To remove cells with a value of zero, use the `alter database` MaxL statement with the `merge` grammar and the `remove_zero_cells` keyword.

**enable replication_assume_identical_outline**
Optimize the replication of an aggregate storage database when the aggregate storage database is the target and a block storage database is the source and the two outlines are identical.

Replication optimization affects only the target aggregate storage application; the source block storage application is not affected. This functionality does not apply to block storage replication.

**disable replication_assume_identical_outline**
Do not optimize the replication of an aggregate storage database when the aggregate storage database is the target and a block storage database is the source and the two outlines are identical.

**begin archive to file**
Prepare the database for backup by an archiving program, and prevent writing to the files during backup.

Begin archive achieves the following outcomes:

- Switches the database to read-only mode. The read-only state persists, even after the application is restarted, until it is changed back to read-write using `end archive`.

- Creates a file containing a list of files that need to be backed up. Unless a different path is specified, the file is stored in the database directory.

Begin archive and end archive do not perform the backup; they simply protect the database during the backup process.
Note:
Using the **begin archive to file** and **end archive** grammar is the only supported way to backup and recover a database using MaxL.

**end archive**
Return the database to read-write mode after backing up the database files.

Note:
Using the **begin archive to file** and **end archive** grammar is the only supported way to backup and recover a database using MaxL.

**Example**

```sql
alter database ASOsamp.Sample clear aggregates;
```

Deletes all aggregate views in the ASOsamp.Sample database.

```sql
alter database ASOsamp.Sample initialize load_buffer with buffer_id 1;
```

See [Loading Data Using Buffers](#).

```sql
alter database ASOsamp.Sample initialize load_buffer with buffer_id 1
resource_usage .5 property ignore_missing_values, ignore_zero_values;
```

Creates a data-load buffer in memory for the ASOsamp.Sample database. The buffer can use only 50% of available resources. Missing values and zeros in the data source are ignored.

```sql
alter database ASOsamp.Sample disable query_tracking;
```

Turns off the harvesting of query data for the ASOsamp.Sample database.

```sql
alter database ASOsamp.Sample merge all data;
```
Merges all incremental data slices into the main slice in the ASOsamp.Sample database.

```
alter database ASOsamp.Sample merge incremental data;
```

Merges all incremental data slices into a single data slice within the ASOsamp.Sample database.

```
alter database ASOsamp.Sample merge all data remove_zero_cells;
```

Merges all incremental data slices into the main slice in the ASOsamp.Sample database, and removes cells with a value of zero.

```
alter database ASOsamp.Sample clear data in region '{Jan, Budget}';
```

Clears all Budget data for the month of Jan, using the logical method, from the ASOsamp.Sample database.

```
alter database ASOsamp.Sample clear data in region '{Jan, Budget}' physical;
```

Clears all Budget data for the month of Jan, using the physical method, from the ASOsamp.Sample database.

```
alter database ASOsamp.Sample clear data in region 'CrossJoin({Jan}, {Forecast1, Forecast2})';
```

Clears all January data for the Forecast1 and Forecast2 scenarios from the ASOsamp.Sample database.

---

**Alter System (Aggregate Storage)**

[Click here for non-aggregate storage version](#)

Change the state of the Essbase Server. Start and stop applications, manipulate system-wide variables, manage password and login activity, disconnect users, kill processes, and shut down the server.

Permission required: Administrator.

**Syntax**
You can change the following system-wide settings using `alter system`.

**Keywords**

- `load application`
  Start an application, or start all applications on the Essbase Server.

- `unload application`
  Stop an application, or stop all applications on the Essbase Server. Unloading an application cancels all active requests and database connections, and stops the application. If Essbase encounters a problem when trying to cancel active requests and database connections, and stopping the application, an error is logged in the application log.

  If you do not want to stop an application if it has active requests and database connections, use the `no_force` grammar. When using `no_force`: 

  ```plaintext
  set session_idle_limit APP-NAME none
  ```
• If the application has active requests and database connections, the application is not stopped; it continues running.

• If the application does not have active requests and database connections, the application is stopped, as if you used `unload application` without specifying `no_force`.

---

**Note:**

Unloading an application cancels all active requests and database connections, and stops the application, unless you explicitly specify otherwise using the `no_force` option. The `no_force` option causes Essbase to return an error if active requests are running on the application. An internal logic error [200] is logged when a database is unable to shut down gracefully when unloading an application or shutting down the system while a process is running on the database.

---

**set session_idle_limit**

Set the interval of time permitted for a session to be inactive before Essbase Server logs off the user. The minimum limit that you can set is five minutes (or 300 seconds). When the session idle limit is set to `none`, all users can stay logged on until the Essbase Server is shut down.

The default user idle logout time is 60 minutes. When a user initiates a calculation in the background, after 60 minutes the user is considered idle and is logged out, but the calculation continues in the background.

Because the user may mistakenly assume that the calculation stopped because he or she was logged out, you can do one of the following to correct the user experience:

• Run the calculation in the foreground

• Increase the session idle limit in to a time that exceeds the duration of the calculation, or to `none`

**set session_idle_poll**

Set the time interval for inactivity checking and security-backup refreshing. The time interval specified in the session idle poll gives Essbase instructions:

• Tells it how often to check whether user sessions have passed the allowed inactivity interval indicated by `session_idle_limit` in the `alter system` statement.

• Tells it how often to refresh the security backup file. If `session_idle_poll` is set to zero, the security backup file is still refreshed every five minutes.

**set invalid_login_limit**

Set the number of unsuccessful login attempts allowed by any user before the user account becomes disabled. When you change this setting, the counter resets to 0. When the invalid login limit is set to `none`, there is no limit. By default, there is no limit.

**set inactive_user_days**

Set the number of days a user account may remain inactive before the system disables it. The counter resets when the user logs in, is edited, or is activated by an administrator. When the inactive days limit is set to `none`, user accounts remain enabled even if they are not used. By default, there is no limit.
set password_reset_days
Set the number of days users may retain passwords. After the allotted number of
days, users are prompted at login to change their passwords. The counter resets for a
user when the user changes the password, is edited, or is activated by an
administrator. When the password reset days limit is set to none, there is no built-in
limit for password retention. By default, there is no limit.

set variable
Change the value of an existing substitution variable on the system. The value must
not exceed 256 bytes. It may contain any character except a leading ampersand (\&).

set server_port
Expand a port range specified in Essbase configuration properties. Each Essbase
application uses two ports from this range. If no more ports are available, an error
message is displayed.

Note:
You can expand port ranges only so that the beginning port range is less
than SERVERPORTBEGIN and the ending port range is greater than
SERVERPORTEND.

add variable
Create a system-level substitution variable by name, and optionally assign a string
value for the variable to represent. You can assign or change the value later using set
variable. A substitution variable acts as a global placeholder for information that
changes regularly. Substitution variables may be referenced by calculations and
report scripts.
If substitution variables with the same name exist at server, application, and database
levels, the order of precedence for the variables is as follows: a database-level
substitution variable supersedes an application-level variable, which supersedes a
server-level variable.

drop variable
Remove a substitution variable and its corresponding value from the system.

logout session all
Terminate all user sessions currently running on the Essbase Server.

logout session...force
Terminate a session (or sessions) even if it is currently processing a request. The
request is allowed to proceed to a safe point, and then the transaction is rolled back.

logout session <session-id>
Terminate a session by its unique session ID number. To see the session ID number,
use display session.

logout session by user
Terminate all current sessions by a particular user, either across the entire Essbase
Server, or limited to a specific application or database.

logout session by user on application
Terminate all current sessions by a particular user across a specific application.
logout session by user on database
Terminate all current sessions by a particular user across a specific database.

logout session on application
Terminate all current user sessions across a specific application.

logout session on database
Terminate all current user sessions across a specific database.

shutdown
Shut down the Essbase Server.

kill request all
Terminate all current requests on the Essbase Server.

Note:
To terminate your own active request in MaxL Shell, press the ESC key.

kill request <session-id>
Terminate the current request indicated by the session ID. You can obtain session IDs using display session.

kill request by user
Terminate all current requests by the specified user on the Essbase Server.

kill request on application
Terminate all current requests on the specified application.

kill request on database
Terminate all current requests on the specified database.

reconcile
When Essbase is started using a security backup file (essbase_timestamp.bak) instead of essbase.sec, reconcile the security file to match the state of Essbase on an external disk. This grammar displays discrepancies in application and database information between the security file and the external disk:

- If an application folder is on the disk but not in the security file, display a message indicating the discrepancy. (Essbase checks for the presence of a appname.app file in the ARBORPATH/app/appname directory.)
  The force option does not apply in this scenario.

- If an application file is in the security file but not on the disk, display a message indicating the discrepancy.
  The force option removes the application from the security file.

- If an application database folder is on the disk but not in the security file, display a message indicating the discrepancy. (Essbase checks for the presence of a dbname.otl file in the ARBORPATH/app/appname/dbname directory.)
  The force option does not apply in this scenario.
• If an application database file is in the security file but not on the disk, display a message indicating the discrepancy.

The force option removes the database from the security file.

Notes

SESSION SPECIFICATION

A session is a single user connection to Essbase Server. The session can be identified by keywords and names indicating context, or by a unique session ID number.

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. Only one request at a time can be processed in each session.

If a session is processing a request at the time that an administrator attempts to terminate the session, the administrator must either terminate the request first, or use the force keyword available with alter system to terminate the session and the current request.

Example

alter system unload application Sample;

Stops the Sample application, if it is currently running.

alter system unload application all;

Terminates all active requests and stops all applications.

alter system unload application Sample no_force;

Essbase prepares to unload the Sample application; however, if active requests are running, the application is not stopped.

alter system shutdown;
Stops all running applications and shuts down Essbase Server.

alter system logout session by user Fiona;

Disconnects Fiona from any applications or databases to which she is connected.

To log out a user, log out the sessions owned by that user.

alter system set password_reset_days 10;

Specifies that all users will be prompted after 10 days to change their passwords. The day count for any user is reset when the user changes the password or is edited or reactivated by an administrator.

alter system unload application Sample;

Create Application (Aggregate Storage)

Click here for non-aggregate storage version

Create or re-create an application, either from scratch or as a copy of another application on the same system. See APP-NAME for information on the maximum length of and special characters that are allowed in an application name. Application names are not case-sensitive.

Syntax

```plaintext
create application APP-NAME
or replace application APP-NAME
  type nonunicode_mode
  using aggregate_storage
  as APP-NAME
  comment COMMENT-STRING
```

- APP-NAME
- COMMENT-STRING

You can create an application in the following ways using the aggregate storage version of `create application`.

Keywords

- `create application`
  Create a new application. Application names are not case-sensitive.
**create or replace application**
Create an application, or replace an existing application of the same name.
Application names are not case-sensitive.

**...type nonunicode_mode**
Create a Non Unicode-mode application. This is also the default if these keywords are omitted.

**...type unicode_mode**
Create a Unicode-mode application.

**...using aggregate_storage**
Create an application using an aggregate storage model. Only one database per application is allowed. Selecting to use aggregate storage model for an application is non-reversible.
Use the aggregate storage model if the following is true for your database:

- The database is sparse and has many dimensions, or a large hierarchical depth of members in the dimensions.
- The database is used primarily for read-only purposes; there are few or no data updates.
- There are no formulas on the outline except in the dimension tagged as Accounts.
- Calculation of the database is frequent and highly aggregational, with no dependency on calculation scripts.

**create application as**
Create an application as a copy of another application. Application names are not case-sensitive.
You cannot copy block storage applications to aggregate storage applications or vice versa. The copy will always use the same storage as the original. However, you can convert an outline from a block storage database to an aggregate storage database, using `create outline`.
Before you copy an aggregate storage application, you must merge all incremental data slices into the main database slice. Data in unmerged incremental data slices is not copied.

**comment**
Create an application description (optional). The description can contain up to 80 characters.

**Example**

```maxl
create application Sample2 using aggregate_storage comment 'aggregate storage application.';
```

Creates a new aggregate storage application called Sample2, with an associated comment.

Create Database (Aggregate Storage)

[Click here for non-aggregate storage version]
Create or re-create a database for an aggregate storage application. See DBS-NAME for information on the maximum length of and special characters that are allowed in a database name. Database names are not case-sensitive.

The syntax for creating an aggregate storage database is the same as for creating a block storage database. You must create an aggregate storage database as part of an aggregate storage application.

Permission required: Application Manager.

Syntax

\[
\begin{align*}
\text{create} &\quad \text{or replace} &\quad \text{database} \text{DBS-NAME} &\quad \text{using non_unique_members} \quad \text{comment} \text{COMMENT-STRING}
\end{align*}
\]

- DBS-NAME
- COMMENT-STRING

Use create database to create a database in the following ways:

Keywords

create database
Create a new database. Database names are not case-sensitive.

create or replace database
Create a database, or replace an existing database of the same name. Database names are not case-sensitive.

create database using non_unique_members
Create a database that supports the use of duplicate member names. Once you have created a database with a duplicate member outline, you cannot convert it back to a unique member outline.

For more information about duplicate member names, see Creating and Working With Duplicate Member Outlines in Designing and Maintaining Essbase Cubes.

comment
Create a database description (optional). The description can contain up to 80 characters.

Notes

- You cannot create an aggregate storage database as a copy of another aggregate storage database. Only one aggregate storage database is allowed per application.
- You cannot copy a block storage database to an aggregate storage database. For an example of how to create an aggregate storage application and database based on a block storage application and database, see Creating an Aggregate Storage Sample Using MaxL.
Example

create or replace database Sample.Basic comment 'This is a test.';

Creates a database called Basic within the Sample application. If a database named Basic within the Sample application already exists, it is overwritten.

Create Outline (Aggregate Storage)

Create an aggregate storage outline based on a block storage outline. The outline you are creating must be for an aggregate storage cube that is local to your current login session. The block-storage cube you are using as a source can be remote. If a remote host is specified, you can also specify a user name and password if the connection is remote.

Permission required: Database Manager.

Essbase supports the following scenarios for converting block storage outlines to aggregate storage outlines:

• Non-Unicode block storage outline to non-Unicode aggregate storage outline
• Non-Unicode block storage outline to Unicode aggregate storage outline
• Unicode block storage outline to Unicode aggregate storage outline

The following conversion scenarios are not supported:

• Unicode block storage outline to non-Unicode aggregate storage outline
• Aggregate storage outline to a block storage outline

Syntax

```
create outline on aggregate_storage database [DBS-NAME] [as outline]
```

```
create or replace outline on aggregate_storage database [DBS-NAME] [as outline]
```

```
on database [DBS-NAME] [at HOST-NAME] [as USER-NAME identified by PASSWORD]
```

• DBS-NAME
• HOST-NAME
• USER-NAME
• PASSWORD

You can create an outline in the following ways using create outline.

Keywords

create outline...

Create an aggregate-storage cube outline based on a block storage outline. If an outline of the same name already exists, it is replaced.
create or replace outline...
This statement has the same result as create outline above.

at HOST-NAME
If the block-storage cube you are using as a source is remote, specify its discovery URL. For example, "https://myEssbase-myDomain.analytics.us2.example.com/essbase/agent"

as USER-NAME identified by PASSWORD
If the block-storage cube you are using as a source is remote, specify the location. If the connection is also remote (requires a different authentication), provide the user name and password, as you would do when creating a remote partition.

Example

create or replace outline on aggregate_storage database Sample2.Basic2 as outline on database sample.basic;

Creates an aggregate storage outline based on the Sample.Basic outline. For a complete example of how to create an aggregate storage version of a block storage cube, see Creating an Aggregate Storage Sample Using MaxL.

Display Tablespace (Aggregate Storage)
View details about a tablespace.
Tablespaces are applicable only to aggregate storage databases.
Permission required: Application Manager.
This statement requires the application to be started.

Syntax

```sql
Display Tablespace (Aggregate Storage)

<table>
<thead>
<tr>
<th>Column Header</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>file_location</td>
<td>C:\Hyperion\products\Essbase\EssbaseServer\APP\</td>
</tr>
<tr>
<td>max_file_size</td>
<td>56</td>
</tr>
</tbody>
</table>
| max_disk_size   | 4294967295                                                                     ```

This example displays the following output:

Table 5-17  Display Tablespace MaxL Output Columns
Execute Allocation

Allocate one or more given source amounts to a target range of cells in an aggregate storage database. The source amount can be allocated to the target proportionately to a given basis, or the source amount can be spread evenly to the target region.

Allocations are typically used in the budgeting process to distribute revenues or costs.

Minimum permission required: Execute.

For more information about allocations and to understand the input parameters, see Performing Custom Calculations and Allocations on Aggregate Storage Databases in Designing and Maintaining Essbase Cubes.

Syntax

```
execute allocation process on database DBS-NAME with pov MDX-SET amount ALLOC-NUMERIC

amount context MDX-TUPLE amount timespan MDX-SET

target timespan MDX-SET target timespan options divide amount repeat amount

offset MDX-TUPLE range MDX-SET

debttimespan MDX-SET credittimespan MDX-MBR

basis MDX-TUPLE basis timespan MDX-SET basis timespan options

splitbasis combinebasis

<Alloc-Method>

zero amount options skip_to_next_amount abort

<zero basis options skip_to_next_amount abort>

-negative basis options skip_to_next_amount abort

absolute_value missing_value zero_value

<Rounding-Method>

override values

add subtract
```

- DBS-NAME
- MDX-SET
- ALLOC-NUMERIC
- MDX-TUPLE
- MDX-MBR

Keywords

```
pov <mdx-set>
```

Required. Provide an MDX set defining the context region in which the allocation is performed.
amount <alloc-numeric>
Required. Provide an MDX numeric value expression indicating the amount to be allocated.

amountcontext <mdx-tuple>
Optional. Provide an MDX tuple with one member from each dimension missing from pov and amount. This clause is required when amount is an arithmetic expression and pov does not specify two or more dimensions. It should not be used otherwise.

amounttimespan <mdx-set>
Optional. Provide an MDX set indicating one or more time periods to be considered for the amount. The amount value is aggregated over the specified time periods, and the aggregated amount value is allocated. Time periods must be level 0 members in a Time dimension.

target <mdx-tuple>
Required. Provide an MDX tuple defining the database region where results are written.

targettimespan <mdx-set>
Optional. Provide an MDX set indicating one or more time periods to be considered for the target. Time periods must be level 0 members in a Time dimension.

targettimespanoptions
Optional, but required if targettimespan is used.
Select a method for allocating values across the target time span:
• divideamount—Divide the amount evenly across the time periods
• repeatamount—Repeat the amount across the time periods

offset <mdx-tuple>
Optional. If offsetting entries are used, provide an MDX tuple defining the location in the database where an offsetting value is written for each source amount.

debitmember <mdx-mbr>
Optional. If double-entry accounting is used, provide an MDX member expression indicating the member to which positive result values are written.

creditmember <mdx-mbr>
Optional. If double-entry accounting is used, provide an MDX member expression indicating the member to which negative result values are written.

range <mdx-set>
Required. Provide an MDX set indicating the database region in which allocated values are calculated and written.

excludedrange <mdx-set>
Optional. Provide an MDX set specifying locations in the range where you do not want allocation values written.

basis <mdx-tuple>
Required in most cases. Provide an MDX tuple that, when combined with the range, defines the location of basis values that determine how the amount is allocated. The basis can consist of upper-level or level 0 members.
Optional if the allocation method used is \textit{spread}, and no values are skipped; required otherwise. Basis must be omitted when the allocation method \textit{spread} is used without \textit{skip} options.

\textit{basistimespan \langle\text{mdx-set}\rangle}

Optional. Provide an MDX set that indicates one or more time periods to be considered for the basis. Time periods must be level 0 members in a Time dimension.

\textit{basistimespanoptions}

Optional, but required if \textit{basistimespan} is used. Select a method for using the basis time span:

- \textit{splitbasis}—Use the basis value for each time period individually
- \textit{combinebasis}—Use the sum of the basis values across the time periods specified by \textit{basistimespan}

\textit{share}

Optional. Specify to allocate the amount(s) proportionately to the basis values. For syntax, see Allocation Method Specification in Notes.

\textit{spread}

Optional. Specify to allocate the amount(s) evenly. For syntax, see Allocation Method Specification in Notes. You can include one or more of the following skip options when using spread allocation:

- \textit{skip\_missing}—Skip missing basis values
- \textit{skip\_zero}—Skip zero basis values
- \textit{skip\_negative}—Skip negative basis values

\textit{zeroamountoptions}

Optional. If omitted, zero or \#MISSING amount values are allocated. Otherwise, specify treatment of amount values that are zero or \#MISSING:

- \textit{skip\_to\_next\_amount}—Skip to the next nonzero, non-\#MISSING amount value
- \textit{abort}—Cancel the entire allocation operation

\textit{zerobasisoptions}

Optional. For \textit{share}, this option specifies the action when the sum of all basis values is zero. For \textit{spread}, this option specifies the action when all the basis values are skipped. Select one of the following options:

- \textit{skip\_to\_next\_amount}—Skip to the next nonzero, non-\#MISSING amount value
- \textit{abort}—Cancel the entire allocation operation

\textit{round}

Optional. Specify rounding options. The following options are available:

- Round to a specified number of decimal places, using an integer or MDX numeric value expression. The value must be between 100 and -100, and is truncated if it is not a whole number.
- Perform rounding, but discard rounding errors
- Add rounding errors to the highest allocated value
- Add rounding errors to the lowest allocated value
• Provide an MDX tuple indicating a cell to which the rounding error should be added

**override|add|subtract values**
Optional. Generated allocation values can be added to (or subtracted from) existing values, instead of overwriting them. Overwriting is the default.

**Notes**
• The clauses following the with keyword can be entered in any order, each separated by white space.
• Each clause can only be entered once.
• The pov, amount, target, range, and basis clauses are mandatory; the others are optional.
• You can specify only stored, level-0 members in all of the clauses except for amount, amountcontext, basis, and the number of rounding digits; for all other arguments, do not use upper-level members, attribute members, or dynamic calc members.

**Allocation Method Specification**

```
<Alloc-Method> ::= share spread
```

**Rounding Method Specification**

```
<Rounding-Method> ::= round
```

**Example**
The following statement executes an allocation. For a more complete use case, see Performing Custom Calculations and Allocations on Aggregate Storage Databases in Designing and Maintaining Essbase Cubes.

```
execute allocation process on database glrpt.db with pov *Crossjoin({[VisionUS]},
```
Crossjoin({[5740]},
Crossjoin({[USD]},
   Descendants([Geography],[Geography].Levels(0))))

amount        "Jan + Feb"
amountcontext "{"[100], [Beginning Balance], [Actual], [CostCenter1]}"
target        "{"[Allocation], [CostCenter1]}"
offset        "{"[Allocation], [CostCenter1], [100], [YearNA]}"
debitmember   "[Debit]"
creditmember  "[Credit]"
range         "Crossjoin(Descendants([999], [Department].Levels(0)),
   Descendants([Year], [Year].Levels(0)))"
excludedrange "{"9994], [9995], [9996]}"
basis         "{"[SQFT], [Balance], [Actual], [CostCenter2]}"
share
zeroamountoptions abort
zerobasisoptions abort
negativebasisoptions zero_value
targettimespanoptions divideamount
round        "Currency.CurrentMember.CurrencyPrecision"
errors_to_location "{"[101], [Jan]}"
add values;

Execute Calculation (Aggregate Storage)

Click here for non-aggregate storage version

Execute a custom calculation script expressed in MDX, specifying the script file, source region, and point of view (POV). Optionally specify the target, offset, and debit or credit members.

Minimum permission required: Execute.

For more information about custom calculation script parameters, see Performing Custom Calculations and Allocations on Aggregate Storage Databases in Designing and Maintaining Essbase Cubes.

Syntax

execute calculation on database DBS-NAME with local script file FILE-NAME pov MDX-SET

source region MDX-SET

target MDX-TUPLE
debitmember MDX-MBR
creditmember MDX-MBR
offset MDX-TUPLE

override
add
subtract

- DBS-NAME
- FILE-NAME
- MDX-SET
- MDX-TUPLE
- MDX-MBR
You can execute custom calculations with the following options:

**Keywords**

**local script_file**
Required. Run the specified local calculation script file. Custom calculation scripts are expressed in MDX. The following is an example of a custom calculation script, script.txt.

```
(AccountA,Proj1) := 100;
([AccountB], [Proj1]) := ([AccountB], [Proj1]) * 1.1;
(AccountC,Proj1) :=
    ((AccountB,Proj1,2007) + (AccountB, Proj1)) / 2;
(AccountA,Proj2) :=
    ((AccountD,Proj1) +
     (AccountB,Proj2)) / 2;
```

For information about writing custom calculation scripts, see Performing Custom Calculations and Allocations on Aggregate Storage Databases in *Designing and Maintaining Essbase Cubes*.

**pov <mdx-set>**
Required. Provide an MDX set defining the context region in which the calculation is performed. The calculation script will be executed once for every cross-product in the POV region.

**sourceregion <mdx-set>**
Required. Provide an MDX set specifying the region of the cube referred to by the formulas in the script. At a minimum, the source region should include all members from the right-hand sides of the assignment statements in the custom calculation script.

**target <mdx-tuple>**
Optional. Provide an MDX tuple defining the database region where results are written. You can use only stored, level-0 members in the tuple; do not use upper-level members, attribute members, or dynamic calc members.

**debitmember <mdx-mbr>**
Optional. If double-entry accounting is used, provide an MDX member expression indicating the member to which positive result values are written. You can specify only stored, level-0 members; do not use upper-level members, attribute members, or dynamic calc members.

**creditmember <mdx-mbr>**
Optional. If double-entry accounting is used, provide an MDX member expression indicating the member to which negative result values are written. You can specify only stored, level-0 members; do not use upper-level members, attribute members, or dynamic calc members.

**offset <mdx-tuple>**
Optional. If offsetting entries are used, provide an MDX tuple defining the location in the database where an offsetting value for each source amount is written. You can use only stored, level-0 members in the tuple; do not use upper-level members, attribute members, or dynamic calc members.
override|add|subtract values
Optional. Generated calculation values can be added to (or subtracted from) existing values, instead of overwriting them. Overwriting is the default.

Notes

• Each clause can only be entered once.
• The script_file, pov, and sourceregion clauses are mandatory; the others are optional.
• The optional clauses following the sourceregion specification can be entered in any order, each separated by white space.
• You can specify only stored, level-0 members on the left side of the assignment statement in the custom calculation script; do not use upper-level members, attribute members, or dynamic calc members.
• You can specify only stored, level-0 members in the following clauses: DebitMember, CreditMember, Target, and Offset.

Example

The following statement executes script.txt referenced above. For a sample use case, see Performing Custom Calculations and Allocations on Aggregate Storage Databases in Designing and Maintaining Essbase Cubes.

execute calculation on database app.db with
  local script_file "script.txt"
  POV "Crossjoin({[VisionUS]},
    Crossjoin({[101]},
      Crossjoin({[Jan]},
        Crossjoin({[Scenario]},
          Descendants(Geography, Geography.Levels(0))))))"
  SourceRegion "Crossjoin({[AccountB], [AccountD]},
    Crossjoin({[Proj1], [Proj2]}, {[2007]})"
  Target "(Allocation)"
  DebitMember "[BeginningBalance_Debit]"
  CreditMember "[BeginningBalance_Credit]"
  Offset "([Account_000], [Project_000])"
add values;

Export Data (Aggregate Storage)

Click here for non-aggregate storage version

Export level-0 data, which does not include calculated values, from an aggregate storage database. Export data files are written to the application directory, unless an absolute path is specified. To use Report Writer, export the data using a report file.

Minimum permission required: Read.

Syntax
On aggregate storage databases, use export data to export in the following ways:

Keywords

export database <dbs-name> level0 data...
Export level-0 input data to a text file. You cannot export aggregates, upper level data, or data from dynamically calculated members.

export database <dbs-name> input data...
This statement performs the same action as export database <dbs-name> level0 data....

export database <dbs-name> ... data anonymous
Export data in anonymized format. Anonymization removes the risk of sensitive data disclosure, and can be used in case sample data needs to be provided for technical support. Essbase replaces real data values with 1, for each value in the block.

export database <dbs-name> ...using...report_file...
Run a stored report script, exporting a subset of the database.

Notes

• This statement requires the database to be started.

• Exports on aggregate storage databases are limited as follows:
  – You can export level-0 data only (level-0 data is the same as input data in aggregate storage databases).
  – You cannot perform upper-level data export on an aggregate storage database.
You cannot perform columnar export on an aggregate storage database.

To export data in parallel, specify a comma-separated list of export files, from 1 to 8 file names. This number should generally be equal to the number of processors on the machine that you wish to commit to doing parallel export. The number of threads Essbase uses typically depends on the number of file names you specify. However, on a very small aggregate storage database with a small number of data blocks, it is possible that only a single file will be created (in effect, performing serial export), even though parallel export to multiple files is requested. In this case, the export file name will be the first file name given as input.

During a data export, the export process allows users to connect and perform read-only operations.

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 exportfile.txt, the next additional file is exportfile_1.txt.

Example

Example 1

The following example exports all level 0 data from ASOsamp.Sample to an export file.

```
export database ASOsamp.Sample data to server data_file 'myfilesamp.txt';
```

Example 2

The following example uses a report script, Bottom.rep, to export a subset of sorted data from ASOsamp.Sample to an output file, Bottom.rpt.

```
export database ASOsamp.Sample using report_file 'Bottom.rep' to data_file 'Bottom.rpt';
```

Sample Report Script and Output

For example 2, assume that Bottom.rep is the following report script file based on ASOsamp.Sample:

```
//Bottom.rep
<Sym
<Column (Measures, Years)
<Row (Geography, Products)
<ICHILDREN Geography
<ICHILDREN Products
<Bottom (3, @DataColumn(1))
```
The report script produces the following report (Bottom.rpt):

Measures Years Time Transaction Type Payment Type Promotions Age Income
Level Stores

<table>
<thead>
<tr>
<th>Region</th>
<th>Category</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>North East</td>
<td>All Merchandise</td>
<td>43,250,241</td>
</tr>
<tr>
<td></td>
<td>Products</td>
<td>43,250,241</td>
</tr>
<tr>
<td></td>
<td>High End Merchandise</td>
<td>11,379,402</td>
</tr>
<tr>
<td>South</td>
<td>All Merchandise</td>
<td>32,790,838</td>
</tr>
<tr>
<td></td>
<td>Products</td>
<td>32,790,838</td>
</tr>
<tr>
<td></td>
<td>High End Merchandise</td>
<td>8,436,598</td>
</tr>
<tr>
<td>Geography</td>
<td>All Merchandise</td>
<td>76,041,079</td>
</tr>
<tr>
<td></td>
<td>Products</td>
<td>76,041,079</td>
</tr>
<tr>
<td></td>
<td>High End Merchandise</td>
<td>19,816,000</td>
</tr>
</tbody>
</table>

Export Query Tracking ( Aggregate Storage )

Export query data from an aggregate storage database to a text file.

Query data can be used to select the most appropriate set of aggregate views to materialize for an aggregate storage database. When an aggregate storage database is refreshed or restarted, query data is not persisted in the database. To rebuild aggregate views after a database refresh or restart, query data can be exported to and then imported from a text file.

Permission required: Database Manager.

Syntax

```
export query_tracking <dbs-name> to server file <file-name>
```

- **DBS-NAME**
- **FILE-NAME**

You can use `export query_tracking` in the following ways.

Keywords

```
export query_tracking <dbs-name> to server file...
```

Export query data from the specified aggregate storage database to the specified file. For FILE-NAME, specify the name, including the path, of the text file that contains the query data to export. If a path is not specified, the file is created in the `$ARBORPATH/app/appname/dbname` folder.

```
export query_tracking <dbs-name> to file...
```

You can omit the `server` keyword, but the result is the same.

Notes

- To export and import query data, query tracking must be enabled for the aggregate storage database. Use the `alter database` (aggregate storage)
statement with the `enable query_tracking` grammar. Query tracking is disabled by default.

- Do not edit the text file with the exported query data.

**Example**

```plaintext
export query_tracking ASOsamp.Sample to server file 'query_data_aso_sample.txt';
```

Exports query data from the ASOsamp.Sample database to the named file.

**See Also**

Import Query Tracking (Aggregate Storage)

**Import Data (Aggregate Storage)**

Click here for non-aggregate storage version

Import data from text or spreadsheet data files, with or without a rules file.

Minimum permission required: Write.

**Syntax**
Use import data in the following ways to load data into an aggregate storage database:

- **DBS-NAME**
- **BUFFER-ID**
- **IMP-FILE**
- **RULE-FILE-NAME**
- **FILE-NAME**
Keywords

**import database <dbs-name> data from...**
Specify whether the data import is from a local or server file, and what type of file to import data from.

...**using ... rules_file**
Import data into the database using a specified rules file.

...**<data error spec> (on error...)**
Required. Tell Essbase what to do in case of errors during the data load: abort the operation, or write or append to a specified error log.

...**<data record spec> from data_string**
Load a single data record into the selected database. The string following `data_string` must be a contiguous line, without newline characters.

...**<SQL connect spec> (connect as...)**
If you are importing data from an SQL source, provide your SQL user name and password. You must always use a rules file when you load SQL data sources. When loading SQL data into aggregate storage databases, you can use up to eight rules files to load data in parallel by using the `multiple rules_file` grammar with the grammar specified in `<buffer-block-spec>`. Essbase initializes multiple temporary aggregate storage data load buffers (one for each rules file) and, when the data is fully loaded into the buffers, commits the contents of all buffers into the database in one operation.

Each rules file must use the same authentication information (SQL user name and password).

In the following example, SQL data is loaded from two rules files (`rule1.rul` and `rule2.rul`):

```plaintext
import database ASOsamp.Sample data
  connect as TBC identified by 'password'
  using multiple rules_file 'rule1','rule2'
  to load_buffer_block starting with buffer_id 100
  on error write to "error.txt";
```

In specifying the list of rules files, use a comma-separated string of rules file names (excluding the `.rul` extension). The file name for rules files must not exceed eight bytes and the rules files must reside on Essbase Server.

In initializing a data load buffer for each rules file, Essbase uses the starting data load buffer ID you specify for the first rules file in the list (for example, ID 100 for rule1) and increments the ID number by one for each subsequent data load buffer (for example, ID 101 for rule2).

The ODBC driver you are using must be configured for parallel SQL connections.
Note:
Performing multiple SQL data loads in parallel to aggregate storage databases is different than using the `to load_buffer with buffer_id` grammar to load data into a buffer, and then using the `from load_buffer with buffer_id` grammar to explicitly commit the buffer contents to the database. For more information on aggregate storage data load buffers, see Loading Data into Aggregate Storage Databases in *Designing and Maintaining Essbase Cubes*.

...to load_buffer with buffer_id
If you are importing data from multiple data files to an aggregate storage database, you can import to a buffer first, in order to make the data import operation more efficient.

...from load_buffer with buffer_id
If you are importing data from multiple data files to an aggregate storage database, you can import from a data load buffer in order to make the data import operation more efficient.

...from load_buffer with buffer_id...values
Specify whether you want to add to existing values, subtract from existing values, or override existing values when committing the contents of the specified data load buffer to the database.

...from load_buffer with buffer_id...create slice
Commit the contents of the specified data load buffer to the database by creating a new data slice.

...from load_buffer with buffer_id override all data
Remove the current contents of the database and replace the database with the contents of the specified data load buffer.

...from load_buffer with buffer_id override incremental data
Remove the current contents of all incremental data slices in the database and create a new data slice with the contents of the specified data load buffer. The new data is created with the data load property "add values" (aggregate_sum). If there are duplicate cells between the new data and the primary slice, their values are added together when you query for them.

Notes
• This statement requires that the database is started.
• When using the import statement, you must specify what should happen in case of an error.
• To import from a SQL data source, you must connect as the relational user name and use a rules file.

Example

import database ASOsamp.Sample data from server data_file '/ASOsamp/Sample/expsamp.txt' on error abort;
Loads data into the ASOsamp.Sample database.

import database ASOsamp.Sample data from load_buffer with buffer_id 1;

Commits the contents of a specified data load buffer to the ASOsamp.Sample database.

import database ASOsamp.Sample data from load_buffer with buffer_id 1, 2;

Commits the contents of multiple data load buffers (buffer_id 1 and buffer_id 2) to the ASOsamp.Sample database.

import database ASOsamp.Sample data from load_buffer with buffer_id 1 add values;

Commits the contents of a specified data load buffer to the ASOsamp.Sample database by adding values.

import database ASOsamp.Sample data from load_buffer with buffer_id 1 override values create slice;

Commits the contents of the specified data load buffer into a new data slice in the ASOsamp.Sample database.

import database ASOsamp.Sample data from load_buffer with buffer_id 1 override all data;

Replaces the contents of the ASOsamp.Sample database with the contents of the specified data load buffer.

import database ASOsamp.Sample data from load_buffer with buffer_id 1 override incremental data;

Replaces the contents of all incremental data slices in the ASOsamp.Sample database by creating a new data slice with the contents of the specified data load buffer. The new data is created with the data load property "add values" (aggregate_sum). If there are duplicate cells between the new data and the primary slice, their values are added together when you query for them.

See Loading Data Using Buffers.

Import Query Tracking (Aggregate Storage)

Import query data, which was previously exported from an aggregate storage database to a text file, to an aggregate storage database.

Query data can be used to select the most appropriate set of aggregate views to materialize for an aggregate storage database. When an aggregate storage database is refreshed or restarted, query data is not persisted in the database. To rebuild aggregate views after a database refresh or restart, query data can be exported to and then imported from a text file.
Permission required: Database Manager.

Syntax

```plaintext
import query_tracking <dbs-name> from server file FILE-NAME
```

- **DBS-NAME**
- **FILE-NAME**

You can use `import query_tracking` in the following ways.

**Keywords**

```plaintext
import query_tracking <dbs-name> from server file...
```
Import query data to the specified aggregate storage database from the specified file. For `FILE-NAME`, specify the name, including the path, of the text file that contains the query data to import. By default, the file is created in the database directory.

```plaintext
import query_tracking <dbs-name> from file...
```
You can omit the `server` keyword, but the result is the same.

**Notes**

- To export and import query data, query tracking must be enabled for the aggregate storage database. Use the `alter database (aggregate storage)` statement with the `enable query_tracking` grammar. Query tracking is disabled by default.
- Do not edit the text file with the exported query data.

**Example**

```plaintext
import query_tracking ASOsamp.Sample from server file "'$ARBORPATH/app/ASOsamp/Sample/query_data_aso_sample.txt'";
```

Imports the query data from the named file to the ASOsamp.Sample database.

**See Also**

- Export Query Tracking (Aggregate Storage)

**Query Application (Aggregate Storage)**

Click here for block storage version

Get information about the current state of the application.

This statement is only applicable for aggregate storage applications.

This statement requires the application to be started.
Syntax

```
query application APP-NAME get cache_size list aggregate_storage storage_info
```

**APP-NAME**

**Example**

The following MaxL statement:

```
query application sample get cache_size;
```

returns the maximum size (in kilobytes) to which the aggregate storage cache may grow.

The following MaxL statement:

```
query application asoapp list aggregate_storage storage_info;
```

returns the following information:

<table>
<thead>
<tr>
<th><strong>Table 5-18  Query Application List Aggregate Storage Storage_Info MaxL Output Columns</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Output Columns</strong></td>
</tr>
<tr>
<td>Cache hit ratio</td>
</tr>
<tr>
<td>Current cache size (KB)</td>
</tr>
<tr>
<td>Current cache size limit (KB)</td>
</tr>
<tr>
<td>Page reads since last startup</td>
</tr>
</tbody>
</table>

**Note:**

This statistic may not be accurate when parallel data load or parallel calculation operations are in use.
Table 5-18  (Cont.) Query Application List Aggregate Storage Information

<table>
<thead>
<tr>
<th>Output Columns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page writes since last startup</td>
<td>Number of data blocks (pages) written to disk since the last time the application was started.</td>
</tr>
<tr>
<td>Page size (KB)</td>
<td>Size of the data block (page) in kilobytes.</td>
</tr>
<tr>
<td>Disk space allocated for data (KB)</td>
<td>Total space used by all disk files in the default tablespace.</td>
</tr>
<tr>
<td>Disk space used by data (KB)</td>
<td>Total space actually in use within the disk files in the default tablespace (some space within files may be free).</td>
</tr>
<tr>
<td>Temporary disk space allocated (KB)</td>
<td>Total space used by all disk files in the temp tablespace.</td>
</tr>
<tr>
<td>Temporary disk space used (KB)</td>
<td>Total space actually in use within the disk files in the temp tablespace (some space within files may be free).</td>
</tr>
</tbody>
</table>

Query Database (Aggregate Storage)

Click here for non-aggregate storage version

Get advanced information about the current state of the database.

Minimum permission required: Read.

This statement requires the database to be started.

Syntax
You can query for database information in the following ways using `query database`:

**Keywords**

**get active alias_table**
Display the active alias table for the user issuing the statement.

**get attribute_info**
Get attribute member, dimension, and name information for the specified attribute member.

**get attribute_spec**
Display the current attribute specifications for the database. These specifications include attribute member name format, Attribute Calculation dimension member names, Boolean and date member names, and numeric range specifications.

**get cube_size_info**
Display information about input data size, aggregated data size, and number of queries tracked (when query tracking is enabled).

This statement returns the output listed in the following table:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>input_data_size_cells</td>
<td>Number of input-level cells in the cube.</td>
</tr>
<tr>
<td>input_data_size_bytes</td>
<td>Number of bytes used by the input-level data (approximate).</td>
</tr>
<tr>
<td>aggregate_data_size_cells</td>
<td>Total number of cells in all aggregate views in the cube.</td>
</tr>
<tr>
<td>aggregate_data_size_bytes</td>
<td>Number of bytes used by the aggregate cells (approximate).</td>
</tr>
<tr>
<td>kernel_queries_tracked</td>
<td>Number of kernel queries executed since the last time query tracking was enabled or query tracking information was reset.</td>
</tr>
<tr>
<td>total_query_cost</td>
<td>Total cost of all queries executed since the last time query tracking information was reset.</td>
</tr>
<tr>
<td>Column Name</td>
<td>Contents</td>
</tr>
<tr>
<td>---------------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>query_tracking_enabled</td>
<td>Values: True or False. Tells whether user retrieval statistics are being collected for the aggregate storage database. The statistics can be used by the following MaxL statements for query-based view optimization:</td>
</tr>
<tr>
<td></td>
<td>• query database &lt;dbs-name&gt; list existing_views</td>
</tr>
<tr>
<td></td>
<td>• execute aggregate process</td>
</tr>
<tr>
<td></td>
<td>• execute aggregate selection</td>
</tr>
<tr>
<td></td>
<td>Query tracking is disabled by default.</td>
</tr>
</tbody>
</table>

**get dbstats dimension**
Get information about dimensions.
The **index_type** field values are numeric, and translate as follows:

- 0               Dense
- 1               Sparse
- 3               None (database is aggregate storage)

**get dbstats data_block**
Get information about data blocks. The information returned has little relevance to aggregate storage databases.

**get member_info <MEMBER-NAME>**
Get information on a specific member.

**Output**
The **unary_type** field values are numeric, and translate as follows:

- 0               Add
- 1               Subtract
- 2               Multiply
- 3               Divide
- 4               Percent
- 5               NoRollUp

The **member_tag_type** field values translate as follows:

- 0               SkipNone
- 16384           SkipMissing
- 32768           SkipZero
- 49152           SkipBoth
- 1               BalFirst
- 2               BalLast
- 4               TwoPass
- 8               Average
- 64              Expense
Variations are possible. The field value consists of one of the first four "skip" values plus any/all/none of the last five values. Some examples:

<table>
<thead>
<tr>
<th>Value</th>
<th>Field Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SkipNone</td>
</tr>
<tr>
<td>77</td>
<td>SkipNone, BalFirst, TwoPass, Average, Expense</td>
</tr>
<tr>
<td>16385</td>
<td>SkipMissing and BalFirst</td>
</tr>
</tbody>
</table>

The first four "skip" values are base values, and added to them are combinations of 1, 2, 4, 8, and 64.

The status field values are hexadecimal, and translate as follows:

<table>
<thead>
<tr>
<th>Value</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Normal</td>
</tr>
<tr>
<td>1</td>
<td>Never Share</td>
</tr>
<tr>
<td>2</td>
<td>Label</td>
</tr>
<tr>
<td>4</td>
<td>Refer Share</td>
</tr>
<tr>
<td>8</td>
<td>Refer Share (with different name)</td>
</tr>
<tr>
<td>16</td>
<td>Implicit share</td>
</tr>
<tr>
<td>32</td>
<td>Virtual Member (stored)</td>
</tr>
<tr>
<td>64</td>
<td>Virtual Member (not stored)</td>
</tr>
<tr>
<td>2048</td>
<td>Attribute</td>
</tr>
<tr>
<td>32768</td>
<td>Referred</td>
</tr>
</tbody>
</table>

**get opg_state of member_data**
Display outline navigational information (for example, parent, child, or sibling), fixed-length information (for example, the line aggregation symbol or the number of children), and text strings (for example, member names or aliases).
See Outline Paging Dimension Statistics for a description of the output.

**get opg_state of member_name_namespace**
Display information that matches member names to internal member identifiers (one section per database, thus the information for all dimensions is the same).
See Outline Paging Dimension Statistics for a description of the output.

**get opg_state of member_formula**
Display all formulas for the dimension.
See Outline Paging Dimension Statistics for a description of the output.

**get opg_state of member_UDA**
Display all user defined attributes (UDAs) for the dimension.
See Outline Paging Dimension Statistics for a description of the output.

**get opg_state of member_UDA_namespace**
Display information that matches UDAs to internal member identifiers.
See Outline Paging Dimension Statistics for a description of the output.

**get opg_state of attribute_to_base_member_association**
Display information that identifies the attribute member associated with each base member of the dimension.
See Outline Paging Dimension Statistics for a description of the output.

**get opg_state of member_comment**
Display all member comments for the dimension.
See Outline Paging Dimension Statistics for a description of the output.
get opg_state of member_alias_namespace
Display information that matches member alias names to internal member identifiers (one section per alias table, thus the information for all dimensions is the same). See Outline Paging Dimension Statistics for a description of the output.

list aggregate_storage runtime_info
Display runtime statistics about the aggregate storage database. For a description of the output returned by this statement, see Aggregate Storage Runtime Statistics.

list aggregate_storage group_id_info
Display information about group IDs and their timestamps related to General Ledger cubes.

Note:
This grammar applies to General Ledger cubes, not to non-general-ledger aggregate storage databases. For normal aggregate storage databases, this table will be empty.
This MaxL grammar is disabled for previous release Essbase MaxL clients.

This statement returns the following output:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>group_id</td>
<td>The allocation group id, according to the begin allocation command that created the allocation group. The number is an unsigned 64-bit integer.</td>
</tr>
<tr>
<td>transaction_id</td>
<td>The aggregate storage transaction ID that is used internally. The number is an unsigned 64-bit integer.</td>
</tr>
<tr>
<td>state</td>
<td>A string describing the state of the group ID. For example: BeginAllocation Done, Allocation In Progress, Allocation Done, EndAllocation In Progress.</td>
</tr>
<tr>
<td>time_last_used</td>
<td>The date and time the group ID was last used. The value is either the time the group ID was created or the time that an allocation or custom calculation was last performed with this group ID. The value is a string.</td>
</tr>
<tr>
<td>time_expired</td>
<td>The date and time when the group ID will time out (expire). The value is a string.</td>
</tr>
</tbody>
</table>
The column "expired" indicates whether the group ID has timed out. If the group ID has expired, the group ID will be rolled back the next time a begin allocation command is executed. The value is a boolean.

For a description of the output returned by this statement, see Aggregate Storage Group ID Information Output.

**list aggregate_storage slice_info**

Display information about data slices and views, some information of which applies only to General Ledger cubes (not to non-general-ledger aggregate storage databases).

**Note:**

Small incremental slices may have fewer aggregate views than the primary slice (slice number 0). Incremental slices with less than 100,000 cells will never have any aggregate views built. However, if an incremental slice is larger than 100,000 cells and it is larger than the primary slice, then it will always have the same aggregate views as the primary slice.

This MaxL grammar is disabled for previous release Essbase MaxL clients.

This statement returns the following output:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>transaction_id</td>
<td><em>(Applies to General Ledger cubes only)</em>&lt;br&gt;The ID of the transaction to which this slice and view belong. There is one transaction ID for each GL group ID. The number is an unsigned 64-bit integer.&lt;br&gt;To find the corresponding group ID, use the following MaxL command: query database app.db list aggregate_storage group_id_info;</td>
</tr>
<tr>
<td>slice_id</td>
<td>ID number of the data slice.&lt;br&gt;The number is an unsigned 32-bit integer.</td>
</tr>
</tbody>
</table>

For non-general-ledger aggregate storage databases, this number is always 0.
### Column Name | Contents
--- | ---
slice_tag | (Applies to General Ledger cubes only)
When an allocation or custom calculation is done within an allocation begin/end, this number is the rule_id of the allocation that made this data slice. The number is an unsigned 64-bit integer. For non-general-ledger aggregate storage databases, this number is always 0.

view_id | 0 indicates an input view; otherwise, the view is an aggregate view. The number is an unsigned 64-bit integer. To list the levels in a given aggregate view, use the following MaxL command:

```plaintext
query database app.db list existing_views;
```

size_cells | The number of cells in the given view of the slice. The number is an unsigned 64-bit integer.

size_kb | The size in KB of the given view of the slice. The number is an unsigned 64-bit integer.

---

For a description of the output returned by this statement, see [Aggregate Storage Slice Information Output](#).

**list aggregate_storage uncommitted_transaction_info**

Display information about uncommitted transactions that are related to General Ledger cubes.

**Note:**

This grammar applies to General Ledger cubes, not to non-general-ledger aggregate storage databases. For normal aggregate storage databases, this table will be empty.

This MaxL grammar is disabled for previous release Essbase MaxL clients.

This statement returns the following output:
For a description of the output returned by this statement, see Aggregate Storage Uncommitted Transaction Information Output.

**list aggregate_storage compression_info**

Display estimated compression for aggregate storage databases when different dimensions are hypothetically used as the compression dimension. These estimates can help you choose the best dimension to use as the compression dimension. In aggregate storage databases, the compression dimension enables database compression. A good candidate for a compression dimension is one that optimizes data compression and maintains retrieval performance. The following table lists data for all non-attribute dimensions, even though it may not be possible to select them as the compression dimension without significant changes to the outline. For information on the requirements of a compression dimension, see Understanding the Compression Dimension for Aggregate Storage Databases in Designing and Maintaining Essbase Cubes.

This statement returns the following output:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>dimension_name</td>
<td>Each dimension name in the database, hypothetically considered to be the compression dimension.</td>
</tr>
<tr>
<td>is_compression</td>
<td>Indicates whether the dimension is the aggregate storage compression dimension. (There can be only one compression dimension in an aggregate storage database.)</td>
</tr>
<tr>
<td>stored_level0_members</td>
<td>The number of leaf-level members in the dimension. A large number of stored level-0 members in a dimension indicates that it may not perform well as a compression dimension.</td>
</tr>
<tr>
<td>Column Name</td>
<td>Contents</td>
</tr>
<tr>
<td>------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>average_bundle_fill</td>
<td>Estimated average number of values per compression dimension bundle. Choosing a compression dimension that has a higher average bundle fill means that the database compresses better.</td>
</tr>
<tr>
<td>average_value_length</td>
<td>Estimated average number of bytes required to store a value. Dimensions with a smaller average value length compress the database better.</td>
</tr>
<tr>
<td>level0_mb</td>
<td>Estimated size of the compressed database, in megabytes. A smaller expected level-0 size indicates that choosing this dimension enables better compression. Except for the scenario in which there is no compression dimension (None), all estimates assume that all pages are compressed. Since compressed pages require additional overhead that uncompressed pages do not, the estimated level-0 database size for some dimensions may be larger than the value for None.</td>
</tr>
</tbody>
</table>

**list alias_table**
Get a list of alias tables that are defined for the database.

**list alias_names in alias_table**
List the alias names defined in an alias table. Alias tables contain sets of aliases for member names and are stored in the database outline. Use this grammar to see a list of alias names defined in the specified table.

**list existing_views**
Display information about all aggregate views. An aggregate view is a collection of aggregate cells based on the levels of the members within each dimension. The optional `based on query_data` clause causes the returned query cost information to be based on the collected cost of actual user queries. If this clause is not used, the default assumption is that all possible queries happen with the same probability.
To use the `based on query_data` clause, query tracking must first be enabled. To enable query tracking, use `alter database <dbs-name> enable query tracking`.

**list ... file information**
Get accurate index and data file information. Provides index and data file names, counts, sizes, and totals, and indicates whether or not each file is presently opened by Essbase. The file size information is accurate. Note that the file size information provided by the Windows operating system for index and data files that reside on NTFS volumes may not be accurate.
list load_buffers
Display a list and description of the data load buffers that exist on an aggregate storage database. See Using Aggregate Storage Data Load Buffers.

list aso_level_info
Display the aggregation level count for each real dimension in the outline. Aggregation level count is the total number of aggregation levels in a real dimension (including associated attribute dimensions) that exist on an aggregate storage database.

dump|force_dump existing views...
Saves existing views of this database to an aggregation script. This action requires a minimum permission of execute (Execute).
If the specified script name already exists, you can use the force_dump keyword to overwrite it; otherwise, an error is returned if the file name already exists.
If the based on query_data phrase is used, the view selection that is saved will be based on previously collected query-tracking data. You must have enabled query tracking to use this option. For more information about query tracking, see the based on query_data description in execute aggregate selection.

Example

query database ASOsamp.Sample list load_buffers;

Display a list and description of the data load buffers that exist on ASOsamp.Sample.

Outline Paging Dimension Statistics

The following columns are the output of the MaxL statement beginning with query database DBS-NAME get opg_state.

This statement is only applicable to databases using aggregate storage.

Table 5-24   Outline Paging Dimension Statistics MaxL Output Columns

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>version</td>
<td>The version of the outline paging section (a Berkeley DB database).</td>
</tr>
<tr>
<td>unique_keys</td>
<td>The number of unique keys in the outline paging section.</td>
</tr>
<tr>
<td>key/data_pairs</td>
<td>The number of key/data pairs in the outline paging section.</td>
</tr>
<tr>
<td>page_size</td>
<td>The page size (in bytes) of the underlying database.</td>
</tr>
<tr>
<td>minimum_keys_per_page</td>
<td>The minimum number of keys per page.</td>
</tr>
<tr>
<td>length of fixed_length_records</td>
<td>The length of the fixed-length records (only available when the outline paging section is a Recno database).</td>
</tr>
<tr>
<td>padding_byte_value_for_fixed_length_columns</td>
<td>The padding byte value for fixed-length records.</td>
</tr>
<tr>
<td>levels</td>
<td>Number of levels in the underlying database corresponding to the outline paging section.</td>
</tr>
</tbody>
</table>
Table 5-24  (Cont.) Outline Paging Dimension Statistics MaxL Output Columns

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>internal_pages</td>
<td>Number of internal pages in the underlying database.</td>
</tr>
<tr>
<td>leaf_pages</td>
<td>Number of leaf pages in the underlying database.</td>
</tr>
<tr>
<td>duplicate_pages</td>
<td>Number of duplicate pages in the underlying database.</td>
</tr>
<tr>
<td>overflow_pages</td>
<td>Number of overflow pages in the underlying database.</td>
</tr>
<tr>
<td>pages_on_free_list</td>
<td>Number of pages on the free list in the underlying database.</td>
</tr>
<tr>
<td>bytes_free_in_internal_pages</td>
<td>Number of bytes free in internal pages of the underlying database.</td>
</tr>
<tr>
<td>bytes_free_in_leaf_pages</td>
<td>Number of bytes free in leaf pages of the underlying database.</td>
</tr>
<tr>
<td>bytes_free_in_duplicate_pages</td>
<td>Number of bytes free in duplicate pages of the underlying database.</td>
</tr>
<tr>
<td>bytes_free_in_overflow_pages</td>
<td>Number of bytes free in overflow pages of the underlying database.</td>
</tr>
</tbody>
</table>

Aggregate Storage Runtime Statistics

Statistics per Dimension

The following MaxL statement:

```
query database asoapp.asodb list aggregate_storage runtime_info;
```

Returns output which includes the following lines:

<table>
<thead>
<tr>
<th>parameter</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimension [Year] has [3] levels, bits used</td>
<td>4</td>
</tr>
<tr>
<td>Dimension [Measures] has [1] levels, bits</td>
<td>4</td>
</tr>
<tr>
<td>Dimension [Product] has [3] levels, bits u</td>
<td>5</td>
</tr>
<tr>
<td>Dimension [Market] has [3] levels, bits us</td>
<td>5</td>
</tr>
<tr>
<td>Dimension [Scenario] has [1] levels, bits</td>
<td>2</td>
</tr>
</tbody>
</table>

For each dimension, the following statistics are shown:

- The name of the dimension.
- How many stored levels the dimension has, in the aggregate storage perspective. Not all levels are stored in aggregate storage databases; some are virtual levels.
- The number of bits being used in the key for the dimension.

Each cell in an aggregate storage database is stored as a key/value pair. The key length is 8 bytes or a multiple of 8 bytes; for example, 8, 16, 24.
Each key corresponds to a numeric value in the database. The number of bits each dimension uses in the dimensional key is shown in the value column for each dimension.

The number of bits used in each key may amount to less than the bytes needed for physical storage of the key. As an example where this knowledge might be useful, consider a case in which a key is using 65 bits. If you can reduce the key length by one bit to 64, then you can have the key length be 8 bytes instead of 16, an improvement which reduces the overall size of the database. Another use for these statistics might be to examine them to see how much you gain from removing any particular dimension.

Statistics for the Whole Database

The same MaxL statement used above also returns the following lines in its output:

<table>
<thead>
<tr>
<th>parameter</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max. key length (bits)</td>
<td>20</td>
</tr>
<tr>
<td>Max. key length (bytes)</td>
<td>8</td>
</tr>
<tr>
<td>Number of input-level cells</td>
<td>0</td>
</tr>
<tr>
<td>Number of incremental data slices</td>
<td>0</td>
</tr>
<tr>
<td>Number of incremental input cells</td>
<td>0</td>
</tr>
<tr>
<td>Number of aggregate views</td>
<td>0</td>
</tr>
<tr>
<td>Number of aggregate cells</td>
<td>0</td>
</tr>
<tr>
<td>Number of incremental aggregate cells</td>
<td>0</td>
</tr>
<tr>
<td>Cost of querying incr. data (ratio to total cost)</td>
<td>0</td>
</tr>
<tr>
<td>Input-level data size (KB)</td>
<td>0</td>
</tr>
<tr>
<td>Aggregate data size (KB)</td>
<td>0</td>
</tr>
</tbody>
</table>

The whole-database statistics are described in the following table.

Table 5-25 Aggregate Storage Runtime Statistics MaxL Output

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max. key length (bits)</td>
<td>The sum of all the bits used by each dimension. For example, there are 20 bits in the key used for dimensions, and the first 4 are used by Year.</td>
</tr>
<tr>
<td>Max. key length (bytes)</td>
<td>How many bytes the key uses per cell.</td>
</tr>
<tr>
<td>Number of input-level cells</td>
<td>The number of existing level-0 cells in the database, including incremental slices.</td>
</tr>
<tr>
<td>Number of incremental data slices</td>
<td>The number of data slices resulting from incremental data loads.</td>
</tr>
<tr>
<td>Number of incremental input cells</td>
<td>The number of level-0 cells in the incremental data slices. To see the number of unique aggregate views, use the MaxL statement: query database applname.dbname list existing_views;</td>
</tr>
</tbody>
</table>
### Table 5-25  (Cont.) Aggregate Storage Runtime Statistics MaxL Output

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of aggregate views</td>
<td>The number of aggregate views in the database, including those automatically built on incremental slices.</td>
</tr>
<tr>
<td>Number of aggregate cells</td>
<td>The number of cells stored in the database's aggregate views.</td>
</tr>
<tr>
<td>Number of incremental aggregate cells</td>
<td>The number of cells stored in the incremental slices' aggregate views.</td>
</tr>
<tr>
<td>Cost of querying incr. data (ratio to total cost)</td>
<td>The average percentage of query time spent processing incremental data slices. This functionality is useful in deciding when slices should be merged together to improve query performance.</td>
</tr>
<tr>
<td>Input-level data size (KB)</td>
<td>The total disk space used by input-level data.</td>
</tr>
<tr>
<td>Aggregate data size (KB)</td>
<td>The total disk space occupied by aggregate cells.</td>
</tr>
</tbody>
</table>

For input-level and aggregate cells, the above statistics show:

1. Number of cells
2. Disk space occupied by those cells

Because Essbase uses compression, these statistics are useful because it is not always possible to derive disk size based on the number of cells.

### Aggregate Storage Slice Information Output

The following MaxL statement:

```maxl
query database "dmglex4"."basic" list aggregate_storage slice_info;
```

Returns the following output:

```
transaction_id  slice_id  slice_tag  view_id  size_cells  size_kb
+----------------+---------+----------+--------+-----------+----------
  0             0         0        0          38         64
  3             1         66       0          21         32
  3             2         77       0          21         32
```

See [Query Database](#).

### Aggregate Storage Group ID Information Output

The following MaxL statement:

```maxl
query database "dmglex4"."basic" list aggregate_storage group_id_info;
```


Returns the following output:

<table>
<thead>
<tr>
<th>group_id</th>
<th>transaction_id</th>
<th>state</th>
<th>time_last_used</th>
<th>time_expired</th>
<th>expired</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>1</td>
<td>Allocation Done</td>
<td>Wed Jul 20 17:39:57</td>
<td>Wed Jul 20 17:44:57</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

See [Query Database](#).

### Aggregate Storage Uncommitted Transaction Information Output

The following MaxL statement:

```maxl
query database "dmglex4"."basic" list aggregate_storage uncommitted_transaction_info;
```

Returns the following output (columns are truncated):

<table>
<thead>
<tr>
<th>unc_trans</th>
<th>unc_data_</th>
<th>unc_input</th>
<th>unc_aggre</th>
<th>unc_aggre</th>
<th>unc_input</th>
<th>unc_aggre</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

See [Query Database](#).

### MaxL Use Cases

The following topics demonstrate some tasks that can be accomplished using MaxL.

- Creating an Aggregate Storage Sample Using MaxL
- Loading Data Using Buffers
- Using Aggregate Storage Data Load Buffers
- Forcing Deletion of Partitions
- Metadata Filtering
- Examples of Triggers

### Creating an Aggregate Storage Sample Using MaxL

Related MaxL statements: create application, create database, create outline, alter database, import data, execute aggregate process,
The following sample MaxL script creates an aggregate storage application and database based on Sample.Basic.

```maxl
login $1 $2;
spool on to 'maxl_log.txt';
create or replace application Sample2 using aggregate_storage
   comment 'aggregate storage version of Sample';
create database Sample2.Basic2
   comment 'aggregate storage version of Sample Basic';
create or replace outline on aggregate_storage database Sample2.Basic2
   as outline on database sample.basic;
alter database Sample2.Basic2 initialize load buffer with buffer_id 1;
import database Sample2.Basic2 data
   from server data_file '/catalog/users/catalogUser/Data.txt'
   to load_buffer with buffer_id 1
   on error abort;
import database Sample2.Basic2 data from load_buffer with buffer_id 1;
execute aggregate process on database Sample2.Basic2
   stopping when total_size exceeds 1.9;
spool off;
logout;
```

### Loading Data Using Buffers

**Related MaxL Statements**

- Alter Database (Aggregate Storage)
- Query Database (Aggregate Storage)
- Import Data (Aggregate Storage)

If you use multiple Import Data (Aggregate Storage) statements to load data values to aggregate storage databases, you can significantly improve performance by loading values to a temporary data load buffer first, with a final write to storage after all data sources have been read.

While the data load buffer exists in memory, you cannot build aggregations or merge slices, as these operations are resource-intensive. You can, however, load data to other data load buffers, and perform queries and other operations on the database. There might be a brief wait for queries, until the full data set is committed to the database and aggregations are created.

The data load buffer exists in memory until the buffer contents are committed to the database or the application is restarted, at which time the buffer is destroyed. Even if
the commit operation fails, the buffer is destroyed and the data is not loaded into the database.

Multiple data load buffers can exist on a single aggregate storage database. To save time, you can load data into multiple data load buffers at the same time by using separate MaxL Shell sessions. Although only one data load commit operation on a database can be active at any time, you can commit multiple data load buffers in the same commit operation, which is faster than committing buffers individually.

You can query the database for a list and description of the data load buffers that exist on an aggregate storage database. See Using Aggregate Storage Data Load Buffers.

Examples:

- Example: Load Multiple Data Sources into a Single Data Load Buffer
- Example: Perform Multiple Data Loads in Parallel

**Example: Load Multiple Data Sources into a Single Data Load Buffer**

Assume there are three data files that need to be imported. With aggregate storage databases, data loads are most efficient when all data files are loaded using one import operation. Therefore, load buffers are useful when loading more than one data file.

1. Use `Alter Database (Aggregate Storage)` to create a load buffer.

   ```
   alter database AS0samp.Sample
   initialize load_buffer with buffer_id 1;
   ```

2. Load data into the buffer, using the `Import Data (Aggregate Storage)` statement.

   ```
   import database AS0samp.Sample data
   from server data_file 'file_1'
   to load_buffer with buffer_id 1
   on error abort;
   ```

   ```
   import database AS0samp.Sample data
   from server data_file 'file_2'
   to load_buffer with buffer_id 1
   on error abort;
   ```

   ```
   import database AS0samp.Sample data
   from server data_file 'file_3'
   to load_buffer with buffer_id 1
   on error abort;
   ```

3. Move the data from the buffer into the database.

   ```
   import database AS0samp.Sample data
   from load_buffer with buffer_id 1;
   ```

   The data-load buffer is implicitly destroyed.

4. Assume that in Step 2, after loading 'file_2' into the load buffer, you decided not to load the data. Because the data is in a buffer and not yet in the database, you
would simply use Alter Database (Aggregate Storage) to destroy the buffer without moving the data to the database.

```sql
alter database ASOsamp.Sample
destroy load_buffer with buffer_id 1;
```

**Example: Perform Multiple Data Loads in Parallel**

1. In one MaxL Shell session, load data into a buffer with an ID of 1:

```sql
alter database ASOsamp.Sample
initialize load_buffer with buffer_id 1 resource_usage 0.5;

import database ASOsamp.Sample data
from data_file "dataload1.txt"
to load_buffer with buffer_id 1
on error abort;
```

2. Simultaneously, in another MaxL Shell session, load data into a buffer with an ID of 2:

```sql
alter database ASOsamp.Sample
initialize load_buffer with buffer_id 2 resource_usage 0.5;

import database ASOsamp.Sample data
from data_file "dataload2.txt"
to load_buffer with buffer_id 2
on error abort;
```

3. When the data is fully loaded into the data load buffers, use one MaxL statement to commit the contents of both buffers into the database by using a comma separated list of buffer IDs:

```sql
import database ASOsamp.Sample data
from load_buffer with buffer_id 1, 2;
```

**Using Aggregate Storage Data Load Buffers**

**Related MaxL Statement:**

**Query Database (Aggregate Storage)**

Use the following MaxL statement to get a list and description of the data load buffers that exist on an aggregate storage database.

```sql
query database appname.dbname list load_buffers;
```

This statement returns the following information about each existing data load buffer:
Table 5-26  List Load Buffers MaxL Output Columns

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>buffer_id</td>
<td>ID of a data load buffer (a number between 1 and 4294967296).</td>
</tr>
<tr>
<td>internal</td>
<td>A Boolean that specifies whether the data load buffer was created internally by Essbase (TRUE) or by a user (FALSE).</td>
</tr>
<tr>
<td>active</td>
<td>A Boolean that specifies whether the data load buffer is currently in use by a data load operation.</td>
</tr>
<tr>
<td>resource_usage</td>
<td>The percentage (a number between .01 and 1.0 inclusive) of the aggregate storage cache that the data load buffer is allowed to use.</td>
</tr>
<tr>
<td>aggregation method</td>
<td>One of the methods used to combine multiple values for the same cell within the buffer:</td>
</tr>
<tr>
<td></td>
<td>• AGGREGATE_SUM: Add values when the buffer contains multiple values for the same cell.</td>
</tr>
<tr>
<td></td>
<td>• AGGREGATE_USE_LAST: Combine duplicate cells by using the value of the cell that was loaded last into the load buffer.</td>
</tr>
<tr>
<td>ignore_missings</td>
<td>A Boolean that specifies whether to ignore #MI values in the incoming data stream.</td>
</tr>
<tr>
<td>ignore_zeros</td>
<td>A Boolean that specifies whether to ignore zeros in the incoming data stream.</td>
</tr>
</tbody>
</table>

Forcing Deletion of Partitions

The force keyword used at the end of the drop partition statement specifies that the source half of a partition definition should be dropped regardless of whether the target half is missing or invalid.

For example, in the following session, assume there is a partition definition between app1.source and app2.target, but the app2.target database has been dropped. An ordinary attempt to drop the partition definition fails:

MAXL> drop transparent partition app1.source to app2.target;

OK/INFO - 1053012 - Object source is locked by user system.
OK/INFO - 1051034 - Logging in user System.
OK/INFO - 1051035 - Last login on Friday, January 10, 2005 2:28:09 PM.
ERROR - 1051032 - Database target does not exist.
OK/INFO - 1053013 - Object source unlocked by user system.
OK/INFO - 1051037 - Logging out user system, active for 0 minutes.
In the second attempt, the **force** keyword allows the invalid source partition to be dropped:

```
MAXL> drop transparent partition appl.source to app2.target force;
```

```
OK/INFO - 1053012 - Object source is locked by user system.
OK/INFO - 1051034 - Logging in user System.
OK/INFO - 1051035 - Last login on Friday, January 10, 2005 2:31:50 PM.
ERROR - 1051032 - Database target does not exist.
OK/INFO - 1051037 - Logging out user system, active for 0 minutes.
OK/INFO - 1053013 - Object source unlocked by user system.
OK/INFO - 1241125 - Partition dropped.
```

**Note:**

The force keyword only works to drop a partition definition when the source half of the partition definition remains valid. In other words, if the source database is deleted, the partition cannot be dropped from the dangling target.

---

**Metadata Filtering**

Related MaxL statements: `create filter`, `alter filter`.

Metadata filtering provides an additional layer of security in addition to data filtering. With metadata filtering, an administrator can remove outline members from a user's view, providing access only to those members that are of interest to the user.

When a filter is used to apply MetaRead permission on a member,

1. Data for all ancestors of that member are hidden from the filter user's view.
2. Data and metadata (member names) for all siblings of that member are hidden from the filter user's view.

**Example**

The following report script for Sample.Basic:

```
//Meta02.rep

<COLUMN (Year, Product)

<CHILDREN Cola

<ROW (Market)

<ICHILDREN West

! 

under normal unfiltered conditions returns

Year 100-10 Measures Scenario
California 3,498
```
Oregon                159  
Washington            679  
Utah                  275  
Nevada                (18)  
West                  4,593  

But with the following filter granted to an otherwise read-access user,

create or replace filter sample.basic.meta02  
  meta_read on "California","Oregon"  
;

the report script then returns:

    Year 100-10 Measures Scenario  
California          3,498  
Oregon                159  
West           #Missing  

In summary, MetaRead permission on California and Oregon means that:

1. The affected user can see no data for ancestors of California and Oregon members. West data shows only #Missing (or #NoAccess, in a grid client interface).

2. The affected user can see no sibling metadata (or data) for siblings of California and Oregon. In other words, the user sees only the western states for which the filter gives MetaRead permission.

Overlapping Metadata Filter Definitions

You should define a MetaRead filter using multiple rows only when the affected member set in any given row (the metaread members and their ancestors) has no overlap with MetaRead members in other rows. Oracle recommends that you specify one dimension per row in filters that contain MetaRead on multiple rows. However, as long as there is no overlap between the ancestors and MetaRead members, it is still valid to specify different member sets of one dimension into multiple MetaRead rows.

For example, in Sample.Basic, the following filter definition has overlap conflicts:

Table 5-27 Sample Filter with Overlap Conflicts

<table>
<thead>
<tr>
<th>Access</th>
<th>Member Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>MetaRead</td>
<td>California</td>
</tr>
<tr>
<td>MetaRead</td>
<td>West</td>
</tr>
</tbody>
</table>

In the first row, applying MetaRead to California has the effect of allowing access to California but blocking access to its ancestors. Therefore, the MetaRead access to West is ignored; users who are assigned this filter will have no access to West.

If you wish to assign MetaRead access to West as well as California, then the appropriate method is to combine them into one row:
Examples of Triggers

Related MaxL statements: alter trigger, create trigger, display trigger, drop trigger.

The following examples are based on the Sample.Basic database.

**Note:**
You cannot define a trigger that requires data from Dynamic Calc members or members from another partition.

**Example 1: Tracking Sales for January**

Example 1 tracks the Actual, Sales value for the following month, product, and region:

- January (Year dimension member Jan)
- Colas (Product dimension member 100)
- In the Eastern region (Market dimension member East)

When the current member being calculated is Jan, and when the Actual, Sales value of Colas for January exceeds 20, the example logs an entry in the file Trigger_jan_Sales.

```maxl
create or replace trigger Sample.Basic.Trigger_Jan_20
Where
  {{Jan,Sales,[100],East,Actual}}
When
  Jan > 20 AND Is(Year.CurrentMember, Jan)
then spool Trigger_Jan_20
end;
```

**Example 2: Tracking Sales for Quarter 1**

Example 2 tracks the Actual, Sales value for the following months, product, and region:

- January, February, March (The children of Year dimension member Qtr1)
- Colas (Product dimension member 100)
- In the Eastern region (Market dimension member East)

When the current member being calculated is Jan, Feb or Mar, and when the Actual, Sales value of Colas for any of the the months January, February, or March exceeds
20, the example logs an entry in the file Trigger_Jan_Sales_20,
Trigger_Feb_Sales_20, or Trigger_Mar_Sales_20.

create or replace trigger Sample.Basic.Trigger_Qtr1_Sales
Where
Crossjoin(
    {Qtr1.children},
    {([Measures].[Sales], [Product].[100], [Market].[East], [Scenario].
    [Actual])})
) When
Year.Jan > 20 and is(Year.currentmember, Jan) then spool Trigger_Jan_Sales_20 When
Year.Feb > 20 and is(Year.currentmember, Feb) then spool Trigger_Feb_Sales_20 When
Year.Mar > 20 and is(Year.currentmember, Mar) then spool Trigger_Mar_Sales_20 end;

Example 3: Tracking Inventory Level

Example 3 tracks the inventory level for the following product, region, and months:

- Colas (product 100)
- In the eastern region (market East)
- For January, February, and March (the children of Qtr1)

If the inventory of Colas in the eastern region falls below 500,000, the example trigger sends an email to recipient@example.com.

create or replace trigger Sample.Basic.Inventory_east
where CrossJoin(
    {Qtr1}.children,
    {([East],[100],[Ending Inventory])})
) when [Ending Inventory] < 500000 then
mail ([smtp_server.example.com],[sender@example.com],
    [recipient@example.com],
    [Subject of E-Mail])
end;
MDX

MDX is a language for anyone who needs to develop scripts or applications to query and report against data and metadata in Oracle Analytics Cloud – Essbase databases.

- Overview of MDX
- MDX Query Format
- MDX Syntax and Grammar Rules
- MDX Operators
- About MDX Properties
- MDX Comments
- MDX Query Limits
- Aggregate Storage and MDX Outline Formulas
- MDX Function Return Values
- MDX Function List

Overview of MDX

MDX is a language-based way to analyze data in Essbase cubes. MDX exhibits all of the following characteristics:

- Provides advanced data extraction capability
- Provides advanced reporting capability
- Includes functions for identifying and manipulating very specific subsets of data
- Is a data-manipulation language, complementing MaxL DDL (the data-definition language for Essbase)
- Utilizes the platform-independent XML for Analysis specification

MDX is a joint specification of the XMLA Council, who are the XML for Analysis founding members.

MDX is a language for anyone who needs to develop scripts or applications to query and report against data and metadata in Essbase databases. The following prerequisite knowledge is assumed:

- A working knowledge of the operating system your server uses and the ones your clients use.
- An understanding of Essbase concepts and features.
- Familiarity with XML.

In order for Essbase to receive MDX statements, you must pass the statements to Essbase. To pass statements, use the Analyze view in the Web interface, or use the MaxL Shell (essmsh). When using the MaxL Shell, terminate all statements with a semicolon. Results are returned in the form of a grid.
MDX Query Format

Every query using the SELECT statement has the following basic format. Items in [brackets] are optional.

\[
\begin{align*}
\text{[<with_section>]}
\text{[<insert_clause>]
\text{[<export_clause>]}}
\text{SELECT [<axis_specification>}
\text{[, <axis_specification>...]]}
\text{<subselect> | FROM <cube_specification>}
\text{[WHERE [<slicer_specification>]]}
\end{align*}
\]

Table 6-1 Description of MDX Query Elements

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;with_section&gt;</td>
<td>An optional section, beginning with the keyword WITH, in which you can define referenceable sets or members.</td>
</tr>
<tr>
<td>&lt;insert Clause&gt;</td>
<td>An optional clause for inserting tuples of data from a source to a target.</td>
</tr>
<tr>
<td>&lt;export Clause&gt;</td>
<td>An optional clause to save query results to a file on Essbase. This is an alternative to viewing the query output on a client.</td>
</tr>
<tr>
<td>SELECT</td>
<td>A literal keyword that must precede axis specifications.</td>
</tr>
<tr>
<td>[&lt;axis_specification&gt; [,&lt;axis_specification&gt;...]]</td>
<td>Any number of comma-separated axis specifications. Axes represent an ( n ) dimensional cube schema. Each axis is conceptually a framework for retrieving a data set; for example, one axis could be thought of as a column, and the next could be considered a row. See MDX Axis Specifications for more information.</td>
</tr>
<tr>
<td>&lt;subselect&gt;</td>
<td>An optional sub selection to filter an axis specification. See MDX Sub Select.</td>
</tr>
<tr>
<td>FROM</td>
<td>A literal keyword that must precede the cube specification.</td>
</tr>
<tr>
<td>&lt;cube_specification&gt;</td>
<td>The name of the database from which to select.</td>
</tr>
<tr>
<td>WHERE</td>
<td>A literal keyword that must precede the slicer specification, if one is used.</td>
</tr>
</tbody>
</table>
Table 6-1  (Cont.) Description of MDX Query Elements

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;slicer specification&gt;</td>
<td>A tuple, member, or set representing any further level of filtering you want done on the results. For example, you may want the entire query to apply only to Actual Sales in the Sample Basic database, excluding budgeted sales. The WHERE clause might look like the following: WHERE ([Scenario].[Actual], [Measures].[Sales])</td>
</tr>
</tbody>
</table>

MDX Syntax and Grammar Rules

The following topics describe syntax and grammar rules for MDX functions:

- Understanding BNF Notation
- MDX Grammar Rules
- MDX Syntax for Specifying Duplicate Member Names and Aliases
- MDX Axis Specifications
- MDX Slicer Specification
- MDX Cube Specification
- MDX Set Specification
- MDX With Section
- MDX Dimension Specification
- MDX Layer Specification
- MDX Member Specification
- MDX Hierarchy Specification
- MDX Tuple Specification
- MDX Create Set / Delete Set
- MDX Sub Select
- MDX Insert Specification
- MDX Export Specification

Understanding BNF Notation

This section briefly explains the meaning of symbolic notations used to describe grammar in this document. The query grammar rules are presented using Backus-Naur Form (BNF) syntax notation.

The following table of conventions is not a complete description of BNF, but it can help you read the grammar rules presented in this document.
Table 6-2  BNF Notation Elements

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;word&gt;</td>
<td>The word presented in angle brackets is not meant to be literally used in a statement; its rules are further defined elsewhere.</td>
<td>When reading the following syntax, SELECT &lt;axis-specification&gt; ... you know that axis-specification is not meant to be typed literally into the statement. The rules for axis-specification are further defined in the documentation (look for &lt;axis-specification&gt; ::= to get the definition).</td>
</tr>
<tr>
<td>&lt;word&gt; ::=</td>
<td>A definition, or BNF &quot;production.&quot; The symbol ::= can be interpreted to mean &quot;is defined as.&quot; The word referred to elsewhere as the placeholder &lt;word&gt; is defined here, directly following &lt;word&gt; ::=</td>
<td>The following syntax tells you that a tuple is defined as either one member in parenthesis, or two or more comma-separated members in parenthesis. &lt;tuple&gt; ::= '(' &lt;member&gt; [,&lt;member&gt;...] ')'</td>
</tr>
<tr>
<td></td>
<td>The pipe symbol or &quot;OR&quot; symbol. Precedes alternatives. The symbol can be interpreted to mean &quot;or.&quot;</td>
<td>The following syntax: ON COLUMNS</td>
</tr>
<tr>
<td>WORD</td>
<td>A query-grammar keyword, to be typed literally.</td>
<td>When reading the following syntax, SELECT &lt;axis-specification&gt; ... you know that SELECT is a keyword, and therefore should be typed literally into its proper location in the statement.</td>
</tr>
</tbody>
</table>
Table 6-2  (Cont.) BNF Notation Elements

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>[&lt;word&gt;] or [word] or [WORD] )</td>
<td>An optional element.</td>
<td>In the following high-level query syntax,</td>
</tr>
<tr>
<td>(Square brackets enclosing some word or item.)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>[&lt;with_section&gt;]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SELECT [&lt;axis_specification&gt; ]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>, [&lt;axis_specification&gt;...]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FROM [&lt;cube_specification&gt;]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[WHERE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[slicer_specification&gt;]]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>everything, technically, is optional except for SELECT and FROM.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Therefore, a query containing only the words</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SELECT FROM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>would in fact be valid; however, it would select one consolidated data</td>
</tr>
<tr>
<td></td>
<td></td>
<td>value from its best estimate of a cube context, which might not be</td>
</tr>
<tr>
<td></td>
<td></td>
<td>very useful.</td>
</tr>
<tr>
<td>[, &lt;word&gt;...] )</td>
<td>You can optionally append a comma-</td>
<td>The following syntax</td>
</tr>
<tr>
<td>(A comma, a word, and an ellipsis, all enclosed in square brackets.)</td>
<td>separated list of one or more</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;words&gt;.</td>
<td></td>
</tr>
</tbody>
</table>

MDX Grammar Rules

The following is a comprehensive view of the syntax for MDX in Essbase.

In this document, the syntax for MDX is illustrated using BNF notation.

[<with_section>]
[<insert_specification>]
[<export_specification>]
SELECT [<axis_specification>]
   , [axis_specification>...]]
[<subselect>]
[FROM [<cube_specification>]]
[WHERE [<slicer_specification> [<dim_props>]]

<insertSpecification> ::= 
    INSERT 
    <source_tuple> TO <target_tuple> .... 
    <source_tuple> TO <target_tuple> 
    [OFFSET <debitmember> <creditmember>] 
    [USING <load_buffer_method>] 
    INTO <cube_specification> <subselect>

<exportSpecification> ::= 
    EXPORT INTO FILE <fileName> [OVERWRITE <USING COLUMNDELMITER <delimiter_character>>]

<subselect> ::= 
    FROM SELECT [<axis_specification> 
    [, <axis_specification>...]]

<cubeSpecification> ::= 
    ['] <ident_or_string>.<ident_or_string> ['] 
    | <delim_ident>.<delim_ident>

<delim_ident> ::= 
    ['] <ident> ['] 
    | <ident_or_string>

<ident_or_string> ::= 
    ' <ident> ' 
    | <ident>

**Note:**

<ident> refers to a valid Essbase application/database name. In the cube specification, if there are two identifiers, the first one should be application name and the second one should be database name. For example, all of the following are valid identifiers:

- Sample.Basic
- [Sample.Basic]
- [Sample].[Basic]
- 'Sample'.Basic'

<axisSpecification> ::= 
    [NON EMPTY] <set> [<dim_props>] ON 
    COLUMNS | ROWS | PAGES | CHAPTERS | SECTIONS | AXIS (<unsigned_integer>)

<dim_props> ::= 

<member> ::=<br>  <member-name-specification><br>  |<member_value_expression><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br>
A hierarchy refers to a root member of an alternate hierarchy, which is always at generation 2 of a dimension. Member value expressions are not allowed as hierarchy arguments.
A layer name can be specified in the following ways:

1. By specifying the generation or level names; for example, States or Regions. The generation or level name can be within brackets; for example, [Regions]. Using brackets is recommended.

2. By specifying the dimension name along with the generation or level name; for example, Market.Regions and [Market].[States] This naming convention is recommended.

A tuple is a collection of member(s) with the restriction that no two members can be from the same dimension. For example, (Actual, Sales) is a tuple. (Actual, Budget) is not a tuple, as both members are from the same dimension.

For example, 
[(Actual, Sales), (Budget, COGS)] is a set.

{(Actual, Sales), (COGS, [100])} is not a set because the second tuple has members from Scenario and Product dimensions, whereas the first tuple has members from Scenario and Measures dimensions.

{(Actual, Sales), (COGS, Budget)} is not a set because the second tuple has members from Scenario and Measures dimensions, whereas the first tuple has members from Measures and Scenario dimensions (the order of dimensions is different).
Note:
The size of an input set to a function has range between 0 and 4294967295 tuples.

\[
<\text{set}> ::= \\
\quad \text{MemberRange} ( <\text{member}>, <\text{member}> \[,<\text{layertype}>]\[,<\text{hierarchy}>]\) \\
\quad | <\text{member}> : <\text{member}> \\
\quad | \{ <\text{tuple}|<\text{set}> \[, <\text{tuple}|<\text{set}>]\.. \} \\
\quad | ( <\text{set}> ) \\
\quad | <\text{set\_value\_expression}>
\]

\[
<\text{set\_value\_expression}> ::= \\
\quad | \text{Members} ( <\text{dim\_hier}> ) \\
\quad | \text{Members} ( <\text{layer}> ) \\
\quad | \text{Children} ( <\text{member}>) \\
\quad | \text{CrossJoin} ( <\text{set}> , <\text{set}> ) \\
\quad | \text{CrossJoinAttribute} ( <\text{set}> , <\text{set}> ) \\
\quad | \text{Union} ( <\text{set}>, <\text{set} > [,\text{ALL}] ) \\
\quad | \text{Intersect} ( <\text{set}>, <\text{set} > [,\text{ALL}]) \\
\quad | \text{Except} ( <\text{set}>, <\text{set} > [,\text{ALL}]) \\
\quad | \text{Extract} ( <\text{set}>, <\text{dim\_hier}> [, <\text{dim\_hier}>].. ) \\
\quad | \text{Head} ( <\text{set}> [, <\text{index}>]) \\
\quad | \text{Subset} ( <\text{set}> , <\text{index}> [,\text{index}]) \\
\quad | \text{Distict} ( <\text{set} >) \\
\quad | \text{Siblings} ( <\text{member} > [, <\text{selection\_flags}>], [\text{INCLUDEMEMBER}|\text{EXCLUCMEMBER}]) \\
\quad | <\text{member}>.\text{Siblings} \\
\quad | \text{Descendants} ( <\text{member}>, [([<\text{layer}|<\text{index}>][, <\text{Desc\_flags}>])] ) \\
\quad | \text{PeriodsToDate} ( [([<\text{layer}>[, <\text{member} >[, <\text{hierarchy}>]]]) \\
\quad | \text{LastPeriods} ( [<\text{index}>, <\text{member} >[,<\text{hierarchy}>]] ) \\
\quad | \text{xTD} ( [][\text{member}])] \\
\quad | \text{Hierarchize} ( <\text{set}> [,\text{POST}]) \\
\quad | \text{Filter} ( <\text{set}>, <\text{search\_condition}> ) \\
\quad | \text{Order} ( <\text{set}>, <\text{value\_expression}> [,\text{BASC }| \text{BDESC}]) \\
\quad | \text{TopCount} ( <\text{set}>, <\text{index}> [,<\text{numeric\_value\_expression}>] ) \\
\quad | \text{BottomCount} ( <\text{set}>, <\text{index}> [,<\text{numeric\_value\_expression}>] ) \\
\quad | \text{TopSum} ( <\text{set}>, <\text{numeric\_value\_expression}> \\
\quad | \text{BottomSum} ( <\text{set}>, <\text{numeric\_value\_expression}> \\
\quad | \text{TopPercent} ( <\text{set}>, <\text{percentage}>, <\text{numeric\_value\_expression}> ) \\
\quad | \text{BottomPercent} ( <\text{set}>, <\text{percentage}>, <\text{numeric\_value\_expression}> ) \\
\quad | \text{Generate} ( <\text{set}>, <\text{set} > [, [\text{ALL}]]) \\
\quad | \text{DrilldownMember} ( <\text{set}>, <\text{set} >[, \text{RECURSIVE}]) \\
\quad | \text{DrilupMember} ( <\text{set}>, <\text{set} >)
\]
| DrilldownByLayer ( <set> [, [<layer>|<index>]] ) |
| DrilldownLevel ( <set> [, [<layer>|<index>]] ) |
| DrillupByLayer ( <set> [, <layer>] ) |
| DrillupLevel ( <set> [, <layer>] ) |
| WithAttr ( <member> , <character_string_literal>, <value_expression> ) |
| WithAttrEx ( <member> , <character_string_literal>, <value_expression>, ANY, <tuple> | <member> [, <tuple> | <member>] ) |
| Attribute ( <member> ) |
| AttributeEx ( <member> , ANY, <tuple> | <member> [, <tuple> | <member>] ) |
| Uda ( <dimension> | <member> , <string_value_expression> ) |
| RelMemberRange ( <member> , <prevcount> , <nextcount> , [, <layertype> ] [, <hierarchy>] ) |
| Ancestors ( <member> , <layer> | <index> ) |
| <conditional_expression> |

**Note:**

<conditional_expression> is expected to return a <set> in the above production.

<Desc_flags> ::= SELF |
| AFTER |
| BEFORE |
| BEFORE_AND_AFTER |
| SELF_AND_AFTER |
| SELF_AND_BEFORE |
| SELF_BEFORE_AFTER |
| LEAVES |

<selection_flags> ::= LEFT |
| RIGHT |
| ALL |

<value_expression> ::= <numeric_value_expression> |
| <string_value_expression> |

<numeric_value_expression> ::= <term> |
| <numeric_value_expression> + <term> |
| <numeric_value_expression> - <term> |

<term> ::= <factor> |
| <term> * <factor> |
| <term> / <factor> |

<factor> ::=
[+ | -]<numeric_primary>

<numeric_primary> ::= 
   <value_expr_primary> 
   | <numeric_value_function> 
   | <mathematical_function> 
   | <date_function>

Note:
The data type of <value_expr_primary> in the above production must be numeric.

<base> ::= 
   <numeric_value_expression>
<power> ::= 
   <numeric_value_expression>

<mathematical_function> ::= 
   Abs ( <numeric_value_expression> )
   Exp ( <numeric_value_expression> )
   Factorial ( <index> )
   Int ( <numeric_value_expression> )
   Ln ( <numeric_value_expression> )
   Log ( <numeric_value_expression> [, <base>] )
   Log10 ( <numeric_value_expression> )
   Mod ( <numeric_value_expression> , <numeric_value_expression> )
   Power ( <numeric_value_expression> , <power> )
   Remainder ( <numeric_value_expression> )
   Round ( <numeric_value_expression> , <index> )
   Truncate ( <numeric_value_expression> )

<date_function> ::= 
   DateRoll(<date>, <date_part>, <index>)
   DateDiff(<date>, <date>, <date_part>)
   DatePart(<date>, <date_part>)
   Today() 
   TodateEx(<date_format_string>, <string>)
   GetFirstDate (<member>)
   GetLastDate (<member>)
   UnixDate(<numeric_value_expression>)
   GetFirstDay(<date>, <date_part>)
   GetLastDay(<date>, <date_part>)
   GetNextDay(<date>, <week-day-specification>, [0|1] )
   GetRoundDate(<date>, <date_part>)
The `<date>` argument is a number representing the input date. The number is the number of seconds elapsed since midnight, January 1, 1970. To retrieve this number, use any of the following functions: `Today()`, `TodateEx()`, `GetFirstDate()`, `GetLastDate()`.

```
<date_part> ::=  
    DP_YEAR  
    | DP_QUARTER  
    | DP_MONTH  
    | DP_WEEK  
    | DP_DAY  
    | DP_DAYOFYEAR  
    | DP_WEEKDAY  
```

**Note:**

`DP_DAYOFYEAR` and `DP_WEEKDAY` are not valid arguments in functions `DateRoll` and `DateDiff`.

```
<week-day-specification> ::=  
    1 | 2 | 3 | 4 | 5 | 6 | 7  
    e.g. 1 implying Sunday, 7 implying Saturday  
```

```
<date_format_string> ::=  
    "mon dd yyyy"  
    | "Month dd yyyy"  
    | "mm/dd/yy"  
    | "mm/dd/yyyy"  
    | "yy.mm.dd"  
    | "dd/mm/yy"  
    | "dd.mm.yy"  
    | "dd-mm-yy"  
    | "dd Month yy"  
    | "dd mon yy"  
    | "Month dd, yy"  
    | "mon dd, yy"  
    | "mm-dd-yy"  
    | "yy/mm/dd"  
    | "yymmd"  
    | "dd Month yyyy"  
    | "dd mon yyyy"  
    | "yyyy-mm-dd"  
    | "yyyy/mm/dd"  
    | "Long format"  
    | "Short format"  
```

```
<string_value_expression> ::=  
    <string_value_primary>  
```
FormatDate (<date>, <date_format_string>)
Concat (<string_value_expression> [, <string_value_expression> ...])
Left(<string_value_expression>, <length>)
Right(<string_value_expression>, <length>)
Substring (<string_value_expression>, <index> [, <index>])
Upper(<string_value_expression>)
Lower (<string_value_expression>)
RTrim(<string_value_expression>)
LTrim(<string_value_expression>)
NumToStr(<value_expr_primary>)
EnumText(<textlistname> | <member>, <numeric_value_expression>)

<value_expr_primary> ::=  
  <unsigned_numeric_literal>  
  ( <numeric_value_expression> )  
  <tuple>[.RealValue]  
  <member>[.RealValue]  
  <tuple> [.Value]  
  <member>[.Value]  
  CellValue()  
  <property>  
  <conditional_expression>  
  MISSING

<string_value_primary> ::=  
  <character_string_literal>  
  | <string_property>

Notes
• <conditional_expression> is expected to return a numeric value in the above production.
• String literals are delimited by double quotes(").
<simple_when_clause> ::=  
  WHEN <when_operand>  
  THEN <result>  

<else_clause> ::=  
  ELSE <value_expression> | <set>  

<case_operand> ::=  
  <value_expression>  

<when_operand> ::=  
  <value_expression>  

<result> ::=  
  <value_expression> | <set>  

<searched_case> ::=  
  Case  
  <searched_when_clause>...  
  [ <else_clause> ]  
  END  

<searched_when_clause> ::=  
  WHEN <search_condition>  
  THEN <result>  

<numeric_value_function> ::=  
  Avg ( <set> [, <numeric_value_expression>] [, IncludeEmpty] )  
  | Max ( <set> [, <numeric_value_expression>] )  
  | Min ( <set> [, <numeric_value_expression>] )  
  | Sum ( <set> [, <numeric_value_expression>] )  
  | NonEmptyCount ( <set> [, <numeric_value_expression>] )  
  | Count ( <set> [, IncludeEmpty] )  
  | <dts-specification> ::= DTS (<dts-operation-specification>,<member>)  

<dts-operation-specification> ::= HTD|YTD|STD|PTD|QTD|MTD|WTD|DTD  
  | Todate ( <string_value_expression> , <string_value_expression> )  
  | Ordinal ( <layer> )  
  | Aggregate ( <set> [, <member-name-specification>] )  
  | Rank ( <member_or_tuple>, <set> [, <numeric_value_expression>  
  |   [, rank_flags]] ] )  
  | NTile ( <member_or_tuple>, <set>, <index>,  
  |   <numeric_value_expression> )  
  | Percentile ( <set>, <numeric_value_expression>,  
  |   <numeric_value_expression> )  
  | Median ( <set>, <numeric_value_expression> )  
  | Len ( <string_value_expression> )  
  | InStr ( <index>, <string_value_expression>,  
  |   <string_value_expression>, <numeric_value_expression> )  
  | StrToNum ( <string_value_expression> )  
  | EnumValue ( <enum_string> )  
  | JulianDate ( <date> )  
  |
Note:
The `<member-name-specification>` in Aggregate function should refer to an Accounts dimension member name.

Note:
`<enum_string>` represents an enumerated string. It should be in the following format. The member should refer to a member of type text.

```plaintext
<enum_string> ::=  
  <textlist-name-specification>.<character_string_literal>  
  | <member>.<character_string_literal>  
<textlist-name-specification> ::=  
  Same as `<member_name-specification>` case 1. The text list name specification should refer to the name of a text list object. 
    e.g. AccountStatus, [AccountStatus]

<member_or_tuple> ::=  
  <member>  
  | <tuple>

<index> ::=  
  <numeric_value_expression>

Note:
The input `<index>` argument has range between -2147483647 and 2147483647.

<percentage> ::=  
  <numeric_value_expression>

<search_condition> ::=  
  <bool_term>  
  | <search_condition> OR <bool_term>

<bool_term> ::=  
  <bool_factor>  
  | <bool_term> AND <bool_factor>

<bool_factor> ::=  
  <bool_primary>  
  | NOT <bool_primary>
<bool_primary> ::=  
  <value_expression> [ = | > | < | <> | >= | <= ] <value_expression> 
  | <property> IN <member> | <character_string_literal> 
  | <property> <IsEmpty> ( <value_expression> ) 
  | ( <search_condition> ) 
  | <IsSibling> (<member>, <member> [, INCLUDEMEMBER]) 
  | <IsLeaf> (<member>) 
  | <IsGeneration> (<member>, <index>) 
  | <IsLevel> (<member>, <index>) 
  | <IsAncestor> (<member>, <member> [, INCLUDEMEMBER]) 
  | <IsChild> (<member>, <member> [, INCLUDEMEMBER]) 
  | <IsUda> (<member>, <string_value_expression>) 
  | <IsAccType> (<member>, <AcctTag>) 
  | <Is> ( <member>, <member> ) 
  | <member> IS <member> 
  | <IsValid> (<member> | <tuple> | <set> | <layer> | <property>) 
  | <IsMatch> (<string_value_expression>, <string_value_expression>, [, MATCH_CASE|IGNORE_CASE]) 
  | <Contains> (<member_or_tuple>, <set>)  

Note: 
Only properties with boolean values can be used as <bool_primary>.

<AcctTag> ::=  
  FIRST  
  | LAST  
  | AVERAGE  
  | EXPENSE  
  | TWO-PASS  

<rank_flags> ::=  
  ORDINALRANK  
  | DENSERANK  
  | PERCENTRANK  

<with_section> ::=  
  WITH <frml_spec>  

<frml_spec> ::=  
  <single_frml_spec>  
  | <frml_spec> <single_frml_spec>  

<single_frml_spec> ::=  
  <set_spec>  
  | <perspective_specification>  
  | <member_specification>  

<set_spec> ::=  
  SET <set_name> AS ' <set> '
<set_name> ::= 

The name of the set to be defined. The name cannot be same as any names/aliases of database members, generation/level names, or UDA names.

<perspective_specification> ::= 

PERSPECTIVE REALITY | <tuple> FOR <dimension-name-specification>

<member_specification> ::= 

MEMBER <member_name> AS ©
<nonempty_specification>
<numeric_value_expression> ' [, <solve_order_specification>]}

<member_name> ::= 
<dimension-name-specification>.<calculated member name>

<calculated member name> ::= 

Names used for calculated members cannot be the same as any names/aliases of database members, generation/level names, or UDA names.

<solve_order_specification> ::= 

SOLVE_ORDER = <unsigned_integer>

<property> ::= 

<member>.<property_specification>
| <dim_hier>.<property_specification>
| <property_specification>
| <property_expr_specification>

Note:

The last three alternatives in the above rule can be used only inside the DIMENSION PROPERTIES section.

Assume an axis has 2 dimensions, Product and Market. Using DIMENSION PROPERTIES Gen_number, [Product].level_number, the generation number will be present in the output for the members of both dimensions, whereas the level number will be present only for the members of the Product dimension.

Within a value expression, [Product].Gen_number refers to the generation number of the member named [Product].

[Product].CurrentMember.Gen_number refers to the generation number of the current member of the [Product] dimension.
For example,

Filter ([Product].Members, [Product].Gen_number > 1)

returns an empty set. Product.Generation is 1, so the search condition fails for each tuple of [Product].Members.

Filter ([Product].Members, [Product].CurrentMember.Gen_number > 1)

returns all members of Product dimension except the top dimension member, [Product].

<string_property> ::= <member>.<property_specification>

Note:
The above rule specifies string properties such as MEMBER_NAME, MEMBER_ALIAS.

<property_specification> ::= MEMBER_NAME
| MEMBER_ALIAS
| GEN_NUMBER
| LEVEL_NUMBER
| <dimension-name-specification>
| <uda-specification>

Note:
The <dimension-name-specification> in <property_specification> should be an attribute dimension-name specification. The attribute dimension names are treated as properties of members from their corresponding base dimensions.

<uda-specification> ::= 

The <uda-specification> specifies a User Defined Attribute(UDA). UDA properties are Boolean-valued properties. A TRUE value indicates presence of a UDA for a member. For example,

Filter (Market.Members, Market.CurrentMember.[Major Market])
returns the Market dimension members tagged with "Major Market" UDA in the outline.

<property_expr_specification> ::= PROPERTY_EXPR (  <dimension name>,  <property_name>,  <member_value_expression>,  <display_name>)

<property_name> ::= <property_specification>
<display_name> ::= <character_string_literal>

For more discussion of properties, see About MDX Properties.

The following rule describes the syntax for Essbase outline formulas in aggregate storage applications.

<formula_specification> ::= <nonempty_specification>  <numeric_value_expression>

<nonempty_specification> ::= NONEMPTYMEMBER <nonempty_member_list>  | NONEMPTYTUPLE ( <nonempty_member_list> )

<nonempty_member_list> ::= <nonempty_member_name>  | <nonempty_member_name> [ ,  <nonempty_member_list> ]

<nonempty_member_name> ::= An Essbase member name or a calculated member name (only when used in another calculated member).

Note:
The member name (or member names when multiple names are specified) in a NONEMPTYMEMBER directive should belong to the same dimension as the calculated member or formula member in which it is specified.

<signed_numeric_literal> ::=  [+|+] <unsigned_numeric_literal>

<unsigned_numeric_literal> ::=  <exact_numeric_literal>  | <approximate_numeric_literal>

<exact_numeric_literal> ::=  <unsigned_integer>[.unsigned_integer]  | <unsigned_integer>.

6-20
<unsigned_integer> ::= 
   {<digit>}...

<approximate_numeric_literal> ::= 
   <mantissa>E<exponent>

<mantissa> ::= 
   < exact_numeric_literal>

<exponent> ::= 
   [<sign>]<unsigned_integer>

<digit> ::= 
   0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

---

Note:
Numbers can also be input in scientific notation (mantissa/exponent), using the E character.

---

<character_string_literal> ::= 
   <quote>[<character_representation>...] <quote>

<character_representation> ::= 
   <nonquote_character> 
   | <quote_symbol>

<nonquote_character> ::= 
   Any character in the character set other than <quote>

<quote_symbol> ::= 
   <quote> <quote>

<quote> ::= "

The following is the syntax for Format Strings in Essbase:
MdxFormat( string_value_expression )

---

**MDX Syntax for Specifying Duplicate Member Names and Aliases**

The following member specification rules apply to databases with duplicate member names enabled.
Qualified names must be used to specify duplicate member names. Qualified member or alias names can be specified using:

- **Fully qualified member names**—Consist of duplicate member or alias name and all ancestors up to and including the dimension name. Each name must be enclosed in square brackets([]) and separated by a period.

  \[[\text{DimensionMember}].[\text{Ancestors}].[\text{DuplicateMember}]\]

  For example:

  \[[\text{Product}].[100].[100-10]\]

- **Shortcut qualified member names**—Essbase internally constructs shortcut qualified names for members in duplicate member outlines.

  You can manually insert shortcut qualified names into scripts, Smart View or other grid clients, or MDX queries.

  Essbase uses the following syntax to construct shortcut qualified names. Using the same syntax that Essbase uses when you reference members in scripts, grid clients, and MDX queries is optimal, but not required.

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Qualified Name Syntax</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Duplicate member names exist at generation 2</td>
<td>[[\text{DimensionMember}]. [\text{DuplicateMember}]]</td>
<td>[[\text{Year}].[\text{Jan}]] or [[\text{Product}].[\text{Jan}]]</td>
</tr>
<tr>
<td>Duplicate member names exist in an outline, but are unique within a dimension</td>
<td>[[\text{DimensionMember}].[\text{DuplicateMember}]]</td>
<td>[[\text{Year}@[\text{Jan}]]</td>
</tr>
<tr>
<td>Duplicate member names have a unique parent</td>
<td>[[\text{ParentMember}]. [\text{DuplicateMember}]]</td>
<td>[[\text{East}].[\text{New York}]]</td>
</tr>
<tr>
<td>Duplicate member names exist at generation 3</td>
<td>[[\text{DimensionMember}]. [\text{ParentMember}]. [\text{DuplicateMember}]]</td>
<td>[[\text{Products}].[\text{Personal Electronics}].[\text{Televisions}]]</td>
</tr>
<tr>
<td>Duplicate member names exist at a named generation or level, and the member is unique at its generation or level</td>
<td>[[\text{DimensionMember}].[\text{GenLevelName}].[\text{DuplicateMember}]]</td>
<td>[[\text{2006}@[\text{Gen1}]][\text{Jan}]]</td>
</tr>
</tbody>
</table>

In MDX, either one the following syntax methods must be used to reference shortcut qualified member names:
– **Escape Character method**—Because MDX syntax also uses square brackets:
1. Any internal closing bracket ( ] ) used by name parts within the shortcut qualified names requires an additional ] escape character.
2. The entire shortcut qualified member name must be enclosed in a set of square brackets ([]).

Examples:

- [Year].[Jan] is referenced as [[Year].[Jan]] in MDX.
- [Year]@[Jan] is referenced as [[Year]@[Jan]] in MDX.
- [2006]@[Gen1]@[Jan] is referenced as [[2006]@[Gen1]@[Jan]] in MDX.

**Note:**
The above syntax also works for fully qualified member names, but is not required.

– **StrToMbr Function method**—You can use the StrToMbr function to convert qualified name strings to member value expressions.

Examples:

- [Year].[Jan] is referenced as StrToMbr("[Year].[Jan]") in MDX.
- [Year]@[Jan] is referenced as StrToMbr("[Year]@[Jan]") in MDX.
- [2006]@[Gen1]@[Jan] is referenced as StrToMbr("[2006]@[Gen1]@[Jan]") in MDX.

**Note:**
The above syntax also works for fully qualified member names, but is not required.

**Duplicate Member Names Query Example**

The following query uses both methods of referencing shortcut member names in MDX:

```
SELECT  
  { Sales, Profit }  
ON COLUMNS,  
  {{[Store]}@[6]}, StrToMbr("Product.Sku.1")}  
ON ROWS  
FROM MySample.Basic  
WHERE {{[1998].[Q1].[1]}}
```
Note:
StrToMbr accepts any type of member-identifier strings: names, aliases or qualified names.

Shared Member Names Example
The following example applies to a unique member name outline that contains shared members.

In the Sample Basic database, the member [100-20] is the referenced member under parent [100], and has a shared member associated with it under parent [Diet]. The shared member [100-20] can be referred to explicitly, using the unique name [Diet]. [100-20], as shown in the following query:

```sql
SELECT
  {Sales}
ON COLUMNS,
  {{[Diet].[100-20]}} PROPERTIES MEMBER_UNIQUE_NAME
ON ROWS
FROM Sample.Basic;
```

MDX Axis Specifications
An axis specification consists of a set and one or more axis keywords.

```latex
<axis_specification> ::= 
  [NON EMPTY] <set> ON COLUMNS|ROWS|PAGES|CHAPTERS|SECTIONS|
  AXIS(<unsigned_integer>)
```

Understanding the following concepts will help you construct axis specifications for many SELECT queries

Ordering of Axes
If providing multiple axes, you cannot skip axes. For example, you can specify a Row axis only if you have a Column axis. You can specify a Pages axis only if you also have Column and Row axes.

You can also use ordinals to represent the axes. For example, you can specify <set> ON AXIS(0), <set> ON AXIS(1), etc.

You can specify up to 64 axes (though it is common to use just two). The first five ordinal axes have keyword aliases:

<table>
<thead>
<tr>
<th>Axis Keyword</th>
<th>Axis Ordinal</th>
</tr>
</thead>
<tbody>
<tr>
<td>COLUMNS</td>
<td>AXIS(0) (default if nothing specified)</td>
</tr>
<tr>
<td>ROWS</td>
<td>AXIS(1)</td>
</tr>
</tbody>
</table>
Table 6-4  (Cont.) Axis Keywords and Corresponding Ordinal Notation

<table>
<thead>
<tr>
<th>Axis Keyword</th>
<th>Axis Ordinal</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAGES</td>
<td>AXIS(2)</td>
</tr>
<tr>
<td>CHAPTERS</td>
<td>AXIS(3)</td>
</tr>
<tr>
<td>SECTIONS</td>
<td>AXIS(4)</td>
</tr>
</tbody>
</table>

For example:

```
SELECT set1 ON COLUMNS,
set2 ON ROWS
FROM Sample.Basic
```

is the same as:

```
SELECT set1 ON AXIS(0),
set2 ON AXIS(1)
FROM Sample.Basic
```

Both return a hypothetical data cube (or subset) of the following format:

Table 6-5  Hypothetical Subset of Data

<table>
<thead>
<tr>
<th>(axis)</th>
<th>Member names in set1</th>
<th>Member names in set2</th>
<th>Data at intersections of set1 and set2 members</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The examples above are hypothetical because they will not return a cube until values are provided for the sets. In the following example, we replace set1 and set2 with real sets:

```
SELECT
{[100-10], [100-20]} ON COLUMNS,
{[Qtr1], [Qtr2], [Qtr3], [Qtr4]} ON ROWS
FROM Sample.Basic
```

which returns the following results:

Table 6-6  Output Grid from MDX Example

<table>
<thead>
<tr>
<th>(axis)</th>
<th>100-10</th>
<th>100-20</th>
</tr>
</thead>
<tbody>
<tr>
<td>Qtr1</td>
<td>5096</td>
<td>1359</td>
</tr>
<tr>
<td>Qtr2</td>
<td>5892</td>
<td>1534</td>
</tr>
<tr>
<td>Qtr3</td>
<td>6583</td>
<td>1528</td>
</tr>
<tr>
<td>Qtr4</td>
<td>5206</td>
<td>1287</td>
</tr>
</tbody>
</table>
Specifying the Set

You can represent the sets in each axis in many ways.

```mdx
SELECT
{} ON COLUMNS
from sample.basic
```

illustrates that you can choose nothing for a set. However, no cell values will be returned. The following rules apply:

- When any of the axes contains an empty set, no cell values are returned. The axes whose sets have at least one tuple will have their tuples returned.
- If there are no axes at all, then exactly one cell is returned using the default member of each dimension. The slicer tuple, if present, overrides the default member for the respective dimensions.

```mdx
SELECT
{ ( [Year].[Qtr2] ) } ON COLUMNS
from sample.basic
```

illustrates using a set that contains a single tuple.

For more information about sets, see MDX Set Specification.

**NON EMPTY**

The axis specification syntax including NON EMPTY is shown below:

```mdx
<axis_specification> ::= 
[NON EMPTY] <set> ON 
  COLUMNS | ROWS | PAGES | CHAPTERS |
  SECTIONS | AXIS (<unsigned_integer>)
```

Including the optional keywords NON EMPTY before the set specification in an axis causes suppression of slices in that axis that would contain entirely #MISSING values.

For any given tuple on an axis (such as (Qtr1, Actual)), a slice consists of the cells arising from combining this tuple with all tuples of all other axes. If all of these cell values are #MISSING, the NON EMPTY keyword causes the tuple to be eliminated.

For example, if even one value in a row is not empty, the entire row is returned. Including NON EMPTY at the beginning of the row axis specification would eliminate the following row slice from the set returned by a query:

<table>
<thead>
<tr>
<th>Qtr1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actual</td>
</tr>
</tbody>
</table>

For another example, see the Tail function.
Dimension Properties
A property, in MDX grammar, refers to the Essbase concepts of attributes and UDAs.
The axis specification syntax including the properties specification is shown below:

\[
\text{<axis\_specification> ::= [\text{NON EMPTY}] <set> [<\text{dim_props}>] ON COLUMNS | ROWS | PAGES | CHAPTERS | SECTIONS | AXIS (<unsigned\_integer>)}
\]

As shown in the above syntax, a properties specification can follow the set specification in an axis.

For more information about properties, see About MDX Properties.

MDX Slicer Specification
This section shows rules for the slicer specification (WHERE clause). The slicer axis is a way of limiting a query to apply only to a specific area of the database.

A slicer specification consists of the WHERE keyword followed by a tuple, member, or set. You can optionally query for certain dimension properties in the slicer specification.

Syntax

\[
[\text{WHERE} [<\text{slice\_specification}> [<\text{dim_props}>]]]
\]

\[
<\text{slice\_specification}> ::= <\text{set}> | <\text{tuple}> | <\text{member}>
\]

Note:
The cardinality of the <set> in the slicer should be 1; in other words, if a set is used, it must evaluate to a single tuple.
Note:
The same dimension cannot appear on an axis and the slicer. To filter an axis using criteria from its own dimension, you can use a sub select. See MDX Sub Select.

<dim_props> ::=  
  [DIMENSION] PROPERTIES <property> [, <property>...]  

Example
For example, you may want an entire query to apply only to Actual Sales in the Sample Basic database, excluding budgeted sales or any other measures. The WHERE clause might look like the following:

SELECT  
  {([West].children)}  
ON COLUMNS,  
  {([Diet].children)}  
ON ROWS  
FROM Sample.Basic  
WHERE ([Scenario].[Actual], [Measures].[Sales])

MDX Cube Specification
Use the cube specification to name the database at which the query is directed. A cube specification consists of the FROM keyword followed by delimited or nondelimited identifiers indicating an application name and a database name.

The first identifier should be an application name and the second one should be a database name. For example, all of the following are valid identifiers:

- Sample.Basic
- [Sample.Basic]
- [Sample].[Basic]
- 'Sample'.Basic'

Syntax

[FROM [<cube_specification>]]

<cube_specification> ::=  
  '['<ident_or_string>.<ident_or_string>']'
  |<delim_ident>.<delim_ident>

<delim_ident> ::=  
  '['<ident>']'
  |<ident_or_string>

<ident_or_string> ::=  
  ...
Notes

If [FROM [cube_specification]] is omitted from a query, the current database context is assumed.

Example

Sample.Basic is the cube specification in the following hypothetical query.

```
SELECT ...
FROM Sample.Basic
```

MDX Set Specification

A set is a collection of tuples. In each tuple of the set, members must represent the same dimensions as do the members of other tuples of the set. Additionally, the dimensions must be represented in the same order.

```
<set> ::= 
  MemberRange ( <member>, <member> )
  | <member> : <member>
  | { [tuple] | <set> } [, <tuple> | <set>.. }
  | <set_value_expression>
```

**Table 6-8  Ways To Specify an MDX Set**

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MemberRange (&lt;member&gt;, &lt;member&gt;)</td>
<td>A set can be a range of members, specified using the MemberRange function.</td>
</tr>
<tr>
<td>&lt;member&gt; : &lt;member&gt;</td>
<td>Alternate syntax that has the same effect as the MemberRange function.</td>
</tr>
<tr>
<td>{ [tuple]</td>
<td>&lt;set&gt; } [, &lt;tuple&gt;</td>
</tr>
<tr>
<td>&lt;set_value_expression&gt;</td>
<td>Output from any function that returns a set. As an alternative to creating sets member-by-member or tuple-by-tuple, you can use a function that returns a set. For a list of functions that return sets, see MDX Function Return Values.</td>
</tr>
</tbody>
</table>

MDX With Section

The WITH section is for defining referential sets or members that can be used multiple times during the life of a query.
Beginning with the keyword `WITH` at the very start of a query, you can define a buffer of reusable logic lasting for the length of the query execution. This can save time in lines of code written as well as in execution time.

If varying attributes are enabled, the `WITH` section can also be used to define perspective for each varying attribute dimension. In case of multiple varying attributes, perspective setting can be defined for each varying attribute dimension separately.

In the `WITH` section, you can create the following reusable elements:

- Calculated members
- Named Sets

**Syntax**

```mdx
WITH

    SET set_name AS ' set ' |
    MEMBER calculated_member_name AS ' <numeric_value_expr> ' |
    [, <solve_order_specification> ] |
    <perspective_specification>
```

**Table 6-9  MDX WITH Section Elements**

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>set_name</code></td>
<td>The name of the set that will be defined after the <code>AS</code> keyword. Any name can be used; it should be something that helps you remember the nature of the set. For example, a set name could be <code>Best5Books</code>, which names a set of the five top-selling paperback titles in December:</td>
</tr>
</tbody>
</table>

```
WITH
    SET [Best5Books] AS
    'Topcount ( [Paperbacks].members, 5, ([Measures].[Sales], [Scenario].[Actual], [Year].[Dec]) )'
```

| `set`     | The logic of a `set specification`; this can be re-used because it is being named. Must be enclosed in single quotation marks. In the example above, the Topcount function defines the entire set. |

```mdx
WITH
    SET [Best5Books] AS 'Topcount ( [Paperbacks].members, 5, ([Measures].[Sales], [Scenario].[Actual], [Year].[Dec]) )'
```
Table 6-9  (Cont.) MDX WITH Section Elements

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
</table>
| calculated_member_name    | A name for a hypothetical member existing for the duration of query execution. In its definition, you must associate the calculated member with a dimension (as [Max Qtr2 Sales] is associated with the Measures dimension, in the example that follows). For example, the calculated member named Max Qtr2 Sales has its value calculated at execution time using the Max function: WITH MEMBER [Measures].[Max Qtr2 Sales] AS 'Max (
  {[Year].[Qtr2]},
  [Measures].[Sales]
)' Calculated members do not work with metadata functions such as Children, Descendants, Parent, and Siblings. For example, if there is a calculated member defined as [CM1], you cannot use it in the following way: [CM1].children. |
| <numeric_value_expr>      | An expression involving real members in the database outline, compared using mathematical functions. The value resulting from the expression is applied to the calculated member. By using calculated members, you can create and analyze a great many scenarios without the need to modify the database outline. |
Table 6-9  (Cont.) MDX WITH Section Elements

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;solve_order_specification&gt;</code></td>
<td>Optional. By adding <code>SOLVE_ORDER = n</code> to the end of each calculated member, you can specify the order in which the members are calculated. For example, solve order in the following hypothetical query is indicated in bold:</td>
</tr>
<tr>
<td></td>
<td>WITH</td>
</tr>
<tr>
<td></td>
<td>MEMBER [Product].[mbr1] AS 'calculation', <code>SOLVE_ORDER = 2</code></td>
</tr>
<tr>
<td></td>
<td>MEMBER [Product].[mbr2] AS 'calculation', <code>SOLVE_ORDER = 1</code></td>
</tr>
<tr>
<td></td>
<td>SELECT</td>
</tr>
<tr>
<td></td>
<td><code>{{[Year].children}}</code> on columns,</td>
</tr>
<tr>
<td></td>
<td><code>{{[Product].[mbr1], [Product].[mbr2]}}</code> on rows</td>
</tr>
<tr>
<td></td>
<td>See Usage Examples for Solve Order.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><code>&lt;perspective_specification&gt;</code></th>
<th>PERSPECTIVE REALITY</th>
<th>tuple FOR dimension</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>When a database uses varying attributes, base members associated with the varying attributes are aggregated according to the specified perspective.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>You can set the perspective to reality (using the REALITY keyword) or to explicit (using an input tuple consisting of level 0 members).</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Reality-based evaluation and reporting is the default, in which independent members are determined by the current context.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>When using explicit evaluation and reporting, you specify a tuple of level 0 members from the independent dimension to be used as the context.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>For an example of a reality-based perspective, see the example for AttributeEx. For an example of an explicit perspective, see the example for WithAttrEx.</td>
<td></td>
</tr>
</tbody>
</table>
**Usage Examples for Solve Order**

```mdx
WITH
MEMBER
[Measures].[Profit Percent]
AS 'Profit *100 /Sales', SOLVE_ORDER=20
MEMBER
[Year].[FirstFourMonths]
AS 'Sum(Jan:Apr)', SOLVE_ORDER=10
SELECT
{{[Profit], [Sales], [Profit Percent]}}
ON COLUMNS,
{{[Jan], [Feb], [Mar], [Apr], [FirstFourMonths]}}
ON ROWS
FROM Sample.Basic
```

The calculated member `[Profit Percent]`, defined in the Measures dimension, calculates Profit as a percentage of Sales.

The calculated member `[FirstFourMonths]`, defined in the Year dimension, calculates sum of data for first four months.

When data for ([Profit Percent], [FirstFourMonths]) is evaluated, SOLVE_ORDER specifies the order of evaluation, ensuring that [Profit Percent] is evaluated first, and resulting in a correct value for percentage. If you change the order of evaluation, you will see that the percentage value is not correct. In this example, SOLVE_ORDER specifies that sum should be calculated before percentage.

**Tie-Case Example for Solve Order**

When evaluating a cell identified by multiple calculated members, the SOLVE_ORDER value is used to determine the order in which the expressions are evaluated. The expression that is used to evaluate the cell is that of the calculated member with the highest SOLVE_ORDER value. In this case, [Profit Percent]'s expression is used to evaluate ([Profit Percent], [FirstFourMonths]). The example above is calculated as:

```mdx
([Profit Percent], [FirstFourMonths])
= ([Profit], [FirstFourMonths]) * 100 / ([Sales], [FirstFourMonths])
= ([Profit], [Jan]) + ([Profit], [Feb]) + ([Profit], [Mar]) + ([Profit], [Apr]) * 100 /
  ([Sales], [Jan]) + ([Sales], [Feb]) + ([Sales], [Mar]) + ([Sales], [Apr])
```

A tie situation is possible because calculated members may have the same SOLVE_ORDER value. The tie is broken based on the position of the dimensions to which the calculated members are attached:

- For aggregate storage outlines, the calculated member belonging to the dimension that comes later in the outline is the one that wins in this case.
- For block storage database outlines (and for pre-Release 7.1.2 aggregate storage outlines), the solve order property applies to calculated members defined in an MDX query. The calculated member belonging to the dimension that comes earlier in the outline is the one that wins in this case, and its expression is used to evaluate the cell.
Calculated Members

For examples of queries using calculated members, see examples for the following functions:

- Abs
- Avg
- BottomPercent
- Case
- ClosingPeriod
- Count
- Exp
- FirstSibling
- IIF
- Int
- Lag
- LastPeriods
- Lead
- Ln
- Max
- Min
- Mod
- NextMember
- NonEmptyCount
- Ordinal
- PrevMember
- Remainder
- Sum
- Todate

Named Sets

For examples of queries using named sets, see examples for the following functions:

- BottomPercent
- CurrentTuple
- Filter (example 3)
- Generate
Parent (example 2)

Perspective

For examples of varying attribute queries using perspective, see examples for the following functions:

AttributeEx

WithAttrEx

MDX Dimension Specification

A dimension is a top-level member in the hierarchy (a member with no parent). Represent a dimension using the following rules:

Syntax

<dimension> :: =
   <dimension-name-specification>
   | <member>.DIMENSION
   | <layer>.DIMENSION
   | DIMENSION ( <member> | <layer> )

Table 6-10 Ways to Specify a Dimension in MDX

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;dimension-name-specification&gt;</td>
<td>A dimension name. See Description, item 1.</td>
</tr>
<tr>
<td>&lt;member&gt;.DIMENSION</td>
<td>Dimension function with a member specification as input.</td>
</tr>
<tr>
<td>&lt;layer&gt;.DIMENSION</td>
<td>Dimension function with a layer specification as input.</td>
</tr>
<tr>
<td>DIMENSION ( &lt;member&gt;</td>
<td>&lt;layer&gt; )</td>
</tr>
</tbody>
</table>

Description

A dimension can be represented in the following ways:

1. Using the dimension name (the name of the top member of a dimension.) For example, [Market].
2. Using the Dimension function with a member of a dimension as input. For example, [New York].Dimension Or Dimension ( [New York] ).
3. Using the Dimension function with a layer specification as input. For example, Dimension ([Market].Generations(2).Members) or Dimension ( [Market].Generations(2).Members).Dimension.

MDX Layer Specification

A layer is a shared depth in the outline hierarchy. Therefore, the concept of layer includes generations and levels. Represent a layer using the following rules:
Syntax

\[
\text{<layer>} ::= \\
\quad \text{<layer-name-specification>} \\
\quad \mid \text{Levels ( <dim_hier>, <index> )} \\
\quad \mid \text{<dim_hier>.Levels ( <index> )} \\
\quad \mid \text{Generations ( <dim_hier>, <index> )} \\
\quad \mid \text{<dim_hier>.Generations ( <index> )} \\
\quad \mid \text{<member>.Generation} \\
\quad \mid \text{<member>.Level}
\]

Table 6-11  Ways to Specify a Layer in MDX

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{&lt;layer-name-specification&gt;}</td>
<td>A layer name can be specified in the following ways:</td>
</tr>
<tr>
<td>\text{1.}</td>
<td>By specifying the generation or level names; for example, States or Regions. The generation or level name can be within brackets; for example, [Regions]. Using brackets is recommended.</td>
</tr>
<tr>
<td>\text{&lt;dimension&gt;.Levels (&lt;index&gt;)}</td>
<td>Levels function with the dimension specification and a level number as input. For example, [Year].Levels(0).</td>
</tr>
<tr>
<td>\text{Levels ( &lt;dimension&gt;, &lt;index&gt; )}</td>
<td>Alternate syntax for Levels function with the dimension specification and a level number as input. For example, Levels ( [Year], 0 ).</td>
</tr>
<tr>
<td>\text{&lt;dimension&gt;.Generations (&lt;index&gt;)}</td>
<td>Generations function with the dimension specification and a generation number as input. For example, [Year].Generations (3).</td>
</tr>
<tr>
<td>\text{Generations ( &lt;dimension&gt;, &lt;index&gt; )}</td>
<td>Alternate syntax for Generations function with the dimension specification and a generation number as input. For example, Generations ( [Year], 3 ).</td>
</tr>
<tr>
<td>\text{&lt;member&gt;.Generation}</td>
<td>Generation function with a member specification as input. For example, [Year].Generation. Returns the generation of the specified member.</td>
</tr>
<tr>
<td>\text{&lt;member&gt;.Level}</td>
<td>Level function with a member specification as input. For example, [Year].Level. Returns the level of the specified member.</td>
</tr>
</tbody>
</table>
Description

Generation numbers begin counting with 1 at the dimension name; higher generation numbers are those that are closest to leaf members in a hierarchy.

Level numbers begin with 0 at the deepest part of the hierarchy; the highest level number is a dimension name.

Note:

In an asymmetric (or ragged) hierarchy, same level numbers does not mean that the members are at the same depth in the outline. For example, in the following diagram, member aa and member f are both level 0 members, and yet they are not at the same depth:

MDX Member Specification

A member is a named hierarchical element in a database outline. Represent a member using the following rules:

Syntax

\[
\text{<member>} ::= \\
\text{<member-name-specification>} \\
\text{<member_value_expression>}
\]
Member Name Specification

A member name can be specified in the following ways:

1. **By specifying the actual name or the alias; for example, Cola, Actual, COGS, and [100].**

   If the member name starts with number or contains spaces, it should be within brackets; for example, [100]. Brackets are recommended for all member names, for clarity and code readability.

   If the member name starts with an ampersand (&), it should be within quotation marks; for example, ["&xyz"]. This is because the leading ampersand is reserved for substitution variables. You can also specify it as StrToMbr("$100").

   For attribute members, the long name (qualified to uniquely identify the member) should be used; for example, [Ounces_12] instead of [12].

2. **By specifying dimension name or any one of the ancestor member names as a prefix to the member name; for example, [Product].[100-10] and [Diet].[100-10]**. This is a recommended practice for all member names, as it eliminates ambiguity and enables you to refer accurately to shared members.

   **Note:**

   Use only one ancestor in the qualification. Essbase returns an error if multiple ancestors are included. For example, [Market].[New York] is a valid name for New York, and so is [East].[New York]. However, [Market].[East].[New York] returns an error.

3. **By specifying the name of a calculated member defined in the WITH section.**

4. **For outlines that have duplicate member names enabled, see also MDX Syntax for Specifying Duplicate Member Names and Aliases.**

Member Value Expression

A member value expression is output from any function that returns a member. As an alternative to referencing the member by name or alias, you can use a function that returns a member in place of <member>. For a list of functions that return a member, see MDX Function Return Values.

Unresolved Member Names

If an MDX query contains references to members that do not exist in the outline, the unresolved member names can be skipped so that the query can continue without error. To enable this feature, use the EssOpMdxQuery Java interface or EssMdxSetQueryOptions C API function. Unresolved names are left out from the result grid in cases where non-existing members are given on query axes or as parameters to functions.

MDX Hierarchy Specification

A hierarchy is a root member of an alternate hierarchy, which is always at generation 2 of a dimension. Member value expressions are not allowed as hierarchy arguments.
Alternate hierarchies are applicable to aggregate storage databases only.

The dimension of the hierarchy argument passed to a function must match the dimension of the other arguments passed to the function. If they do not match, an error is returned, and the query is aborted.

MDX Tuple Specification

This section shows rules for tuple specifications.

A **tuple** is a collection of member(s) with the restriction that no two members can be from the same dimension. For example, (Actual, Sales) is a tuple. (Actual, Budget) is not a tuple, as both members are from the same dimension.

**Syntax**

<tuple> ::=  
  <member>  
  | ( <member> [, <member> ].. )  
  | <tuple_value_expression>

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;member&gt;</td>
<td>A member name. If a member name contains spaces or special characters, enclose it in brackets [ ]. It is good practice to use brackets for member names, even if they do not contain special characters. Example: [West]</td>
</tr>
<tr>
<td>( &lt;member&gt; [, &lt;member&gt; ].. )</td>
<td>One or more member names, separated by commas. The members must be from different dimensions. The list of members must be enclosed in parentheses (). Example: ( [West], [Feb] )</td>
</tr>
</tbody>
</table>
| <tuple_value_expression> | An instance of a function that extracts a tuple from a set. There are two such functions available:  
  • CurrentTuple  
  • Item |

**Description**

A tuple represents a single data cell if all dimensions are represented. For example, this tuple from Sample Basic is a single data value:

( [Qtr1], [Sales], [Cola], [Florida], [Actual] )

MDX Create Set / Delete Set

This section shows how to create and delete a named set that persists for the duration of a login session.
A named set is a re-usable member selection that can help streamline the writing and execution of MDX queries.

Syntax

The syntax to create or delete session-persistent named sets is shown below:

```
CREATE SET set name AS 'set' [FROM <cube_specification>] [WHERE [<slicer_specification>]]
DROP SET set_name [FROM <cube_specification>]
```

Examples

Example 1

The following statement creates a named set called "Most Selling Products," which is a selection of the top selling products for Qtr1:

```
CREATE SET [Most Selling Products] AS 
  {TopCount
   (Descendants
    ( [Product], [Product].level, AFTER ), 3,
     ([Measures].[Sales], [Year].[Qtr1])
   )
  }
```

The following query, issued in the same login session as the CREATE statement, references the stored named set “Most Selling Products”:

```
SELECT {Measures].[Sales]
ON COLUMNS,
[Most Selling Products]
ON ROWS
FROM [Sample.Basic]
```

Example 2

To provide a context, a slicer clause maybe added to the set creation statement, as shown in bold:

```
CREATE SET [Most Selling Products] AS 
  {TopCount
   (Descendants
    ( [Product], [Product].level, AFTER ), 3,
     ([Measures].[Sales], [Year].[Qtr1])
   )
  }
```

MDX Sub Select

A sub select is a secondary SELECT statement nested within the primary SELECT statement, in a FROM clause. Its purpose is to reduce, or filter out, the volume of scanned data. Using a sub select provides an effective way of processing queries that require partial aggregations.

Syntax

The syntax for using a sub select is shown in the context of the MDX query format:

```mdx
[<with_section>]
SELECT <axis_specification>
    [, <axis_specification>...]
    <subselect>
[WHERE [<slicer_specification>]]
```

Where `<subselect>` is:

```mdx
FROM
    (SELECT <axis_specification>
        [, <axis_specification>...]
        FROM <cube_specification>)
```

Notes

The following guidelines apply to members you can use in the sub select:

- Can be from any generation or level. The consolidation operators of descendants are analyzed, for potential filtering out of results. If a descendant's operator is ~ (non consolidation) and its descendants do not have any shared members or referenced members of shared members, its sub-hierarchy is removed from results. Similarly, a stored, non-level-0 member in a block storage hierarchy is the sole contributor to the aggregation; its children are not treated as dependencies.
- Can be calculated members defined in the WITH section.
- Can be formulas. Formula contributors are analyzed, but not their descendants.
- Functions that return a value are not evaluated (see MDX Functions that Return a Number), nor are functions that derive their results using data (see Data-based Set Functions in MDX Functions that Return a Set. All dependencies from such expressions are included.
- If members are from the same dimension, they must also be in the same level and hierarchy (applies to aggregate storage databases only).
• The NON EMPTY syntax is not relevant in a sub-select axis specification.

Example

SELECT
    [Digital Cameras/Camcorders].Children ON COLUMNS
FROM
    (SELECT
        {[Digital Cameras],[Camcorders]} ON COLUMNS
    FROM ASOsamp.Sample)
WHERE ( [Curr Year],[94706],[Coupon],[Cash],[1 to 13 Years],[Under 20,000],
    [Sale],[Units],[Mar])

MDX Insert Specification

The insert clause is a way you can use MDX to update the database with new data, by inserting tuples from a source to a target.

MDX Insert is supported for aggregate storage databases and hybrid aggregation mode databases.

Syntax

[WITH MEMBER calculated_member_name AS ' <numeric_value_expr> ']
INSERT
    <source_tuple> TO <target_tuple>
    ....
    <source_tuple> TO <target_tuple>
    [<offset> <debitmember> <creditmember>]
    [USING <load_buffer_method>]
INTO
    APP.DB
FROM
    ( <nested_select_statement> )
[WHERE [ <slicer_specification> ]]
### Table 6-13 MDX INSERT Clause Elements

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
</table>
| source_tuple                 | A database region from which to retrieve data values. The source tuple can contain dynamic or stored members. It can contain member-based functions, but it cannot contain context-dependent member functions, such as CurrentMember. Examples:  
  - "([Scenario].[S1], [Jan])"  
  - "([Scenario].[S1])"  
  - "([Measures].[Payroll])"
|                              | Map the source tuple to a target tuple that you will be updating.                                                                                                                                               |
| target_tuple                 | The database region to populate with values from the source tuple. The target tuple must consist of only stored members, dynamic calc and store members, or member-based functions. It cannot contain dynamic members. Examples:  
  - "([Actual])"  
  - "([Actual], [Revised_payroll])"  
  - "([Actual], [Year].CurrentMember.PrevMember)"
|                              |                                                                                                                                                                                                             |
| offset, creditmember, debitmember | Optional parameters for double-entry accounting, applicable only for custom calculations in aggregate storage cubes. For details about these parameters, see Performing Custom Calculations and Allocations on Aggregate Storage Databases |
| USING load_buffer_method     | Optional, and supported only for aggregate storage databases. Specifies the data load buffer method to use when updating the aggregate storage database. Examples:  
  - USING Add Values  
  - USING Subtract Values  
  If no method is specified, the update replaces values with the contents of the load buffer. |
| INTO app.db                  | The cube specification naming the database at which the Insert clause is directed. Must be same as the cube used in the FROM clause of the inner SELECT statement.                                              |
| FROM nested_select_statement | An inner select statement defining the database region from which the tuples you want to insert should be retrieved.                                                                                     |

The WITH section is optional, enabling you to define the area to insert using a calculated member.
The WHERE section is optional, enabling you to define a slicer.

Notes
- Do not use attribute dimension members in the source or target tuples.
- Do not use context-dependent member functions, such as CurrentMember or PrevMember, in the source tuple.
- The source and target tuples should have the same dimensionality. For example, the following source and target tuple have the same dimensionality because the target tuple, [Scenario].[Actual], which is stored, matches the format of the source tuple, [Scenario].[S1], which is a calculated member defined in the WITH section.

"([Scenario].[S1])" TO "([Scenario].[Actual])"

- #Missing values are not inserted/copied.
- The source cube (app.db) of the INTO clause must be same as the source cube used in the FROM clause of the inner SELECT statement.

Example 6-1  Calculated Member and Nested Select Statement

The following example uses a calculated member, M1, as the source tuple to update a target member, Commission, in Sample Basic.

WITH
  Member [Measures].[M1] as 'Sales * 0.1'
INSERT
  "([Measures].[M1])" TO "([Measures].[Commission])"
INTO [Sample].[Basic]
FROM (SELECT
  {[Measures].[M1]} on columns,
  {Jan, Actual, [100-10], [New York]} on ROWS
FROM [Sample].[Basic] )

Example 6-2  Copying Data

The following example uses an inner select statement of crossjoins to copy data from one outline member to another.

INSERT "([Measures].[Payroll])" TO "([Measures].[Revised_Payroll])"
INTO [Test].[Basic]
FROM (SELECT
  {[Measures].[Payroll]} ON COLUMNS,
  (Crossjoin (Crossjoin (Descendants([Year]),
    Crossjoin (Descendants([Scenario]),
      Descendants([Product])))
    Descendants([Market])))
  ON ROWS
FROM [Test].[Basic] );
Example 6-3  Inserting Multiple Tuples

The following example inserts multiple tuples into Test.Basic.

WITH
  Member [Measures].[M2] as 'Sales * 0.5'
INSERT
  "([Measures].[M2])"
  TO
  "([Measures].[Commission])"
  "([East].[New York],[Measures].[Payroll])"
  TO
  "([Measures].[Revised_Payroll])"
INTO [Test].[Basic]
FROM (SELECT
  {[Measures].[M2]} ON COLUMNS,
  Crossjoin(Crossjoin(Descendants([Year]),
    Crossjoin(Descendants([Scenario]),
      Descendants([Product]))),
    Descendants([Market])) ON ROWS
  FROM [Test].[Basic] );

Example 6-4  Performing Allocations

The following example uses a calculated member to perform an allocation in the Scenario dimension.

WITH MEMBER
  [Scenario].[S1] AS
  '([PY Actual], [Total Expenses]) *
  ([Budget] / ([Total Expenses], [Budget]))'
INSERT
  "([Scenario].[S1])"
  TO
  "([Scenario].[PY Actual])"
INTO
  [Sample1].[Basic]
FROM
  (SELECT
    {[Scenario].[S1]} ON COLUMNS,
    Crossjoin
    (Crossjoin
      (Crossjoin
        (Descendants([Market]),
          Descendants([Product]))),
        Descendants([Year]))
    ON ROWS
  FROM [Test].[Basic] );
The above MDX example has similar functionality to a block storage allocation as shown in the following calc script example:

```mdx
FIX("Total Expenses", ([Jan]), [[New York]])
"PY Actual" = @ALLOCATE("PY Actual"->"Total Expenses", @CHILDREN("Total Expenses"), "Budget", ,share);
ENDFIX
```

**Example 6-5  Inserting Using Member Context**

The following example updates the revised payroll based on previous year context.

```mdx
INSERT "([Measures].[Payroll])" TO "([Measures].[Revised_Payroll],[Year].CurrentMember.PrevMember)"
INTO [Test].[Basic]
FROM
  
  SELECT 
    {[Measures].[Payroll]} ON COLUMNS, 
    Descendants([Year]) ON ROWS 
  FROM [Test].[Basic] 
  WHERE ([Actual],[100-10],[New York])
```

**Example 6-6  Performing a Custom Calculation**

The following example runs a custom calculation on an aggregate storage database.

```mdx
WITH
  MEMBER [Amount Type].[AT1] AS
  CASE
    WHEN IS ([Account].CurrentMember, [ACC19802])
      THEN ([ACC19802],[CC10000],[ORG63],[Beginning Balance])
    WHEN IS([Account].CurrentMember, [ACC19803])
      THEN ([ACC19803],[FEB-05/06],[ORG00],[CC20000],[Beginning Balance]) * 2
    WHEN IS([Account].CurrentMember, [ACC19804])
      THEN ([ACC19804],[FEB-05/06],[ORG65],[CC19000],[Beginning Balance]) * 
        ([ACC19803],[Feb-08],[ORG63],[CC12000],[Beginning Balance])
    WHEN IS([Account].CurrentMember, [ACC19805])
      THEN ([ACC12000],[Beginning Balance]) + ([ACC19802],[Beginning Balance]) + 20
    WHEN IS([Account].CurrentMember, [ACC19806])
      THEN ([ACC19805],[Feb-08],[ORG63],[CC12000],[Beginning Balance]) - 
        0.00000002e7
```

---

Chapter 6

MDX Syntax and Grammar Rules

6-46
WHEN IS([Account].CurrentMember, [ACC19807])
    THEN 1
ELSE Missing
END

MEMBER [Amount Type].[AT3]
AS
[IIF]
    ([Amount Type].[AT1] < 0,
    [Amount Type].[AT1] * -1, Missing)'

MEMBER [Amount Type].[AT4]
AS
[IIF]
    ([Amount Type].[AT1] >= 0,
    [Amount Type].[AT1], Missing)'

MEMBER [Amount Type].[AT5]
AS
[IIF](IS([Organisation].CurrentMember, [ORG00])
    AND IS([Account].CurrentMember, [ACC19807]),
    SUM(Crossjoin{
        [ACC19801].Children,
        ([ORGT].Children),
        [Amount Type].[AT1]), Missing)'

MEMBER [Amount Type].[AT6]
AS
[IIF]
    ([Amount Type].[AT5] < 0,
    [Amount Type].[AT5] * -1, Missing)'

MEMBER [Amount Type].[AT7]
AS
[IIF]
    ([Amount Type].[AT5] >= 0,
    [Amount Type].[AT5], Missing)'

INSERT
    "([Amount Type].[AT3])"
    TO
    "([Allocations],[Beginning Balance Credit])"

    "([Amount Type].[AT4])"
    TO
    "([Allocations],[Beginning Balance Debit])"

    "([Amount Type].[AT6])"
    TO
    "([Allocations],[Beginning Balance Debit],[ORG66])"

    "([Amount Type].[AT7])"
    TO
    "([Allocations],[Beginning Balance Credit],[ORG66])"
INTO [Gl].[Basic]
FROM 
{ 
SELECT 
{[ACC19801].Children}
ON COLUMNS,
{Crossjoin(Crossjoin([ORGT].Children,[CCT].Children),
{[Amount Type].[AT1],
[Amount Type].[AT3],
[Amount Type].[AT4],
[Amount Type].[AT5],
[Amount Type].[AT6],
[Amount Type].[AT7]})}
ON ROWS
FROM [Gl].[Basic]
WHERE ([Actual],[PUBT],[OUTT], [Feb-08],[FRED],[ANLT])
};

Example 6-7 Performing a Custom Allocation

The following example runs a custom allocation on an aggregate storage database.

WITH 
MEMBER [Amount Type].[AT1] 
AS
'([Beginning Balance],[ORG63],[CC10000])'

MEMBER [Amount Type].[AT2] 
AS
'([Amount Type].[AT1]/
 Count(
 Crossjoin(
  {[Beginning Balance Credit]},
  CrossJoin(
   Descendants(
     [ORGT],
     [Organisation].Levels(0)
   ),
   Descendants([CCT],[Cost Centre].Levels(0))
  )
))'

MEMBER [Amount Type].[AT3] 
AS
'IIF([Amount Type].[AT2] < 0, [Amount Type].[AT2] * -1, Missing)'

MEMBER [Amount Type].[AT4] 
AS
'IIF([Amount Type].[AT2] >= 0, [Amount Type].[AT2], Missing)'
MEMBER [Amount Type].[AT5] AS
'IF(([Organisation].CurrentMember, [ORG00])
   AND IS([Cost Centre].CurrentMember,[CC19000])
   AND [Amount Type].[AT1] < 0, [Amount Type].[AT1] * -1, Missing)'

MEMBER [Amount Type].[AT6] AS
'IF (([Organisation].CurrentMember, [ORG00])
   AND IS([Cost Centre].CurrentMember,[CC19000])
   AND [Amount Type].[AT1] >= 0, [Amount Type].[AT1], Missing)'

INSERT "([Amount Type].[AT3],[Scenario])"
TO "([Allocations],[Beginning Balance Credit])"

"([Amount Type].[AT4],[Scenario])"
TO "([Allocations], [Beginning Balance Debit])"

"([Amount Type].[AT5],[Scenario])"
TO "([Allocations],[Beginning Balance Debit],[ORG63],[CC19000])"

"([Amount Type].[AT6],[Scenario])"
TO "([Allocations],[Beginning Balance Credit],[ORG63],[CC19000])"

INTO [Gl].[Basic]
FROM
{
    SELECT
        {[Amount Type].[AT1],
         [Amount Type].[AT2],
         [Amount Type].[AT3],
         [Amount Type].[AT4],
         [Amount Type].[AT5],
         [Amount Type].[AT6]}
    ON COLUMNS,
        (Crossjoin(
            [Acc19801].Children,
            CrossJoin(
                Descendants(
                    [ORGT],[Organisation].Levels(0)
                ),
                Descendants(
                    [CCT],[Cost Centre].Levels(0)
                )
            )
        )
    )
}
See Also

MDXINSERTREQUESTTIMEOUT
MDXINSERTBUFFERAGGMETHOD

MDX Export Specification

The export clause is a way to save query results to a file on Essbase. This is an alternative to viewing the query output on a client, and can be useful for large queries, or for exporting data to import later using a data load.

Syntax

[<with_section>]
EXPORT INTO FILE <file_name> [OVERWRITE <USING COLUMNDELIMITER <delimiter_character>>]
SELECT [axis_specification] [, <axis_specification>...]]
<subselect> | FROM <cube_specification>
[WHERE [slicer_specification]]

Table 6-14  MDX EXPORT Clause Elements

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file_name</td>
<td>The name of a text file in which to save the exported MDX query results. A file extension is not required.</td>
</tr>
<tr>
<td>OVERWRITE</td>
<td>Optional keyword specifying that if file_name already exists, overwrite it.</td>
</tr>
<tr>
<td>USING COLUMNDELIMITER</td>
<td>Optional argument specifying a character or word to use as a column separator. If omitted, the default MaxL column output is used, and the default column width is 20 characters.</td>
</tr>
<tr>
<td>delimiter_character</td>
<td></td>
</tr>
</tbody>
</table>

Notes

MDX Export is designed for large data exports. For optimal performance, Essbase treats the row axis as NON EMPTY, in a two-axis MDX Export query. This is the default behavior even if NON EMPTY is not specified. For more information about NON EMPTY, see MDX Axis Specifications.

Example

The following query

EXPORT INTO FILE "example" OVERWRITE USING COLUMNDELIMITER "#~"
SELECT
  ([Mar],[Apr]) ON COLUMNS,
  Crossjoin([100],[200]) , crossjoin([Actual],[Budget]),
  ([Opening Inventory],[Ending Inventory])) ON ROWS
FROM [Sample].[Basic]
WHERE ([New York]);

returns only minimal information to the client (where status 1 indicates successful query execution):

<table>
<thead>
<tr>
<th>Axis-1</th>
<th>(File)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(+-------------------+-------------------)</td>
<td></td>
</tr>
<tr>
<td>(Mdx Export)</td>
<td>1</td>
</tr>
</tbody>
</table>

The output file, example.txt, is saved to the database directory, and contains the actual query output:

```
Product#~Scenario#~Measures#~Mar#~Apr
Colas#~Actual#~Opening Inventory#~2041#~2108
Colas#~Actual#~Ending Inventory#~2108#~2250
Colas#~Budget#~Opening Inventory#~1980#~2040
Colas#~Budget#~Ending Inventory#~2040#~2170
Root Beer#~Actual#~Opening Inventory#~2378#~2644
Root Beer#~Actual#~Ending Inventory#~2644#~2944
Root Beer#~Budget#~Opening Inventory#~2220#~2450
Root Beer#~Budget#~Ending Inventory#~2450#~2710
```

**MDX Operators**

This section describes operators that can be used in MDX queries as part of numeric value expressions or search conditions.

**Mathematical Operators**

<table>
<thead>
<tr>
<th>Operator</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Adds. Also can be used as a unary operator.</td>
</tr>
<tr>
<td>-</td>
<td>Subtracts. Also can be used as a unary operator; for example, -5, -(Profit).</td>
</tr>
<tr>
<td>*</td>
<td>Multiplies.</td>
</tr>
<tr>
<td>/</td>
<td>Divides.</td>
</tr>
<tr>
<td>%</td>
<td>Evaluates percentage. For example, Member1%Member2 evaluates Member1 as a percentage of Member2. <strong>Note:</strong> Aggregate storage outline formulas cannot contain the % operator. In outline formulas, replace % with expression: (value1/value2)*100</td>
</tr>
</tbody>
</table>

**Conditional and Logical Operators**

Conditional operators take two operands and check for relationships between them, returning TRUE or FALSE.
Table 6-16  Conditional and Logical Operators in MDX

<table>
<thead>
<tr>
<th>Operator</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;</td>
<td>Data value is greater than.</td>
</tr>
<tr>
<td>&lt;</td>
<td>Data value is less than.</td>
</tr>
<tr>
<td>=</td>
<td>Data value is equal to.</td>
</tr>
<tr>
<td>&lt;&gt;</td>
<td>Data value is not equal to.</td>
</tr>
<tr>
<td>=&gt;</td>
<td>Data value is greater than or equal to.</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Data value is less than or equal to.</td>
</tr>
<tr>
<td>IN</td>
<td>The syntax for the IN operator is as follows:</td>
</tr>
<tr>
<td></td>
<td>&lt;property&gt; IN &lt;member&gt;</td>
</tr>
</tbody>
</table>

The first argument, <property> should be an attribute property; for example, Population in the following example.

The second argument, <member> or <character_string_literal>, should be an attribute member that is neither a level-0 member nor a generation-1 member; for example, Medium in the following example.

**Example**

The following filter evaluates the Population property (attribute) of the current member of Market dimension:

```mdx
Filter ([Market].Members, Market.CurrentMember.Population IN Medium)
```

If the population attribute of the current member is Medium, the expression returns TRUE.

**IS**

The IS operator syntax is as follows: `member1 IS member2`. The IS operator is equivalent to the IS function. For details and examples, see the IS function.

**Boolean Operators**

Boolean operators can be used in the following functions to perform conditional tests: Filter, Case, IIF, Generate. Boolean operators operate on boolean operands (TRUE/FALSE values).

See also MDX Functions that Return a Boolean.
Table 6-17 Boolean Operators in MDX

<table>
<thead>
<tr>
<th>Operator</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>Logical AND linking operator for multiple value tests. Result is TRUE if both conditions are TRUE. Otherwise the result is FALSE. For an example using AND, see Is_valid.</td>
</tr>
<tr>
<td>OR</td>
<td>Logical OR linking operator for multiple value tests. Result is TRUE if either condition is TRUE. Otherwise the result is FALSE.</td>
</tr>
<tr>
<td>NOT</td>
<td>Logical NOT operator. Result is TRUE if condition is FALSE. Result is FALSE if condition is TRUE. For an example using NOT, see Is_empty.</td>
</tr>
<tr>
<td>XOR</td>
<td>Logical XOR linking operator for multiple value tests. Result is TRUE if only one condition is TRUE. Otherwise the result is FALSE.</td>
</tr>
</tbody>
</table>

About MDX Properties

Properties describe certain characteristics of data and metadata. MDX enables users to write queries that use properties to retrieve and analyze data. Properties can be intrinsic or custom.

MDX Intrinsic Properties
MDX Custom Properties
MDX Property Expressions
MDX Optimization Properties
Querying for Member Properties in MDX
The Value Type of MDX Properties
MDX NULL Property Values

MDX Intrinsic Properties

Intrinsic properties are defined for members in all dimensions. In Essbase, the intrinsic MDX member properties defined for all members in an Essbase database outline are MEMBER_NAME, MEMBER_ALIAS, LEVEL_NUMBER, GEN_NUMBER, IS_EXPENSE, COMMENTS, and MEMBER_UNIQUE_NAME.

The MEMBER_NAME intrinsic property returns a member name string for each member.

The MEMBER_ALIAS intrinsic property returns a member alias string for each member.

The LEVEL_NUMBER intrinsic property returns the level number of each member.

The GEN_NUMBER intrinsic property returns the generation number of each member.
The IS_EXPENSE intrinsic property returns TRUE if a member has the Expense account

type, and FALSE otherwise. Example:

SELECT
[Measures].Members
  DIMENSION PROPERTIES [Measures].[IS_EXPENSE] on columns
from Sample.Basic;

The COMMENTS intrinsic property returns a comment string for each member where

applicable. Example:

SELECT
[Market].Members
  DIMENSION PROPERTIES [Market].[COMMENTS] on columns
from Sample.Basic;

The MEMBER_UNIQUE_NAME intrinsic property is a member-name property. It returns

NULL for unique members, and a system-generated key for duplicate members.

MDX Custom Properties

MDX in Essbase supports three types of custom properties: attribute properties, UDA

properties, and alias-table-name properties. Attribute properties are defined by the

attribute dimensions in an outline. In the Sample Basic database, the [Pkg Type]

attribute dimension describes the packaging characteristics of members in the Product
dimension. This information can be queried in MDX using the property name [Pkg

Type].

Attribute properties are defined only for specific dimensions and only for a specific

level in each dimension. For example, in the Sample Basic outline, [Ounces] is an

attribute property defined only for members in the Product dimension, and this

property has valid values only for the level-0 members of the Product dimension. The

[Ounces] property does not exist for other dimensions, such as Market. The [Ounces]

property for a non level-0 member in the Product dimension is a NULL value. The

attribute properties in an outline are identified by the names of attribute dimensions in

that outline.

The custom properties also include UDAs. For example, [Major Market] is a UDA

property defined on Market dimension members. It returns a TRUE value if [Major

Market] UDA is defined for a member, and FALSE otherwise.

Custom alias-table-name properties enable you to query for alias table names used by
each member returned in the output.

MDX Property Expressions

In addition to querying for intrinsic and custom properties of a member, you can also
query for MDX properties using the PROPERTY_EXPR function. This function enables
you to query for properties of related members based on a member value expression.
Syntax

PROPERTY_EXPR (dimension name, property_name, member_value_expression, display_name)

Table 6-18 PROPERTY_EXPR Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dimension name</td>
<td>The dimension name, or the keyword ALL. When a dimension name is specified, the property expression is evaluated for members from that dimension only. When the keyword ALL is specified, the property expression is evaluated for all members on the axis.</td>
</tr>
<tr>
<td>property_name</td>
<td>Property specification. One of the intrinsic properties (MEMBER_NAME, MEMBER_ALIAS, LEVEL_NUMBER, GEN_NUMBER, IS_EXPENSE, COMMENTS, or MEMBER_UNIQUE_NAME), or one of the custom properties (an attribute dimension name, alias-table name, or UDA specification).</td>
</tr>
<tr>
<td>member_value_expression</td>
<td>Member value expression. See &lt;member_value_expression&gt; ::= in MDX Grammar Rules.</td>
</tr>
<tr>
<td>display_name</td>
<td>Character string literal. The display name to use for the queried properties information in the query output.</td>
</tr>
</tbody>
</table>

Description

For every member on an axis from dimension name, the member_value_expression is evaluated with the current member from dimension name in the context. The property_name is evaluated on the output of member_value_expression. The specified display_name indicates the label to use for the queried properties output.

You can refer to the current member on the axis by using CurrentAxisMember.

Example

SELECT
{[100]}
ON COLUMNS,
Market.Levels(0).Members
DIMENSION PROPERTIES
PROPERTY_EXPR
{
  Market,
  MEMBER_NAME,
  Ancestor
  { Currentaxismember(),
```mdx
Currentaxismember().Dimension.Levels(1)
)"
"Parent_level_1"
),
PROPERTY_EXPR
(  
  Market,
  MEMBER_NAME,
  Ancestor
  (    
    Currentaxismember(),
    Currentaxismember().Dimension.Levels(2)
  ),
  "Parent_level_2"
)
ON ROWS
FROM Sample.Basic;
```

which returns the following grid (truncated):

### Table 6-19 Output Grid from MDX Example

<table>
<thead>
<tr>
<th>(axis)</th>
<th>Axis-1.properties</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>[New York]</td>
<td>Parent_level_1 = East,</td>
<td>3498</td>
</tr>
<tr>
<td></td>
<td>Parent_level_2 = market</td>
<td></td>
</tr>
<tr>
<td>[Massachusetts]</td>
<td>Parent_level_1 = East,</td>
<td>5105</td>
</tr>
<tr>
<td></td>
<td>Parent_level_2 = market</td>
<td></td>
</tr>
<tr>
<td>[Florida]</td>
<td>Parent_level_1 = East,</td>
<td>2056</td>
</tr>
<tr>
<td></td>
<td>Parent_level_2 = market</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

### MDX Optimization Properties

Optimization properties can improve the performance of formulas and calculated members, as well as the performance of queries that rely on them.

Optimization properties are applicable to outline members with formulas and calculated members only. Stored members are not associated with these properties.

The NONEMPTYMEMBER and NONEMPTYTUPLE properties enable MDX in Essbase to query on large sets of members or tuples while skipping formula execution on non-contributing values that contain only #MISSING data.

Because large sets tend to be very sparse, only a few members contribute to the input member (have non #MISSING values) and are returned. As a result, the use of NONEMPTYMEMBER and NONEMPTYTUPLE in calculated members and formulas conserves memory resources, allowing for better scalability, especially in concurrent user environments.

**NONEMPTYMEMBER**

NONEMPTYMEMBER `nonempty_member_list`
where nonempty_member_list is one or more comma-separated member names or calculated member names from the same dimension as the formula or calculated member.

Use a single NONEMPTYMEMBER property clause at the beginning of a calculated member or formula expression to indicate to Essbase that the value of the formula or calculated member is empty when any of the members specified in nonempty_member_list are empty.

**NONEMPTYMEMBER**

NONEMPTYMEMBER "("nonempty_member_list")"

where nonempty_member_list is one or more comma-separated member names or calculated member names, each from different dimensions.

Use a single NONEMPTYTUPLE property clause at the beginning of a calculated member or formula expression to indicate to Essbase that the value of the formula or calculated member is empty when the cell value at the tuple given in nonempty_member_list is empty.

**Example**

The following query calculates a member [3 Month Units] that represents the sum of Units (items per package) for the current month and the previous two months, where Units data is not missing.

The calculated member [3 Month Units] calculates Units shipped for last three months. If the units shipped for [MTD] (units shipped in a year) is empty, it follows that Units data is empty for all months in the Year; therefore, the sum of Units shipped for last three months is also empty. Because the row axis in the query is very large and sparse, the NONEMPTYTUPLE property would significantly increase the performance of the query in this case.

WITH MEMBER [Measures].[3 Month Units] AS

```
NONEMPTYTUPLE ( [Units], [MTD] )
```

 Sum( 
   { 
     ClosingPeriod(Time.Generations(5), Time.CurrentMember), 
     Time.CurrentMember.Lag(1), 
     Time.CurrentMember.Lag(2) 
   }, 
   Units 
 ),

 SELECT 
 ( Units, [3 Month Units] ) ON COLUMNS,
 NON EMPTY 
 CrossJoin( 
   Stores.Levels(0).Members,
   [Store Manager].Children 
 )
 ON ROWS
FROM Asosamp.Sample
WHERE (Mar);

This query returns the following grid (results truncated):

<table>
<thead>
<tr>
<th>(axis)</th>
<th>Items Per Package</th>
<th>3 Month Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>(017589, Carrie)</td>
<td>610</td>
<td>1808</td>
</tr>
<tr>
<td>(020408, Debra)</td>
<td>584</td>
<td>1778</td>
</tr>
<tr>
<td>(020486, Kalluri)</td>
<td>551</td>
<td>1670</td>
</tr>
<tr>
<td>(047108, Kimberley)</td>
<td>593</td>
<td>1723</td>
</tr>
<tr>
<td>(051273, Madhukar)</td>
<td>541</td>
<td>1642</td>
</tr>
<tr>
<td>(056098, Melisse)</td>
<td>607</td>
<td>1750</td>
</tr>
</tbody>
</table>

Querying for Member Properties in MDX

Properties can be used inside an MDX query in two ways. In the first approach, you can list the dimension and property combinations for each axis set. When a query is executed, the specified property is evaluated for all members from the specified dimension and included in the result set.

For example, on the column axis, the following query will return the GEN_NUMBER information for every Market dimension member. On the row axis, the query returns MEMBER_ALIAS information for every Product dimension member.

```
SELECT [Market].Members
  DIMENSION PROPERTIES [Market].[GEN_NUMBER] on columns,
Filter ([Product].Members, Sales > 5000)
  DIMENSION PROPERTIES [Product].[MEMBER_ALIAS] on rows
from Sample.Basic
```

When querying for member properties using the DIMENSION PROPERTIES section of an axis, a property can be identified by the dimension name and the name of the property, or just by using the property name itself. When a property name is used by itself, that property information is returned for all members from all dimensions on that axis, for which that property applies.

**Note:**

When a property name is used by itself within the DIMENSION PROPERTIES section, do not use brackets [] around the property name.
In the following query, the MEMBER_ALIAS property is evaluated on the row axis for both Year and Product dimensions.

```mdx
SELECT
    [Market].Members
    DIMENSION PROPERTIES [Market].[GEN_NUMBER] on columns,
    CrossJoin([Product].Children, Year.Children)
    DIMENSION PROPERTIES MEMBER_ALIAS on rows
from Sample.Basic
```

In a second approach, properties can be used inside value expressions in an MDX query. For example you can filter a set based on a value expression that uses properties of members in input set.

The following query returns all caffeinated products that are packaged in cans.

```mdx
Select
    Filter([Product].levels(0).members,
        [Product].CurrentMember.Caffeinated and
        [Product].CurrentMember.[Pkg Type] = "Can")
    Dimension Properties
        [Caffeinated], [Pkg Type] on columns
```

The following query uses the UDA [Major Market] to calculate the value [BudgetedExpenses] based on whether the current member of the Market dimension is a major market or not.

```mdx
With
    MEMBER [Measures].[BudgetedExpenses] AS
        'IIF([Market].CurrentMember.[Major Market],
            [Marketing] * 1.2, [Marketing])'
Select
    {[Measures].[BudgetedExpenses]} on columns,
    Market.Members on rows
Where
    {[Budget]}
```

The following queries use alias table names.

```mdx
SELECT
    [Product].Members
    DIMENSION PROPERTIES [Default] on columns
from Sample.Basic;

SELECT
    [Product].Members
    DIMENSION PROPERTIES [Long Names] on columns
from Sample.Basic;
```
The Value Type of MDX Properties

The value of an MDX property in Essbase can be a numeric, Boolean, or string type. MEMBER_NAME and MEMBER_ALIAS properties return string values. LEVEL_NUMBER and GEN_NUMBER properties return numeric values.

The attribute properties return numeric, Boolean, or string values based on the attribute dimension type. For example, in Sample Basic, the [Ounces] attribute property is a numeric property. The [Pkg Type] attribute property is a string property. The [Caffeinated] attribute property is a Boolean property.

Essbase allows attribute dimensions with date types. The date type properties are treated as numeric properties in MDX. When comparing these property values with dates, you need to use the TODATE function to convert date strings to numeric before comparison.

The following query returns all Product dimension members that have been introduced on date 03/25/1996. Since the property [Intro Date] is a date type, the TODATE function must be used to convert the date string "03-25-1996" to a number before comparing it.

```mdx
Select
  Filter ([Product].Members,
    [Product].CurrentMember.[Intro Date] = TODATE("mm-dd-yyyy","03-25-1996")) on columns
```

When a property is used in a value expression, you must use it appropriately based on its value type: string, numeric, or Boolean.

MDX NULL Property Values

Not all members may have valid values for a given property name. For example, the MEMBER_ALIAS property returns an alternate name for a given member as defined in the outline; however, not all members may have aliases defined. In these cases A NULL value would be returned for those members that do not have aliases.

In the following query:

```mdx
SELECT
  [Year].Members
  DIMENSION PROPERTIES MEMBER_ALIAS on columns
```

none of the members in the Year dimension have aliases defined for them. Therefore, the query returns NULL values for the MEMBER_ALIAS property for members in the Year dimension.

The attribute properties are defined for members of a specific dimension and a specific level in that dimension. In the Sample Basic database, the [Ounces] property is defined only for level-0 members of the Product dimension.
Therefore, if you query for the [Ounces] property of a member from the Market dimension, as shown in the following query, you will get a syntax error:

```
SELECT
  Filter([Market].members,
  [Market].CurrentMember.[Ounces] = 32) on columns
```

Additionally, if you query for the [Ounces] property of a non level-0 member of the dimension, you will get a NULL value.

When using property values in value expressions, you can use the function IsValid() to check for NULL values. The following query returns all Product dimension members with [Ounces] property value of 12, after eliminating members with NULL values.

```
Select
  Filter([Product].Members,
  IsValid([Product].CurrentMember.[Ounces]) and
  [Product].CurrentMember.[Ounces] = 12) on columns
```

**MDX Comments**

This section describes how to add comments to MDX queries.

**Syntax**

MDX supports two types of syntax for comments:

1. MDX supports the "C++ style" comments that are also supported by the Essbase Server calculator framework. This type of comment can cover multiple lines. Everything in between is ignored by the MDX parser.

   Example:

   ```
   /*
   commented text is ignored by parser
   */
   ```

2. MDX supports inline comments beginning with two hyphens. Beginning with two hyphens, the rest of the line is ignored by the MDX parser. A new line ends the span of the comment.

   Example:

   ```
   -- short comment can go on till line break
   ```

**Example**

The following example uses both styles of comments:

```
/* Query the profit figures in each
  market for the "100" products
```
SELECT {{[Market].levels(1).members}} --L1 members of Market
ON COLUMNS,
--Cross of the “100” products and their profit figures:
CrossJoin ([100].children, [Profit].children)
ON ROWS
FROM Sample.Basic

MDX Query Limits

Overview

The following concepts are applicable to understanding MDX query limits.

Table 6-21  MDX Query Limit Concepts

<table>
<thead>
<tr>
<th>Concept</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NON EMPTY processing</td>
<td>Refers to how Essbase processes MDX queries and sets when the NON EMPTY keywords are used in an axis specification. The NON EMPTY specification optimizes processing by suppressing slices that would contain entirely #MISSING values.</td>
</tr>
</tbody>
</table>
### Table 6-21  (Cont.) MDX Query Limit Concepts

<table>
<thead>
<tr>
<th>Concept</th>
<th>Description</th>
</tr>
</thead>
</table>
| Cluster elements/symmetric sets   | Although an MDX set is a collection of **tuples**, internally, Essbase represents sets using clusters and tuples. A cluster is a type of set derived using the **CrossJoin** function, where the arguments to CrossJoin are sets from one dimension only.  
A cluster can also be thought of as a symmetric set. The following set is a symmetric set and can be stored as one cluster.  
\[
\text{CROSSJOIN(Products.LEVELS(0).MEMBERS, [Market].LEVELS(0).MEMBERS)}
\]
|                                  |                                                                                                                                                                                                          |
| A **tuple**                      | is a collection of members from different dimensions. The following set has one tuple.  
\[
\{([Product].Product_1, [Market].Market_1)}
\]
|                                  |                                                                                                                                                                                                          |
|                                  | The following set is a **union** of the above two sets. It is stored internally as a cluster and a tuple.  
\[
\text{UNION(}
\text{CROSSJOIN(Products.LEVELS(0).MEMBERS, [Market].LEVELS(0).MEMBERS)}
\text{,}
\text{\{([Product].Product_1, [Market].Market_1)}}
\text{\})}
\]                                                                                                                                 |
| Compact set                      | A set is stored in compact form if it can be internally represented as a cluster or symmetric set.                                                                                                                                                               |
### Table 6-21  (Cont.) MDX Query Limit Concepts

<table>
<thead>
<tr>
<th>Concept</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flattened set</td>
<td>A set that must be internally expanded into tuples is a flattened set. Flattened sets consume more memory to be processed. Certain MDX functions, such as Order, need to flatten sets in order to process them correctly. Therefore, certain functions, as listed in the next section, have different set size or query limits. The following set is an example of a flattened set.</td>
</tr>
<tr>
<td></td>
<td>{(Colas, East) (Colas, West) (Colas, South) (Colas, Central) (Root Beer, East) (Root Beer, West) (Root Beer, South) (Root Beer, Central (Cream Soda, East) (Cream Soda, West) (Cream Soda, South) (Cream Soda, Central) (Fruit Soda, East) (Fruit Soda, West) (Fruit Soda, South) (Fruit Soda, Central) }</td>
</tr>
<tr>
<td>Asymmetric set</td>
<td>The following set is stored internally as a collection of a tuple element and a cluster element. The two elements cannot be combined into a single element. Such sets are called asymmetric sets.</td>
</tr>
<tr>
<td></td>
<td>UNION({(Colas, East}) CROSSJOIN([Product].CHILDREN, [Market].CHILDREN))</td>
</tr>
</tbody>
</table>

### MDX Query Limits

The following size limitations apply to MDX queries, sets, and certain functions.
Note:
The following exception applies to the general query limits: If the database being queried is the target database of a partition, the maximum size of a cube region you can query using MDX is $2^{32}$ potential cells.

Table 6-22  MDX Query Limit Descriptions and Units

<table>
<thead>
<tr>
<th>Limitations</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of cells in a query region defined by all axis sets in an MDX query with NON EMPTY clause</td>
<td>$2^{640}$</td>
</tr>
<tr>
<td>Number of cells that can be returned to a client after NON EMPTY processing</td>
<td>$2^{32}$</td>
</tr>
<tr>
<td>Number of cells in a query region defined by all axis sets in an MDX query with no NON EMPTY clause</td>
<td>$2^{32}$</td>
</tr>
<tr>
<td>Number of tuples in an axis set with NON EMPTY directive after NON EMPTY processing</td>
<td>$2^{28}$</td>
</tr>
<tr>
<td>Size of a set in compact form</td>
<td>$2^{640}$</td>
</tr>
<tr>
<td>Size of a set in flattened form</td>
<td>$2^{32}$</td>
</tr>
<tr>
<td>Number of elements in a set</td>
<td>$2^{32}$</td>
</tr>
<tr>
<td>Number of members (from all dimensions) in a cluster element</td>
<td>$2^{32}$</td>
</tr>
<tr>
<td>Number of cells in a query after applying non empty cell processing</td>
<td>$2^{32}$</td>
</tr>
<tr>
<td>Size of a set that can be processed by the following functions:</td>
<td>Less than $2^{28}$</td>
</tr>
<tr>
<td>• Distinct</td>
<td></td>
</tr>
<tr>
<td>• Except</td>
<td></td>
</tr>
<tr>
<td>• Filter</td>
<td></td>
</tr>
<tr>
<td>• Intersect</td>
<td></td>
</tr>
<tr>
<td>• Ntile</td>
<td></td>
</tr>
<tr>
<td>• Order</td>
<td></td>
</tr>
<tr>
<td>• Percentile</td>
<td></td>
</tr>
<tr>
<td>• Rank</td>
<td></td>
</tr>
<tr>
<td>• TopPercent</td>
<td></td>
</tr>
<tr>
<td>• BottomPercent</td>
<td></td>
</tr>
<tr>
<td>• TopSum</td>
<td></td>
</tr>
<tr>
<td>• BottomSum</td>
<td></td>
</tr>
<tr>
<td>• Hierarchize</td>
<td></td>
</tr>
<tr>
<td>• Union (with removal of duplicates)</td>
<td></td>
</tr>
<tr>
<td>• NonEmptySubset (output set size)</td>
<td></td>
</tr>
<tr>
<td>• TopCount (output set size)</td>
<td></td>
</tr>
<tr>
<td>• BottomCount (output set size)</td>
<td></td>
</tr>
</tbody>
</table>
Table 6-22  (Cont.) MDX Query Limit Descriptions and Units

<table>
<thead>
<tr>
<th>Limitations</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEssOpMdxQuery Java API interface or EssMdx C API functions</td>
<td>• Maximum number of tuples/clusters on an axis—$2^{29} - 1$</td>
</tr>
<tr>
<td></td>
<td>• Maximum number of cells (when cell status is requested)—$2^{26} - 1$</td>
</tr>
<tr>
<td></td>
<td>• Maximum number of cells (when cell status is not requested)—approximately $2^{27} - 1$</td>
</tr>
<tr>
<td>MDX queries run through MaxL</td>
<td>• Maximum number of columns—$2^{29} - 1$</td>
</tr>
<tr>
<td></td>
<td>• Maximum number of rows—$2^{29} - 1$</td>
</tr>
</tbody>
</table>

Aggregate Storage and MDX Outline Formulas

To write formulas for block storage outlines, Essbase provides a set of calculation functions and operators known as the Calculator, or Calc, language. The Calculator language cannot be used to write member formulas for aggregate storage databases. Formulas in aggregate storage outlines use the MDX language.

The following sections provide information for rewriting Calculator formulas in MDX for outlines that have been migrated from block storage to aggregate storage. Before attempting to rewrite formulas you should be familiar with the basic workings of aggregate storage outlines. See Designing and Maintaining Essbase Cubes for more information about aggregate storage.

Translating Calculator Functions to MDX Functions

When translating Calculator formulas to MDX, keep in mind the following differences between block storage outlines and aggregate storage outlines:

- The storage characteristics of a member and hence all its associated cells are defined in a block storage outline through Dynamic Calc (and Dynamic Calc and Store) attributes, and stored attributes. Such attributes do not exist in an aggregate storage outline. Upper level members along an explicitly tagged accounts dimension and members with formulas attached to them are always calculated dynamically in such a database.

- In block storage outlines, calculation order is dependent on the order in which members appear in the outline whereas formulas are executed in order of their dependencies in aggregate storage outlines. In addition, calculation order in the event of ambiguity in the evaluation of a cell, and two-pass calculation tags are not required in an aggregate storage outline.

- The layout of block storage outlines and the separation of dimensions into dense and sparse has an effect on the semantics of certain calculations, giving rise to concepts such as top-down calculation mode, cell and block calculation mode, and create-blocks on equations. The simplicity of the aggregate storage outlines, which do not separate dimensions into dense and sparse, do not require such concepts.

General Guidelines for Translating Calculator Formulas to MDX

This section provides some general guidelines for translating Calculator formulas to MDX.
Be certain that the application has been redesigned to use an aggregate storage outline. In this regard, make certain that formulas do not reference any block-storage specific outline constructs, such as variance functions that rely on expense tagging, or functions that operate on shared members (for example, @RDESCENDANTS). Such constructs are not valid in aggregate storage outlines.

Rewrite each function in the formulas attached to an explicitly tagged accounts dimension for which a direct counterpart in MDX exists. Table 6-23 provides specific information and examples. Then identify functions for which an indirect rewrite is required. Table 6-23 also provides information and examples for these functions.

Understand the calculation order semantics for the formulas in the block storage outline. Organize the dependent formulas in the aggregate storage outline carefully to achieve the same results as block storage.

If formulas reference custom-defined functions or macros consider rewriting them, if possible, using other MDX functions.

The following table lists all functions in the Calculator language and their analogs in MDX (and vice versa). Where a direct analog does not exist, transformation rules and examples are provided.

### Table 6-23 Calculator to MDX Function Mapping

<table>
<thead>
<tr>
<th>Calculator</th>
<th>MDX</th>
<th>Remarks/Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>@ABS</td>
<td>Abs</td>
<td>Calculator</td>
</tr>
<tr>
<td>@ABS(Actual-Budget)</td>
<td>MDX</td>
<td>Abs([Actual]-[Budget])</td>
</tr>
<tr>
<td>@ALLANCESTORS</td>
<td>Ancestors</td>
<td>Shared members are not relevant to aggregate storage outlines.</td>
</tr>
<tr>
<td>@ALIAS</td>
<td>Not required.</td>
<td>In MDX, the argument to @ALIAS can be passed as-is to the outer function.</td>
</tr>
<tr>
<td>@ANCEST</td>
<td>Ancestor with CurrentMember as input. Use a tuple to combine the result with the optional third argument to the @ANCEST function.</td>
<td>Calculator</td>
</tr>
<tr>
<td>@ANCEST(Product,2,Sales)</td>
<td>MDX</td>
<td>(Sales, Ancestor(Product.CurrentMember, Product.Generations(2)) )</td>
</tr>
</tbody>
</table>
Table 6-23  (Cont.) Calculator to MDX Function Mapping

<table>
<thead>
<tr>
<th>Calculator</th>
<th>MDX</th>
<th>Remarks/Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>@ANCESTORS</td>
<td>Ancestors</td>
<td>Calculator</td>
</tr>
<tr>
<td></td>
<td></td>
<td>@ANCESTORS(&quot;New York&quot;)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MDX</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ancestors([New York].parent, [Market].levels(2))</td>
</tr>
<tr>
<td>@ANCESTVAL</td>
<td>Ancestor with CurrentMember as input. Use a tuple to combine the result with the optional third argument to the @ANCESTVAL function.</td>
<td>Calculator</td>
</tr>
<tr>
<td></td>
<td></td>
<td>@ANCESTVAL(Product,2,Sales)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MDX</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Sales, Ancestor(Product.CurrentMember, Product.Generations(2)) ) .Value</td>
</tr>
<tr>
<td>@ATTRIBUTE</td>
<td>Attribute</td>
<td>Calculator</td>
</tr>
<tr>
<td></td>
<td></td>
<td>@ATTRIBUTE(Can)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MDX</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Attribute([Can])</td>
</tr>
<tr>
<td>@ATTRIBUTEBVAL</td>
<td>[BaseDim].CurrentMember.Attribute Dim</td>
<td>See About MDX Properties.Calculator</td>
</tr>
<tr>
<td></td>
<td></td>
<td>@ATTRIBUTEBVAL(Caffeinated)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MDX</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Product.CurrentMember.Caffeinated</td>
</tr>
<tr>
<td>Calculator</td>
<td>MDX</td>
<td>Remarks/Examples</td>
</tr>
<tr>
<td>---------------------</td>
<td>----------------------------------------------</td>
<td>------------------------------------------------------</td>
</tr>
<tr>
<td>@ATTRIBUTESVAL</td>
<td><code>[BaseDim].CurrentMember.Attribute Dim</code></td>
<td>See About MDX Properties.Calculator</td>
</tr>
<tr>
<td></td>
<td></td>
<td>@ATTRIBUTESVAL(&quot;Pkg Type&quot;)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MDX</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Product.CurrentMember.[Pkg Type]</td>
</tr>
<tr>
<td>@ATTRIBUTEVAL</td>
<td><code>[BaseDim].CurrentMember.Attribute Dim</code></td>
<td>See About MDX Properties.Calculator</td>
</tr>
<tr>
<td></td>
<td></td>
<td>@ATTRIBUTEVAL(Ounces)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MDX</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Product.CurrentMember.Ounces</td>
</tr>
</tbody>
</table>
## Table 6-23  (Cont.) Calculator to MDX Function Mapping

<table>
<thead>
<tr>
<th>Calculator</th>
<th>MDX</th>
<th>Remarks/Examples</th>
</tr>
</thead>
</table>
| @AVG       | If the dimensionality of all elements in the input set to @AVG is the same, use Avg. Translate SKIPNONE to INCLUDEEMPTY. If the dimensionality of all elements in the input set to @AVG is not the same, then perform average by explicitly adding the tuples and dividing by the set cardinality (the number of tuples in the set). | Note that the MDX Avg function skips missing cell values by default. **Calculator**  
@AVG(SKIPMISSING, @CHILDREN(East))  
**MDX**  
Avg([East].Children)  
If SKIPMISSING is replaced by SKIPNONE, the translation changes to:  
Avg([East].Children, Sales, INCLUDEEMPTY)  
For SKIPZERO, the translation is:  
Avg([East].Children,  
IIF(Market.CurrentMember.Value=0, Missing,  
IIF(Market.CurrentMember=Missing,0,  
Market.CurrentMember.Value  
)}  
)}  
For SKIPBOTH, the translation is:  
Avg([East].Children,  
IIF(Market.CurrentMember=0, Missing,  
Market.CurrentMember.Value  
)}  
}
Table 6-23  (Cont.) Calculator to MDX Function Mapping

<table>
<thead>
<tr>
<th>Calculator</th>
<th>MDX</th>
<th>Remarks/Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>@AVGRANGE</td>
<td>CrossJoin (first argument, set created out of second argument). The rest is similar to @AVG when the dimensionality of all elements of the input set is identical.</td>
<td>Calculator: @AVGRANGE(SKIPMISSING, Sales, @CHILDREN(West)) MDX: Avg(CrossJoin({Sales}, {[West].Children})) If SKIPMISSING is replaced by SKIPNONE, the translation becomes: Avg({[West].Children}, Sales, INCLUDEEMPTY) If SKIPZERO is used, then the translation is: Avg({[West].Children}, IIF(Sales = 0, Missing, IIF(Sales = Missing, 0, Sales))</td>
</tr>
<tr>
<td>@CHILDREN</td>
<td>Children</td>
<td>Calculator: @CHILDREN(Market) MDX: Children(Market) or Market.Children</td>
</tr>
<tr>
<td>Calculator</td>
<td>MDX</td>
<td>Remarks/Examples</td>
</tr>
<tr>
<td>------------------</td>
<td>---------------------------------------------</td>
<td>---------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>@CONCATENATE</td>
<td>Concat</td>
<td>Calculator</td>
</tr>
<tr>
<td></td>
<td></td>
<td>@MEMBER(@CONCATENATE(&quot;Qtr1&quot;, &quot;1&quot;));</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MDX</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Concat(&quot;01&quot;, &quot;01&quot;)</td>
</tr>
<tr>
<td>@CORRELATION</td>
<td>Not supported in MDX.</td>
<td>.</td>
</tr>
<tr>
<td>@COUNT</td>
<td>Use Count if SKIPNONE.</td>
<td>Calculator</td>
</tr>
<tr>
<td></td>
<td>Use NonEmptyCount if SKIPMISSING.</td>
<td>@COUNT(SKIPMISSING,@RANGE(Sales, Children(Product)))</td>
</tr>
<tr>
<td></td>
<td>For SKIPZERO, see the example in the next</td>
<td>MDX</td>
</tr>
<tr>
<td></td>
<td>column.</td>
<td>NonEmptyCount(CrossJoin({Sales}, {Product.Children}))</td>
</tr>
<tr>
<td></td>
<td>For SKIPBOTH, use Count</td>
<td>Note that Count always counts including the empty cells, whereas NonEmptyCount</td>
</tr>
<tr>
<td></td>
<td></td>
<td>does not.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For SKIPNONE, the translation is:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Count(Product.Children)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For SKIPZERO, the translation is:</td>
</tr>
</tbody>
</table>
|                  |                                             | NonEmptyCount(Product.Children, IIF(Sales=0, Missing, IIF(Sales = Missing, 0, sales)) )

Note that Count always counts including the empty cells, whereas NonEmptyCount does not. For SKIPNONE, the translation is:

Count(Product.Children)

For SKIPZERO, the translation is:

NonEmptyCount(Product.Children, IIF(Sales=0, Missing, IIF(Sales = Missing, 0, sales))

)
<table>
<thead>
<tr>
<th>Calculator</th>
<th>MDX</th>
<th>Remarks/Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>@CURGEN</td>
<td>Generation (CurrentMember (dimension))</td>
<td>Calculator @CURGEN(Year)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MDX</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Year.CurrentMember.Generation</td>
</tr>
<tr>
<td>@CURLEV</td>
<td>Level (CurrentMember (dimension))</td>
<td>Calculator @CURLEV(Year)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MDX</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Year.CurrentMember.Level</td>
</tr>
<tr>
<td>@CURRMBR</td>
<td>CurrentMember</td>
<td>Calculator @CURRMBR(Product)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MDX</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[Product].CurrentMember</td>
</tr>
<tr>
<td>@CURRMBRRANGE</td>
<td>RelMemberRange</td>
<td>Calculator @CURRMBRRANGE(Year, LEV, 0, -1, 1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MDX</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RelMemberRange (Year.CurrentMember, 1, 1, LEVEL)</td>
</tr>
<tr>
<td>@DESCENDANTS</td>
<td>Descendants (member)</td>
<td>See MDX Descendants documentation for examples.</td>
</tr>
</tbody>
</table>
Table 6-23  (Cont.) Calculator to MDX Function Mapping

<table>
<thead>
<tr>
<th>Calculator</th>
<th>MDX</th>
<th>Remarks/Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>@EXP</td>
<td>Exp</td>
<td>Calculator @EXP(&quot;Variance %&quot;/100);</td>
</tr>
<tr>
<td>@FACTORIAL</td>
<td>Factorial</td>
<td>Calculator @FACTORIAL(5)</td>
</tr>
<tr>
<td>@GEN, @LEV</td>
<td>Generation, Level</td>
<td>.</td>
</tr>
<tr>
<td>@GENMBRS, @LEVMBRS</td>
<td>layer.Members</td>
<td>.</td>
</tr>
<tr>
<td>@IANCESTORS</td>
<td>Ancestors</td>
<td>Shared members are not relevant to aggregate storage outlines.</td>
</tr>
<tr>
<td>@ICHILDREN</td>
<td>Union(member, member.Children)</td>
<td>Shared members are not relevant to aggregate storage outlines.</td>
</tr>
<tr>
<td>@IDESCENDANTS</td>
<td>Descendants(member)</td>
<td>Calculator @IDESCENDANTS(Market)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MDX Union({Market}, {Market.children})</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MDX Descendants(Market)</td>
</tr>
<tr>
<td>Calculator</td>
<td>MDX</td>
<td>Remarks/Examples</td>
</tr>
<tr>
<td>----------------</td>
<td>--------------------------------------------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>@ILSIBLINGS</td>
<td>MemberRange</td>
<td>Calculator</td>
</tr>
<tr>
<td></td>
<td>(member.FirstSibling, member)</td>
<td>@ILSIBLINGS(Florida)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MDX</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MemberRange(Florida.FirstSibling, Florida.Lag(1))</td>
</tr>
<tr>
<td>@INT</td>
<td>Int</td>
<td>Calculator</td>
</tr>
<tr>
<td></td>
<td></td>
<td>@INT(104.504)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MDX</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Int(104.504)</td>
</tr>
<tr>
<td>@ISACCTYPE</td>
<td>IsAccType</td>
<td>See MDX IsAccType documentation for examples.</td>
</tr>
<tr>
<td>@ISANCEST</td>
<td>IsAncestor</td>
<td>Calculator</td>
</tr>
<tr>
<td></td>
<td></td>
<td>@ISANCEST(California)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MDX</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IsAncestor(Market.CurrentMember, California)</td>
</tr>
<tr>
<td>@ISCHILD</td>
<td>IsChild</td>
<td>See MDX IsChild documentation for examples.</td>
</tr>
<tr>
<td>Calculator</td>
<td>MDX</td>
<td>Remarks/Examples</td>
</tr>
<tr>
<td>----------------</td>
<td>----------------------------------------------------------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>@ISDESC</td>
<td>See examples.</td>
<td>Calculator</td>
</tr>
<tr>
<td></td>
<td>@ISDESC(Market)</td>
<td>MDX</td>
</tr>
<tr>
<td></td>
<td>IsAncestor([Market], [Market].Dimension.CurrentMember)</td>
<td>or</td>
</tr>
<tr>
<td></td>
<td>Count(Intersect({Member.Descendants}, {Member.dimension.CurrentMember}) = 1</td>
<td></td>
</tr>
<tr>
<td>@ISGEN</td>
<td>IsGeneration</td>
<td>Calculator</td>
</tr>
<tr>
<td></td>
<td>@ISGEN(Market, 2)</td>
<td>MDX</td>
</tr>
<tr>
<td></td>
<td>IsGeneration(Market.CurrentMember, 2)</td>
<td></td>
</tr>
<tr>
<td>@ISANCEST</td>
<td>IIF(Is(member, ancestormember) OR IsAncestor(member, ancestormember), &lt;true-part&gt;, &lt;false-part&gt;)</td>
<td>Calculator</td>
</tr>
<tr>
<td></td>
<td>@ISANCEST(California)</td>
<td>MDX</td>
</tr>
<tr>
<td></td>
<td>IIF( Is(Market.CurrentMember, California) OR IsAncestor(Market.CurrentMember, California), &lt;true-part&gt;, &lt;false-part&gt; )</td>
<td></td>
</tr>
<tr>
<td>@ISIBLINGS</td>
<td>Siblings(member)</td>
<td>Returns a set that includes the specified member and its siblings.</td>
</tr>
<tr>
<td>Calculator</td>
<td>MDX</td>
<td>Remarks/Examples</td>
</tr>
<tr>
<td>---------------</td>
<td>----------------------------------------------------------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>@ISICHLDF</td>
<td><code>IIF(Is(member, childmember) OR IsChild(member, childmember), &lt;true-part&gt;, &lt;false-part&gt;)</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>@ISICHLDF(South)</code></td>
<td></td>
</tr>
</tbody>
</table>
|               | `IIF(  
|               | Is(Market.CurrentMember, South) 
|               | OR IsChild(Market.CurrentMember, South), <true-part>, <false-part> 
|               | )`                                                                   |                  |
| @ISIDESC      | See examples.                                                        |                  |
|               | `@ISIDESC(South)`                                                   |                  |
|               | `IIF(  
|               | (Count(Intersect({[South].Descendants}, {South}) = 1 
|               | OR Is(CurrentMember, [South])) 
|               | )`                                                                   |                  |
| @ISIPARENT    | `IIF(Is(member, parentmember)`                                      |                  |
|               | `@ISIPARENT(Qtr1)`                                                  |                  |
|               | `IIF(  
|               | Is(Time.CurrentMember, [Qtr1]) 
|               | OR IsChild([Qtr1], Time.CurrentMember), <true-part>, <false-part> 
<p>|               | )`                                                                   |                  |</p>
<table>
<thead>
<tr>
<th>Calculator</th>
<th>MDX</th>
<th>Remarks/Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>@ISISIBLING</td>
<td><code>IsSibling(member, siblingmember)</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td>@ISISIBLING(Qtr2)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MDX</td>
<td></td>
</tr>
<tr>
<td></td>
<td>IIF(</td>
<td></td>
</tr>
<tr>
<td></td>
<td>IsSibling(</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[Qtr2],</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Time.CurrentMember</td>
<td></td>
</tr>
<tr>
<td></td>
<td>),</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;true-part&gt;, &lt;false-part&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>@ISLEV</td>
<td><code>IsLevel</code></td>
<td></td>
</tr>
<tr>
<td>@ISMBR</td>
<td>IIF(Count(Intersect (member-set,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>member)) = 1, true-part, false-part)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Calculator allows a collection of</td>
<td></td>
</tr>
<tr>
<td></td>
<td>members or cross members that do</td>
<td></td>
</tr>
<tr>
<td></td>
<td>not subscribe to the rules of an MDX</td>
<td></td>
</tr>
<tr>
<td></td>
<td>set to appear as the second argument.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>This functionality cannot be easily</td>
<td></td>
</tr>
<tr>
<td></td>
<td>replicated without enumerating each</td>
<td></td>
</tr>
<tr>
<td></td>
<td>element of the second set and testing</td>
<td></td>
</tr>
<tr>
<td></td>
<td>for intersection.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>However, if the second argument</td>
<td></td>
</tr>
<tr>
<td></td>
<td>subscribes to MDX set rules then the</td>
<td></td>
</tr>
<tr>
<td></td>
<td>translation is easier, as shown.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>For example:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Calculator</td>
<td></td>
</tr>
<tr>
<td></td>
<td>@ISMBR(&quot;New York&quot;:&quot;New Hampshire&quot;)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MDX</td>
<td></td>
</tr>
<tr>
<td></td>
<td>IIF(</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Count(</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Intersect(</td>
<td></td>
</tr>
<tr>
<td></td>
<td>{MemberRange([New York],</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[New Hampshire]),</td>
<td></td>
</tr>
<tr>
<td></td>
<td>{Market.CurrentMember}</td>
<td></td>
</tr>
<tr>
<td></td>
<td>) = 1,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;true-part&gt;, &lt;false-part&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>)</td>
<td></td>
</tr>
</tbody>
</table>
### Table 6-23  (Cont.) Calculator to MDX Function Mapping

<table>
<thead>
<tr>
<th>Calculator</th>
<th>MDX</th>
<th>Remarks/Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>@ISPARENT</td>
<td>Use IsChild.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>@ISPARENT(&quot;New York&quot;)</td>
<td>MDX</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IsChild(Market.CurrentMember, [New York])</td>
</tr>
<tr>
<td>@ISSAMEGEN, @ISSAMELEV</td>
<td>IIF (member.Generation = CurrentMember(%dimension).Generation, &lt;true-part&gt;, &lt;false-part&gt;)</td>
<td>Calculator</td>
</tr>
<tr>
<td></td>
<td>@ISSAMEGEN(West)</td>
<td>MDX</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IIF(</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ordinal(</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Market.CurrentMember.Generation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>= Ordinal(West.Generation),&lt;true-part&gt;,&lt;false-part&gt; )</td>
</tr>
<tr>
<td>@ISSIBLING</td>
<td>IsSibling</td>
<td>See MDX IsSibling documentation for examples.</td>
</tr>
<tr>
<td>@ISUDA</td>
<td>IsUda</td>
<td>See MDX IsUda documentation for examples.</td>
</tr>
<tr>
<td>@LIST</td>
<td>.</td>
<td>If the member set does not subscribe to MDX set rules, then explicit enumeration is required. For rangelist use CrossJoin(member, set).</td>
</tr>
<tr>
<td>@LN, @LOG, @LOG10</td>
<td>Ln, Log, Log10</td>
<td>.</td>
</tr>
<tr>
<td>Calculator</td>
<td>MDX</td>
<td>Remarks/Examples</td>
</tr>
<tr>
<td>------------</td>
<td>----------------------------------------------------------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>@LSIBLINGS</td>
<td>MemberRange(member.FirstSibling, member.Lag(1))</td>
<td>Calculator @LSIBLINGS(Qtr4)</td>
</tr>
<tr>
<td>@RSIBLINGS</td>
<td>MemberRange(member.Lead(1), member.LastSibling)</td>
<td>MDX MemberRange([Qtr4].FirstSibling, [Qtr4].Lag(1))</td>
</tr>
<tr>
<td>@MATCH</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>@MAX</td>
<td>Max</td>
<td>Use Max if argument list is a set. Otherwise, rewrite logic using Case constructs by explicit enumeration of the argument list. Calculator @MAX(Jan:Mar)</td>
</tr>
</tbody>
</table>

Chapter 6 Aggregate Storage and MDX Outline Formulas
<table>
<thead>
<tr>
<th>Calculator</th>
<th>MDX</th>
<th>Remarks/Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>@MAXRANGE</td>
<td>Max</td>
<td>Calculator</td>
</tr>
<tr>
<td></td>
<td>@MAXRANGE(Sales,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>@CHILDREN(Qtr1))</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>MDX</td>
</tr>
<tr>
<td></td>
<td>Max(</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CrossJoin(</td>
<td></td>
</tr>
<tr>
<td></td>
<td>{Sales},</td>
<td></td>
</tr>
<tr>
<td></td>
<td>{{Qtr1}.Children}</td>
<td></td>
</tr>
<tr>
<td></td>
<td>)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>OR</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Max([Qtr1].Children, Sales)</td>
<td></td>
</tr>
<tr>
<td>@MAXS</td>
<td>Max</td>
<td>Calculator</td>
</tr>
<tr>
<td></td>
<td>@MAXS(SKIPMISSING,Sales,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>@CHILDREN(Qtr1))</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>MDX</td>
</tr>
<tr>
<td></td>
<td>Max(</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Children([Qtr1]),Sales)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>For SKIPZERO, the translation is:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Max (Children ([Qtr1]), IIF</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(Sales = 0, MISSING, Sales))</td>
<td></td>
</tr>
<tr>
<td></td>
<td>For SKIPBOTH, the translation is the same as for SKIPZERO, because Max skips missing values by default.</td>
<td></td>
</tr>
<tr>
<td>Calculator</td>
<td>MDX</td>
<td>Remarks/Examples</td>
</tr>
<tr>
<td>---------------------</td>
<td>------------------------------------------</td>
<td>---------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>@MAXSRANGE</td>
<td>Max</td>
<td>@MAXSRANGE(SKIPMISSING, Sales, @CHILDREN(Qtr1))</td>
</tr>
<tr>
<td></td>
<td>MDX</td>
<td>Max(Children([Qtr1]),Sales)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For SKIPZERO, the translation is:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Max (Children ([Qtr1]), IIF (Sales = 0, MISSING, Sales))</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For SKIPBOTH, the translation is the same as for SKIPZERO, because Max skips</td>
</tr>
<tr>
<td></td>
<td></td>
<td>missing values by default.</td>
</tr>
<tr>
<td>@MDANCESTVAL</td>
<td>Use Ancestor, Value, and Currentmember as shown in the example.</td>
<td>@MDANCESTVAL(2, Market, 2, Product, 2, Sales)</td>
</tr>
<tr>
<td></td>
<td>MDX</td>
<td>Construct a tuple consisting of Sales from the Measures dimension, the ancestor</td>
</tr>
<tr>
<td></td>
<td></td>
<td>of the current member along the Market dimension, and the ancestor of the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>current member along the Product dimension. Then get the value of the tuple.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Sales, Ancestor(Market.CurrentMember, 2), Ancestor(Product.CurrentMember, 2)).Value</td>
</tr>
</tbody>
</table>
## Table 6-23  (Cont.) Calculator to MDX Function Mapping

<table>
<thead>
<tr>
<th>Calculator</th>
<th>MDX</th>
<th>Remarks/Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>@MDPARENTVAL</td>
<td>Use <code>Parent</code>, <code>Value</code>, and <code>CurrentMember</code> as shown in the example.</td>
<td></td>
</tr>
<tr>
<td>@MDSHIFT</td>
<td>See MDX equivalent for <code>@NEXT</code>, and repeat it for each dimension that needs to be shifted. <code>CrossJoin</code> the results from each dimension and get the value of the final tuple. See comments for <code>@MDANCESTVAL</code>.</td>
<td></td>
</tr>
<tr>
<td>@MEDIAN</td>
<td>Not supported in MDX.</td>
<td></td>
</tr>
<tr>
<td>@MEMBER</td>
<td>Not needed in MDX.</td>
<td></td>
</tr>
<tr>
<td>@MERGE</td>
<td><code>Union(set1,set2)</code></td>
<td></td>
</tr>
</tbody>
</table>

### MDX

Construct a tuple consisting of Sales from the Measures dimension, the parent of the current member along the Market dimension, and the parent of the current member along the Product dimension. Then get the value of the tuple.

```
(Sales,
 Market.CurrentMember.Parent,
 Product.CurrentMember.Parent
 ).Value
```
<table>
<thead>
<tr>
<th>Calculator</th>
<th>MDX</th>
<th>Remarks/Examples</th>
</tr>
</thead>
</table>
| @MIN            | Min                                      | Use Min if argument list is a set. Otherwise, rewrite logic using **Case** constructs by explicit enumeration of the argument list. **Calculator**  
@MIN(Jan:Mar)    | **MDX**                                  | Min(MemberRange([Jan], [Mar]))                                                  |
| @MINRANGE       | Min                                      | **Calculator**                                                                  
@MINRANGE(Sales, @CHILDREN(Qtr1)) | **MDX**                                  | Min(CrossJoin( 
  {Sales}, 
  {[Qtr1].Children} 
); 
  [Qtr1].Children, Sales) |

OR

Min([Qtr1].Children, Sales)
Table 6-23  (Cont.) Calculator to MDX Function Mapping

<table>
<thead>
<tr>
<th>Calculator</th>
<th>MDX</th>
<th>Remarks/Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>@MINS</td>
<td>Min</td>
<td>@MINS(SKIPMISSING, Sales, @CHILDREN(Qtr1))</td>
</tr>
</tbody>
</table>

**MDX**

Min(
Filter(
Children([Qtr1]),
Sales <> Missing
)
)

For SKIPZERO, the translation is:

Min(
Filter(
Children([Qtr1]),
Sales <> 0
)
)

For SKIPBOTH, the translation is:

Min(
Filter(
Children([Qtr1]),
Sales <> 0 AND
Sales <> Missing
)
)
### Table 6-23 (Cont.) Calculator to MDX Function Mapping

<table>
<thead>
<tr>
<th>Calculator</th>
<th>MDX</th>
<th>Remarks/Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>@MINSRANGE</td>
<td>Min</td>
<td>@MINSRANGE(SKIPMISSING, Sales, @CHILDREN(Qtr1))</td>
</tr>
<tr>
<td></td>
<td>MDX</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Min(</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Filter(Children([Qtr1]), Sales &lt;&gt; Missing</td>
<td></td>
</tr>
<tr>
<td></td>
<td>)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>For SKIPZERO, the translation is:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Min(</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Filter(Children([Qtr1]), Sales &lt;&gt; 0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>For SKIPBOTH, the translation is:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Min {</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Filter(Children([Qtr1]), Sales &lt;&gt; 0 AND Sales &lt;&gt; Missing</td>
<td></td>
</tr>
<tr>
<td></td>
<td>)</td>
<td></td>
</tr>
<tr>
<td>@MOD</td>
<td>Mod</td>
<td>.</td>
</tr>
<tr>
<td>@MODE</td>
<td>Not supported in MDX.</td>
<td>.</td>
</tr>
<tr>
<td>@NAME</td>
<td>Not needed in MDX.</td>
<td>.</td>
</tr>
</tbody>
</table>
Table 6-23 (Cont.) Calculator to MDX Function Mapping

<table>
<thead>
<tr>
<th>Calculator</th>
<th>MDX</th>
<th>Remarks/Examples</th>
</tr>
</thead>
</table>
| @NEXT          | @NEXT(member,[n, range]) returns the nth cell value in the range from the supplied member. The function returns a missing value if the supplied member does not exist in the range. If range is not specified, level-0 members of the Time dimension are used. MDX does not have an equivalent function for an arbitrary range. However, if the range is restricted to members from a specific level or generation, then using NextMember (if n=1) or Lead/Lag will work as shown in the sample translation. This is probably the common case. | Calculator

Calculator: @Next(Cash)

MDX: (NextMember([Year].CurrentMember, LEVEL), [Cash]).Value

Alternative: Calculator

@Next(Cash, 2)

MDX: CrossJoin{

Year.CurrentMember.Lead(2, LEVEL),
Cash).Value

@NEXTS         | Not supported in MDX.                                               | .                |
| @PARENT       | Parent                                                               | .                |
| @PARENTVAL    | Parent with CurrentMember as input. Use a tuple to combine the result with the optional second argument to the @PARENTVAL function. | Calculator

Calculator: @PARENTVAL(Market, Sales)

MDX: ([Sales],
[Market].CurrentMember.Parent).Value

@POWER         | Power                                                                | .                |
<table>
<thead>
<tr>
<th>Calculator</th>
<th>MDX</th>
<th>Remarks/Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>@PRIOR</td>
<td>@PRIOR([member],[n, range]) returns the n-th cell value in the range from the supplied member. The function returns a missing value if the supplied member does not exist in the range. If range is not specified, level-0 members of the Time dimension are used. MDX does not have an equivalent function for an arbitrary range. However, if the range is restricted to members from a specific level or generation, then using PrevMember (if n=1) or Lead/Lag will work as shown in the sample translation. This is probably the common case.</td>
<td>Calculator @Prior(Cash)</td>
</tr>
<tr>
<td></td>
<td>MDX</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PrevMember(Year.CurrentMember, LEVEL), [Cash]).Value</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Alternative:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Calculator @Prior(Cash, 2)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MDX</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(Year.CurrentMember.Lag(2, LEVEL), [Cash]).Value</td>
<td></td>
</tr>
<tr>
<td>@PRIORS</td>
<td>Not supported in MDX.</td>
<td></td>
</tr>
<tr>
<td>@RANGE</td>
<td>CrossJoin([member], [rangeset])</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Calculator automatically uses level-0 members of the Time dimension if a range is unspecified. That feature does not exist in MDX, so you must explicitly include the range.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MDX</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CrossJoin([Sales], [East].Children)</td>
<td></td>
</tr>
<tr>
<td>@RANK</td>
<td>Not supported in MDX. This is a vector function.</td>
<td></td>
</tr>
<tr>
<td>@REMAINDER</td>
<td>Remainder</td>
<td></td>
</tr>
</tbody>
</table>
Table 6-23  (Cont.) Calculator to MDX Function Mapping

<table>
<thead>
<tr>
<th>Calculator</th>
<th>MDX</th>
<th>Remarks/Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>@REMOVE</td>
<td><code>Except(set1, set2)</code></td>
<td>Translation will work only if set1 and set2 are true MDX sets.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Calculator</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>@REMOVE(@CHILDREN(East), @LIS T(&quot;New York&quot;, Connecticut))</code></td>
</tr>
<tr>
<td>@ROUND</td>
<td><code>Round</code></td>
<td>.</td>
</tr>
<tr>
<td>@SHIFT</td>
<td><code>See @PRIOR and @NEXT.</code></td>
<td>.</td>
</tr>
<tr>
<td>@SIBLINGS</td>
<td><code>Siblings</code></td>
<td>.</td>
</tr>
<tr>
<td>@STDEV, @STDEVP, @STDEVRANGE</td>
<td>Not supported in MDX.</td>
<td>.</td>
</tr>
<tr>
<td>@SUBSTRING</td>
<td><code>Not supported in MDX.</code></td>
<td>.</td>
</tr>
<tr>
<td>@SUM</td>
<td><code>Sum</code></td>
<td>Convert each element of the explist to a tuple so that collectively the tuples can form a set.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Calculator</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>@SUMRANGE(&quot;New York&quot;, Jan:Jun)</code></td>
</tr>
<tr>
<td>@SUMRANGE</td>
<td><code>Sum(CrossJoin(member, Xrangelist))</code></td>
<td>.</td>
</tr>
<tr>
<td>@TODATE</td>
<td><code>Todate</code></td>
<td>.</td>
</tr>
<tr>
<td>@TRUNCATE</td>
<td><code>Truncate</code></td>
<td>.</td>
</tr>
<tr>
<td>@UDA</td>
<td><code>Uda</code></td>
<td>.</td>
</tr>
<tr>
<td>@VAR, @VARPER</td>
<td><code>Arg1 - Arg2</code></td>
<td>An aggregate storage outline has no expense tags. Therefore, variance functionality defaults to subtraction.</td>
</tr>
<tr>
<td>@VARIANCE, @VARIANCEP</td>
<td><code>Not supported in MDX.</code></td>
<td>.</td>
</tr>
<tr>
<td>@WITHATTR</td>
<td><code>WithAttr</code></td>
<td>.</td>
</tr>
<tr>
<td>@XRANGE</td>
<td><code>Not supported in MDX.</code></td>
<td>.</td>
</tr>
<tr>
<td>@XREF</td>
<td><code>Not supported in MDX.</code></td>
<td>.</td>
</tr>
</tbody>
</table>
MDX Function Return Values

Functions can be used to generate metadata and/or value information that you need to pass to a SELECT statement. Becoming proficient with the functions reduces the need to enumerate tuples, members, numeric values, or other needed values explicitly in the set specifications of a query. More importantly, using functions allows in-depth analysis of your database.

This section contains a listing of query functions by return value. The possible return values are described in these topics:

- MDX Functions that Return a Member
- MDX Functions that Return a Set
- MDX Functions that Return a Tuple
- MDX Functions that Return a Number
- MDX Functions that Return a Dimension
- MDX Functions that Return a Layer
- MDX Functions that Return a Boolean
- MDX Functions that Return a Date
- MDX Functions that Return a String

MDX Functions that Return a Member

The following functions return a member or a member value expression.

Table 6-24  MDX Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ancestor</td>
<td>Returns a member that is an ancestor of the specified member, at a specified generation or level.</td>
</tr>
<tr>
<td>ClosingPeriod</td>
<td>Returns the last descendant of a layer, or the last child of the Time dimension.</td>
</tr>
<tr>
<td>Cousin</td>
<td>Returns a child member at a matching outline level and location as a member from another parent.</td>
</tr>
<tr>
<td>CurrentAxisMember</td>
<td>Returns the current axis member in the context of a member value expression argument.</td>
</tr>
<tr>
<td>CurrentMember</td>
<td>Returns the current member in the input dimension. Current is in the context of query execution mechanics. Use in combination with iterative functions such as Filter.</td>
</tr>
<tr>
<td>DateToMember</td>
<td>Returns the date-hierarchy member specified by the input date.</td>
</tr>
<tr>
<td>DefaultMember</td>
<td>Returns the default member in the input dimension.</td>
</tr>
<tr>
<td>FirstChild</td>
<td>Returns the first child of the input member.</td>
</tr>
</tbody>
</table>
Table 6-24  (Cont.) MDX Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>FirstSibling</td>
<td>Returns the first child of the input member's parent.</td>
</tr>
<tr>
<td>Lag</td>
<td>Using the default order of members in a database outline, returns a member that is ( n ) steps behind the input member.</td>
</tr>
<tr>
<td>LastChild</td>
<td>Returns the last child of the input member.</td>
</tr>
<tr>
<td>LastSibling</td>
<td>Returns the last child of the input member's parent.</td>
</tr>
<tr>
<td>Lead</td>
<td>Using the default order of members in a database outline, returns a member that is ( n ) steps past the input member.</td>
</tr>
<tr>
<td>NextMember</td>
<td>Returns the member (in the same layer) that is one step past the input member.</td>
</tr>
<tr>
<td>OpeningPeriod</td>
<td>Returns the first descendant of a layer, or the first child of the Time dimension.</td>
</tr>
<tr>
<td>ParallelPeriod</td>
<td>Returns a member from a prior time period as the specified or default time member.</td>
</tr>
<tr>
<td>Parent</td>
<td>Returns a member's parent.</td>
</tr>
<tr>
<td>PrevMember</td>
<td>Returns the member (in the same layer) that is one step prior to the input member.</td>
</tr>
<tr>
<td>StrToMbr</td>
<td>Converts a string to a member name.</td>
</tr>
</tbody>
</table>

MDX Functions that Return a Set

The following categories of functions return a set or a set value expression.

- Pure Set Functions
- Metadata-based Set Functions
- Data-based Set Functions

Pure Set Functions

Functions in this category derive their results without getting any further information from the cube.

Table 6-25  MDX Pure Set Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>CrossJoin</td>
<td>Returns a cross-section of two sets from different dimensions.</td>
</tr>
<tr>
<td>Distinct</td>
<td>Deletes duplicate tuples from a set.</td>
</tr>
<tr>
<td>Except</td>
<td>Returns a subset containing the differences between two sets.</td>
</tr>
<tr>
<td>Generate</td>
<td>For each tuple in set1, return set2.</td>
</tr>
<tr>
<td>Head</td>
<td>Returns the first ( n ) members or tuples present in a set.</td>
</tr>
<tr>
<td>Intersect</td>
<td>Returns the intersection of two input sets.</td>
</tr>
</tbody>
</table>
Table 6-25  (Cont.) MDX Pure Set Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subset</td>
<td>Returns a subset from a set, in which the subset is a numerically specified range of tuples.</td>
</tr>
<tr>
<td>Tail</td>
<td>Returns the last ( n ) members or tuples present in a set.</td>
</tr>
<tr>
<td>TupleRange</td>
<td>Returns the range of tuples between (and inclusive of) two tuples at the same level.</td>
</tr>
<tr>
<td>Union</td>
<td>Returns the union of two input sets.</td>
</tr>
</tbody>
</table>

Metadata-based Set Functions

Functions in this category derive their results using metadata information from the cube.

Table 6-26  MDX Metadata-based Set Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ancestors</td>
<td>Returns a set of ancestors up to a specified layer or distance.</td>
</tr>
<tr>
<td>Attribute</td>
<td>Returns all base members that are associated with the specified attribute member.</td>
</tr>
<tr>
<td>Children</td>
<td>Returns all child members of the input member.</td>
</tr>
<tr>
<td>Descendants</td>
<td>Returns the set of descendants of a member at specified layers.</td>
</tr>
<tr>
<td>DrilddownByLayer</td>
<td>Drills down members of a set that are at a specified layer.</td>
</tr>
<tr>
<td>DrilddownMember</td>
<td>Drills down on any members or tuples of (&lt;\text{set1}&gt;) that are also found in (&lt;\text{set2}&gt;).</td>
</tr>
<tr>
<td>DrilupByLayer</td>
<td>Drills up the members of a set that are below a specified layer.</td>
</tr>
<tr>
<td>DrilupMember</td>
<td>Tests two sets for common ancestors, and drills up members in the first set to the layer of the ancestors which are present in the second set.</td>
</tr>
<tr>
<td>Extract</td>
<td>Returns a subset containing only the tuples of a specified dimensionality.</td>
</tr>
<tr>
<td>Hierarchize</td>
<td>Sorts members according to the default member ordering as represented in the database outline.</td>
</tr>
<tr>
<td>LastPeriods</td>
<td>Returns a set of members ending either at the specified member or at the current member in the time dimension.</td>
</tr>
<tr>
<td>MemberRange</td>
<td>Returns the range of members positioned between two input members (inclusive) at the same generation or level.</td>
</tr>
<tr>
<td>Members</td>
<td>Returns a set of all members of a given dimension, hierarchy, or layer.</td>
</tr>
</tbody>
</table>
### Table 6-26  (Cont.) MDX Metadata-based Set Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>PeriodsToDate</td>
<td>Returns a set of dynamic-time-series members from the beginning of a given layer up to a given member in that layer (or up to the default member); or, returns members up to the current member of the Time dimension.</td>
</tr>
<tr>
<td>RelMemberRange</td>
<td>Returns a set based on the relative position of the specified member.</td>
</tr>
<tr>
<td>Siblings</td>
<td>Returns the siblings of the input member.</td>
</tr>
<tr>
<td>Uda</td>
<td>Returns all members that share a specified user-defined attribute.</td>
</tr>
<tr>
<td>WithAttr</td>
<td>Returns all base members that are associated with an attribute member of the specified type.</td>
</tr>
<tr>
<td>AttributeEx</td>
<td>Given the varying attribute member and the perspective setting, returns the associated base member list.</td>
</tr>
<tr>
<td>WithAttrEx</td>
<td>Given the varying attribute dimension, condition, predicate, and perspective setting, returns the base member list satisfying the predicate.</td>
</tr>
<tr>
<td>xTD</td>
<td>Functions returning period-to-date values.</td>
</tr>
</tbody>
</table>

### Data-based Set Functions

Functions in this category derive their results using data values from the cube.

### Table 6-27  MDX Data-based Set Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>BottomCount</td>
<td>Returns a set of ( n ) elements ordered from smallest to largest, optionally based on an evaluation.</td>
</tr>
<tr>
<td>BottomPercent</td>
<td>Returns the smallest possible subset, with elements listed from smallest to largest, of a set for which the total results of a numeric evaluation are at least a given percentage.</td>
</tr>
<tr>
<td>BottomSum</td>
<td>Returns the smallest possible subset, with elements listed from smallest to largest, of a set for which the total results of a numeric evaluation are at least a given sum.</td>
</tr>
<tr>
<td>Case</td>
<td>Performs conditional expressions.</td>
</tr>
<tr>
<td>Filter</td>
<td>Returns those parts of a set which meet the criteria of a search condition.</td>
</tr>
<tr>
<td>IIF</td>
<td>Performs a conditional test, and returns an appropriate numeric expression or set depending on whether the test evaluates to true or false.</td>
</tr>
<tr>
<td>Leaves</td>
<td>Returns the set of level 0 (leaf) members that contribute to the value of the specified member.</td>
</tr>
</tbody>
</table>
### MDX Data-based Set Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Order</td>
<td>Sorts members of a set in order based on an expression.</td>
</tr>
<tr>
<td>TopCount</td>
<td>Returns a set of ( n ) elements ordered from largest to smallest, optionally based on an evaluation.</td>
</tr>
<tr>
<td>TopPercent</td>
<td>Returns the smallest possible subset, with elements listed from largest to smallest, of a set for which the total results of a numeric evaluation are at least a given percentage.</td>
</tr>
<tr>
<td>TopSum</td>
<td>Returns the smallest possible subset, with elements listed from largest to smallest, of a set for which the total results of a numeric evaluation are at least a given sum.</td>
</tr>
</tbody>
</table>

### MDX Tuple Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>CurrentTuple</td>
<td>Returns the current tuple in a set. Current is in the context of query execution mechanics. Use in combination with iterative functions such as Filter.</td>
</tr>
<tr>
<td>Item</td>
<td>Extracts a member from a tuple.</td>
</tr>
</tbody>
</table>

### MDX Numeric Value Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abs</td>
<td>Returns absolute value of an expression.</td>
</tr>
<tr>
<td>Aggregate</td>
<td>Aggregates the Accounts member based on its Time Balance behavior.</td>
</tr>
<tr>
<td>Avg</td>
<td>Returns the average of values found in the tuples of a set.</td>
</tr>
<tr>
<td>Case</td>
<td>Performs conditional expressions.</td>
</tr>
<tr>
<td>CellValue</td>
<td>Returns the numeric value of the current cell.</td>
</tr>
<tr>
<td>CoalesceEmpty</td>
<td>Returns the first non #Missing value from the given value expressions.</td>
</tr>
<tr>
<td>Count</td>
<td>Returns the count of the number of tuples in a set.</td>
</tr>
<tr>
<td>DateDiff</td>
<td>Returns the difference between two input dates.</td>
</tr>
<tr>
<td>Function</td>
<td>Result</td>
</tr>
<tr>
<td>-----------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DatePart</td>
<td>Returns a number representing a date part (such as Week).</td>
</tr>
<tr>
<td>EnumText</td>
<td>Returns the text value corresponding to a numeric value in a text list.</td>
</tr>
<tr>
<td>EnumValue</td>
<td>Returns the internal numeric value for a text value in a text list.</td>
</tr>
<tr>
<td>Exp</td>
<td>Returns the exponent of an expression.</td>
</tr>
<tr>
<td>Factorial</td>
<td>Returns the factorial of an expression.</td>
</tr>
<tr>
<td>IIF</td>
<td>Performs a conditional test, and returns an appropriate numeric expression or set depending on whether the test evaluates to true or false.</td>
</tr>
<tr>
<td>InStr</td>
<td>Returns a number specifying the position of the first occurrence of one string within another.</td>
</tr>
<tr>
<td>Int</td>
<td>Returns the next lowest integer value of an expression.</td>
</tr>
<tr>
<td>Len</td>
<td>Returns length of a string.</td>
</tr>
<tr>
<td>Ln</td>
<td>Returns the natural logarithm of an expression.</td>
</tr>
<tr>
<td>Log</td>
<td>Returns the logarithm of an expression to a specified base.</td>
</tr>
<tr>
<td>Log10</td>
<td>Returns the base-10 logarithm of an expression.</td>
</tr>
<tr>
<td>Max</td>
<td>Returns the maximum of values found in the tuples of a set.</td>
</tr>
<tr>
<td>Median</td>
<td>Returns the value of the median tuple of a set.</td>
</tr>
<tr>
<td>Min</td>
<td>Returns the minimum of values found in the tuples of a set.</td>
</tr>
<tr>
<td>Mod</td>
<td>Returns the modulus (remainder value) of a division operation.</td>
</tr>
<tr>
<td>NonEmptyCount</td>
<td>Returns the count of the number of tuples in a set that evaluate to nonempty values.</td>
</tr>
<tr>
<td>NTile</td>
<td>Returns a division number of a tuple in a set.</td>
</tr>
<tr>
<td>Ordinal</td>
<td>Returns a number indicating depth in the hierarchy.</td>
</tr>
<tr>
<td>Percentile</td>
<td>Returns the value of the tuple that is at a given percentile of a set.</td>
</tr>
<tr>
<td>Power</td>
<td>Returns the value of the numeric value expression raised to power.</td>
</tr>
<tr>
<td>Rank</td>
<td>Returns the numeric position of a tuple in a set.</td>
</tr>
<tr>
<td>RealValue</td>
<td>Returns a value for the specified member or tuple without the inherited attribute dimension context.</td>
</tr>
<tr>
<td>Remainder</td>
<td>Returns the remainder value of the numeric value expression.</td>
</tr>
<tr>
<td>Round</td>
<td>Rounds a numeric value expression to the specified number of digits.</td>
</tr>
</tbody>
</table>
### Table 6-29  (Cont.) MDX Numeric Value Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stddev</td>
<td>Calculates standard deviation based on a sample.</td>
</tr>
<tr>
<td>Stddevp</td>
<td>Calculates standard deviation based on a population.</td>
</tr>
<tr>
<td>StrToNum</td>
<td>Converts a string to a number.</td>
</tr>
<tr>
<td>Sum</td>
<td>Returns the sum of values of tuples in a set.</td>
</tr>
<tr>
<td>Todate</td>
<td>Converts a date string to a value that is usable in calculations.</td>
</tr>
<tr>
<td>Truncate</td>
<td>Removes the fractional part of a numeric value expression, returning the integer.</td>
</tr>
</tbody>
</table>

### MDX Functions that Return a Dimension

The **Dimension** function returns the dimension that contains the input element.

### MDX Functions that Return a Layer

The following functions return a **layer**. A layer is used to group the members of a dimension by hierarchical depth.

In Essbase, a layer is either a generation or a level, indicated by a name or a number.

### Table 6-30  MDX Layer Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generation</td>
<td>Returns the generation of the input member.</td>
</tr>
<tr>
<td>Generations</td>
<td>Returns the generation specified by the input numerical depth and the input dimension or hierarchy.</td>
</tr>
<tr>
<td>Level</td>
<td>Returns the level of the input member.</td>
</tr>
<tr>
<td>Levels</td>
<td>Returns the level specified by the input numerical depth and the input dimension or hierarchy.</td>
</tr>
</tbody>
</table>

### MDX Functions that Return a Boolean

The following functions return a Boolean (TRUE or FALSE).

### Table 6-31  MDX Boolean Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Is</td>
<td>Returns TRUE if two members are identical.</td>
</tr>
<tr>
<td>IsAccType</td>
<td>Returns TRUE if the current member has the associated accounts tag.</td>
</tr>
<tr>
<td>IsAncestor</td>
<td>Returns TRUE if the first member is an ancestor of the second member.</td>
</tr>
</tbody>
</table>
### Table 6-31 (Cont.) MDX Boolean Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>IsChild</td>
<td>Returns TRUE if the first member is a child of the second member.</td>
</tr>
<tr>
<td>IsEmpty</td>
<td>Returns True if the value of an input numeric-value-expression is #MISSING.</td>
</tr>
<tr>
<td>IsGeneration</td>
<td>Returns TRUE if the member is in a specified generation.</td>
</tr>
<tr>
<td>IsLeaf</td>
<td>Returns TRUE if the member is a level-0 member.</td>
</tr>
<tr>
<td>IsLevel</td>
<td>Returns TRUE if the member is in a specified level.</td>
</tr>
<tr>
<td>IsSibling</td>
<td>Returns TRUE if the first member is a sibling of the second member.</td>
</tr>
<tr>
<td>IsUda</td>
<td>Returns TRUE if the member has the associated UDA tag (user-defined attribute).</td>
</tr>
<tr>
<td>IsValid</td>
<td>Returns TRUE if the specified element validates successfully.</td>
</tr>
<tr>
<td>Contains</td>
<td>Returns TRUE if a tuple is found within a set.</td>
</tr>
</tbody>
</table>

### MDX Functions that Return a Date

The following functions return a date.

### Table 6-32 MDX Date Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>DateRoll</td>
<td>To the given date, rolls (adds or subtracts) a number of specific time intervals, returning another date.</td>
</tr>
<tr>
<td>GetFirstDate</td>
<td>Returns the start date for a date-hierarchy member.</td>
</tr>
<tr>
<td>GetLastDate</td>
<td>Returns the end date for a date-hierarchy member.</td>
</tr>
<tr>
<td>GetNextDay</td>
<td>To the given date and the week day, gets the next date after input date that corresponds to the week day.</td>
</tr>
<tr>
<td>GetFirstDay</td>
<td>For a given date_part, returns the first day of the time interval for the input date.</td>
</tr>
<tr>
<td>GetLastDay</td>
<td>For a given date_part, returns the last day of the time interval for the input date.</td>
</tr>
<tr>
<td>TodateEx</td>
<td>Converts date strings to dates.</td>
</tr>
<tr>
<td>Today</td>
<td>Returns a number representing the current date.</td>
</tr>
<tr>
<td>JulianDate</td>
<td>For the given UNIX date, gets its Julian date.</td>
</tr>
<tr>
<td>UnixDate</td>
<td>For the given Julian date, gets its UNIX date.</td>
</tr>
</tbody>
</table>
MDX Functions that Return a String

The following functions return a string.

**Table 6-33  MDX String Functions**

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>DateFormat</td>
<td>Formats date strings.</td>
</tr>
<tr>
<td>Concat</td>
<td>Concatenates input strings.</td>
</tr>
<tr>
<td>Left</td>
<td>Returns a specified number of characters from the left side of the string.</td>
</tr>
<tr>
<td>Right</td>
<td>Returns a specified number of characters from the right side of the string.</td>
</tr>
<tr>
<td>LTrim</td>
<td>Trims whitespace on the left of the string.</td>
</tr>
<tr>
<td>RTrim</td>
<td>Trims whitespace on the right of the string.</td>
</tr>
<tr>
<td>Lower</td>
<td>Converts upper-case string to lower case.</td>
</tr>
<tr>
<td>Upper</td>
<td>Converts lower-case string to upper case.</td>
</tr>
<tr>
<td>Substring</td>
<td>Returns the substring between a starting and ending position.</td>
</tr>
<tr>
<td>NumToStr</td>
<td>Converts a double-precision floating-point value into a decimal string.</td>
</tr>
</tbody>
</table>

**MDX Function List**

Consult the Contents pane for a list of MDX functions by return value.

**Table 6-34  MDX Function List**

<table>
<thead>
<tr>
<th>Alphabetical List of MDX Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abs</td>
</tr>
<tr>
<td>Aggregate</td>
</tr>
<tr>
<td>Ancestor</td>
</tr>
<tr>
<td>Ancestors</td>
</tr>
<tr>
<td>Attribute</td>
</tr>
<tr>
<td>AttributeEx</td>
</tr>
<tr>
<td>Avg</td>
</tr>
<tr>
<td>BottomCount</td>
</tr>
<tr>
<td>BottomPercent</td>
</tr>
<tr>
<td>BottomSum</td>
</tr>
<tr>
<td>Case</td>
</tr>
<tr>
<td>CellValue</td>
</tr>
<tr>
<td>Children</td>
</tr>
<tr>
<td>ClosingPeriod</td>
</tr>
<tr>
<td>CoalesceEmpty</td>
</tr>
<tr>
<td>Concat</td>
</tr>
<tr>
<td>Contains</td>
</tr>
<tr>
<td>Count</td>
</tr>
</tbody>
</table>
Table 6-34  (Cont.) MDX Function List

<table>
<thead>
<tr>
<th>Alphabetical List of MDX Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cousin</td>
</tr>
<tr>
<td>CrossJoin</td>
</tr>
<tr>
<td>CrossJoinAttribute</td>
</tr>
<tr>
<td>CurrentAxisMember</td>
</tr>
<tr>
<td>CurrentMember</td>
</tr>
<tr>
<td>CurrentTuple</td>
</tr>
<tr>
<td>DateDiff</td>
</tr>
<tr>
<td>DatePart</td>
</tr>
<tr>
<td>DateRoll</td>
</tr>
<tr>
<td>DateToMember</td>
</tr>
<tr>
<td>DefaultMember</td>
</tr>
<tr>
<td>Descendants</td>
</tr>
<tr>
<td>Distinct</td>
</tr>
<tr>
<td>Dimension</td>
</tr>
<tr>
<td>DrilldownByLayer</td>
</tr>
<tr>
<td>DrilldownMember</td>
</tr>
<tr>
<td>DrillupByLayer</td>
</tr>
<tr>
<td>DrillupMember</td>
</tr>
<tr>
<td>DTS</td>
</tr>
<tr>
<td>EnumText</td>
</tr>
<tr>
<td>EnumValue</td>
</tr>
<tr>
<td>Except</td>
</tr>
<tr>
<td>Exp</td>
</tr>
<tr>
<td>Extract</td>
</tr>
<tr>
<td>Factorial</td>
</tr>
<tr>
<td>Filter</td>
</tr>
<tr>
<td>FirstChild</td>
</tr>
<tr>
<td>FirstSibling</td>
</tr>
<tr>
<td>FormatDate</td>
</tr>
<tr>
<td>Generate</td>
</tr>
</tbody>
</table>

Abs

Returns the absolute value of expression. The absolute value of a number is that number less its sign. A negative number becomes positive, while a positive number remains positive.

Syntax

Abs ( numeric_value_expression )
Parameters

**numeric_value_expression**
Numeric value expression (see MDX Grammar Rules).

Example

The following example is based on the Demo Basic database. The absolute value is taken in case Variance is a negative number. Absolute Variance is always a non-negative number.

The following query:

```sql
WITH MEMBER [Scenario].[Absolute Variance]
AS 'Abs([Scenario].[Actual] - [Scenario].[Budget])'
SELECT
  { [Year].[Qtr1].children }
ON COLUMNS,
  { [Scenario].children, [Scenario].[Absolute Variance] }
ON ROWS
FROM Demo.Basic
WHERE
  ([Accounts].[Sales], [Product].[VCR], [Market].[San_Francisco])
```

returns the grid:

**Table 6-35  Output Grid from MDX Example**

<table>
<thead>
<tr>
<th>(axis)</th>
<th>Jan</th>
<th>Feb</th>
<th>Mar</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actual</td>
<td>1323</td>
<td>1290</td>
<td>1234</td>
</tr>
<tr>
<td>Budget</td>
<td>1200</td>
<td>1100</td>
<td>1100</td>
</tr>
<tr>
<td>Variance</td>
<td>123</td>
<td>190</td>
<td>134</td>
</tr>
<tr>
<td>Absolute Variance</td>
<td>123</td>
<td>190</td>
<td>134</td>
</tr>
</tbody>
</table>

Aggregate

Aggregates the Accounts member based on its Time Balance behavior.

Syntax

```
Aggregate ( set [, accounts_member] )
```

Parameters

**set**
A set containing tuples to be aggregated. If empty, #Missing is returned.
accounts_member
A member from an Accounts dimension. If omitted, the current member from Accounts is used. If there is no Accounts dimension, this function behaves the same as Sum.

Notes
For optimized performance of this function on aggregate storage databases, include in your query the following kinds of sets:

• Any of the following functions, used within the named set and/or as an argument to this function: Intersect, CurrentMember, Distinct, CrossJoin, PeriodsToDate.
• The Filter function, with the search condition defined as:
  dimensionName.CurrentMember IS memberName.
• The IIF function, with the true_part and false_part being sets that meet the above criteria.
• The use of any other functions (such as Members) disables the optimization.
• The second parameter, numeric_value_expression, must be included for optimal performance.

Optimal query performance may require a larger formula cache size. If you get an error message similar to the following, adjust the MAXFORMULACACHESIZE configuration setting accordingly:

Not enough memory for formula execution. Set MAXFORMULACACHESIZE configuration parameter to [1072]KB and try again.

For each tuple in set, the value of accounts_member is evaluated.

If accounts_member has no time balance tag, or if set is one-dimensional, this function behaves the same as Sum().

If accounts_member has a time balance tag, this function behaves as follows:

• For TB First, returns the value of accounts_member for the first tuple in set.
• For TB First with SKIP, scans tuples in set from first to last and returns first tuple with non-empty value for accounts_member.
• For TB Last, returns the value of accounts_member for the last tuple in set.
• For TB Last with SKIP, scans tuples in set from last to first and returns first tuple with non-empty value for accounts_member.
• For TB Average, returns the average of values of accounts_member at each tuple in set.
• For TB Average with SKIP, returns the average of value of accounts_member at each tuple in set without factoring empty values.

Example

WITH
  SET [T1] AS '([Time].[1st Half])'
MEMBER [Measures].[m1] as 'Aggregate(CrossJoin([T1],
  {[Geography].CurrentMember})),[Measures].[Price Paid])'
SELECT {
    [Measures].[m1]
} ON COLUMNS,
    NON EMPTY {CrossJoin([T1], [GM])} ON ROWS
FROM ASOSamp.Sample

returns the grid:

Table 6-36  Output Grid from MDX Example

<table>
<thead>
<tr>
<th>(axis)</th>
<th>m1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1st Half, DISTRICT OF COLUMBIA)</td>
<td>961107.26</td>
</tr>
<tr>
<td>(1st Half, DELAWARE)</td>
<td>245394.68</td>
</tr>
<tr>
<td>(1st Half, FLORIDA)</td>
<td>1446868.96</td>
</tr>
<tr>
<td>(1st Half, GEORGIA)</td>
<td>4766285.74</td>
</tr>
<tr>
<td>(1st Half, MARYLAND)</td>
<td>2496467.86</td>
</tr>
<tr>
<td>(1st Half, NORTH CAROLINA)</td>
<td>4660670.94</td>
</tr>
<tr>
<td>(1st Half, SOUTH CAROLINA)</td>
<td>2524777.6</td>
</tr>
<tr>
<td>(1st Half, VIRGINIA)</td>
<td>6253779.5</td>
</tr>
<tr>
<td>(1st Half, WEST VIRGINIA)</td>
<td>5009523.72</td>
</tr>
</tbody>
</table>

See Also

Sum

Ancestor

Given the input member, this function returns an ancestor at the specified layer.

Syntax

Ancestor ( member, layer | index [, hierarchy ] )

Parameters

member
The member for which an ancestor is sought.

layer
Layer specification.

index
A number of hierarchical steps up from member, locating the ancestor you want returned.

hierarchy
Optional. A specific hierarchy within the time dimension.
Notes

- The return value of this function is a member. If you want the return value to be a set, use Ancestors.
- Do not use negative numbers for index. If you want to return lower members, use Descendants instead of Ancestor. Ancestor([Qtr1], -1) would return an empty member, not a descendant.
- If you use layer to specify a level but no ancestor exists at that level, then the return value is an empty member. For example, in the Sample Basic database, consider the level numbers of the ancestors of the member [Additions] in the [Measures] dimension:

```
[Measures]
+---Profit (+)
    +---Margin (+)
          +---Sales
          +---COGS
    +---Total Expenses (-)
          +---Marketing
          +---Payroll
          +---Misc
+---Inventory (~)
    +---Opening Inventory
    +---Additions
    +---Ending Inventory
+---Ratios (~)
    +---Margin %
    +---Profit %
    +---Profit per Ounce
```

- [Additions], being a leaf-level member, has level number 0.
- [Inventory] has level number 1.
- [Measures] has level number 3, as one of its children [Profit] has level number 2.

The level number of a member = (highest level number among its children) + 1. Therefore, Ancestor ([Measures].[Additions], [Measures].Levels(2)) returns an empty member, because [Additions] does not have an ancestor with level number 2.

Example

```
Ancestor ( [New York], [Market].levels(2) )
```

returns the member [Market], which is the ancestor of [New York] that is located at level 2 in the outline.

```
Ancestor ([Year].[Jan], [Year].generations(2))
```
returns the member [Qtr1], which is the ancestor of Jan that is located in the second
generation of the Year dimension.

Ancestor ( [Feb], 2 )

returns the member [Year], which is the grandparent of Feb.

Ancestor ( [Feb], 0 )

returns the member [Feb]. An "ancestor" that is zero steps away is considered to be
the member itself.

Ancestors

Given the input member and a layer or distance, this function returns a set of
ancestors along with the input member.

When the layer specification is a level, this function returns all ancestors having a level
no greater than the input level. For example, Ancestors ([Additions],
[Measures].Levels(2)) returns { [Inventory], [Additions] }.

Syntax

Ancestors ( member, layer | index )

Parameters

member
The member for which a set of ancestors is sought.

layer
Layer specification.

index
A number of hierarchical steps up from member, locating the highest ancestor you
want returned in the result set.

Notes

• Do not use negative numbers for index. If you want to return lower members, use
Descendants instead of Ancestors. Ancestors ([Qtr1], -1) would return an
empty member, not a descendant.

• If you use layer to specify a level but no ancestors exist at that level, then the
return value is an empty member.

Example

Ancestors ( [New York], [Market].levels(2) )
returns \([\text{Market}], [\text{East}], [\text{New York}]\), the self-inclusive set of \([\text{New York}]\)
ancestors beginning with the ancestor that is located at level 2 of the Market
dimension.

Ancestors \( ([\text{Feb}], 1) \)

returns \([\{\text{Qtr1}\}, [\text{Feb}]\})\), the self-inclusive set of ancestors beginning with the ancestor
one step higher than \([\text{Feb}]\).

Ancestors \( ([\text{Feb}], 0) \)

returns \([\{\text{Feb}\}]\).

Using the ASOSamp.Sample database,

Ancestors \( ([94089], [\text{Geography}].\text{generations}(2)) \)

returns \([\{\text{West}\}, [\text{CA}], [\text{SUNNYVALE - CA}], [94089]\})\), the self-inclusive set of \(94089\)
ancestors beginning with the second generation of the Geography dimension.

**Attribute**

Returns all base members that are associated with a specified attribute member.

**Syntax**

\[
\text{Attribute ( member )}
\]

**Parameters**

*member*

Specification of a member from an attribute dimension.

**Example**

The following query

\[
\text{SELECT}
\{(\text{[Year].Children})
\ON\ \text{COLUMNS},
\text{Attribute ([Ounces_12])}
\ON\ \text{ROWS}
\FROM\ \text{Sample.Basic}
\]

returns the grid:

<table>
<thead>
<tr>
<th>(axis)</th>
<th>Qtr1</th>
<th>Qtr2</th>
<th>Qtr3</th>
<th>Qtr4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cola</td>
<td>5096</td>
<td>5892</td>
<td>6583</td>
<td>5206</td>
</tr>
<tr>
<td>Diet Cola</td>
<td>1359</td>
<td>1534</td>
<td>1528</td>
<td>1287</td>
</tr>
</tbody>
</table>
Table 6-37  (Cont.) Output Grid from MDX Example

<table>
<thead>
<tr>
<th>(axis)</th>
<th>Qtr1</th>
<th>Qtr2</th>
<th>Qtr3</th>
<th>Qtr4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Old Fashioned</td>
<td>1697</td>
<td>1734</td>
<td>1883</td>
<td>1887</td>
</tr>
<tr>
<td>Sarsaparilla</td>
<td>1153</td>
<td>1231</td>
<td>1159</td>
<td>1093</td>
</tr>
<tr>
<td>Diet Cream</td>
<td>2695</td>
<td>2723</td>
<td>2855</td>
<td>2820</td>
</tr>
</tbody>
</table>

See Also

WithAttr

AttributeEx

Returns the set of base members that are associated with a specified varying attribute member or dimension, given the perspective setting.

Syntax

AttributeEx ( member|dimension, ANY, tuple|member[,tuple|member] )

Parameters

member
Specification of a member from an attribute dimension.

dimension
Specification of an attribute dimension.

ANY
The keyword ANY.

tuple | member
Level 0 start tuple (or member) of the independent dimension set. The tuple must contain all the discrete dimensions followed by the continuous dimension members, in the same order that the continuous range has been defined.

tuple | member
Optional level 0 end tuple (or member) of the independent dimension set. The tuple must contain all the discrete dimensions followed by the continuous dimension members, in the same order that the continuous range has been defined.

Example

Consider the following scenario: Products are packaged under different ounces over time and the market state, according to the marketing strategy of the company. Ounces is defined as a varying attribute for the Product dimension, to capture the varying attribute association over the continuous Year dimension and the discrete Market dimension.

Year and Market are the independent dimensions, and level-0 tuple months (for example, Jan) combined with a market state (for example, California) is a perspective for which the varying attribute association is defined.
The following query analyzes the Ounces_32 sales performance of products packaged as Ounces_32 any time from Jul to Dec in New York over all quarters. This is the reality view, which gives the most current view of metrics as they happened over time.

WITH PERSPECTIVE REALITY for Ounces
SELECT
  { Qtr1, Qtr2, Qtr3, Qtr4}
ON COLUMNS,
  {AttributeEx(Ounces_32, ANY, ([New York], Jul), ([New York], Dec))}
ON ROWS
FROM
  app.db
WHERE
  (Sales, [New York], Ounces_32);

See Also
WithAttrEx

Avg

Returns the average of values found in the tuples of a set.

Syntax

Avg ( set [,numeric_value_expression [,IncludeEmpty ] ])

Parameters

set
Set specification.

numeric_value_expression
Numeric value expression (see MDX Grammar Rules). Avg() sums the numeric value expression and then takes the average.

IncludeEmpty
Use this keyword if you want to include in the average any tuples with #MISSING values. Otherwise, they are omitted by default.

Notes

The average is calculated as (sum over the tuples in the set of numeric_value_expr) / count, where count is the number of tuples in the set. Tuples with missing values are not included in count unless IncludeEmpty is specified.

The return value of Avg is #MISSING if either of the following is true:

• The input set is empty.
• All tuple evaluations result in #MISSING values.

Example

Empty Values Included in Calculation of the Average
The following query

WITH MEMBER
    [Market].[Western Avg]
AS
'TAvg ( [Market].[California]:[Market].[Nevada], [Measures].[Sales],
INCLUDEEMPTY)'
SELECT
    { [Product].[Colas].children }
ON COLUMNS,
    { [Market].[West].children, [Market].[Western Avg] }
ON ROWS
FROM
    Sample.Basic
WHERE
    ([Measures].[Sales], [Year].[Jan], [Scenario].[Actual])

returns the grid:

Table 6-38  Output Grid from MDX Example

<table>
<thead>
<tr>
<th>(axis)</th>
<th>Cola</th>
<th>Diet Cola</th>
<th>Caffeine Free Cola</th>
</tr>
</thead>
<tbody>
<tr>
<td>California</td>
<td>678</td>
<td>118</td>
<td>145</td>
</tr>
<tr>
<td>Oregon</td>
<td>160</td>
<td>140</td>
<td>150</td>
</tr>
<tr>
<td>Washington</td>
<td>130</td>
<td>190</td>
<td>#Missing</td>
</tr>
<tr>
<td>Utah</td>
<td>130</td>
<td>190</td>
<td>170</td>
</tr>
<tr>
<td>Nevada</td>
<td>76</td>
<td>62</td>
<td>#Missing</td>
</tr>
<tr>
<td>Western Avg</td>
<td>234.8</td>
<td>140</td>
<td>93</td>
</tr>
</tbody>
</table>

Western Avg for Caffeine Free Cola is 93 because the sales for all Western states is divided by 5, the number of states.

Empty Values Not Included in Calculation of the Average

The following query is the same as the above query, except that it does not use IncludeEmpty:

WITH MEMBER
    [Market].[Western Avg]
AS
'TAvg ( [Market].[California]:[Market].[Nevada], [Measures].[Sales])'
SELECT
    { [Product].[Colas].children }
ON COLUMNS,
    { [Market].[West].children, [Market].[Western Avg] }
ON ROWS
FROM
    Sample.Basic
WHERE
    ([Measures].[Sales], [Year].[Jan], [Scenario].[Actual])

returning the grid:
Table 6-39  Output Grid from MDX Example

<table>
<thead>
<tr>
<th>(axis)</th>
<th>Cola</th>
<th>Diet Cola</th>
<th>Caffeine Free Cola</th>
</tr>
</thead>
<tbody>
<tr>
<td>California</td>
<td>678</td>
<td>118</td>
<td>145</td>
</tr>
<tr>
<td>Oregon</td>
<td>160</td>
<td>140</td>
<td>150</td>
</tr>
<tr>
<td>Washington</td>
<td>130</td>
<td>190</td>
<td>#Missing</td>
</tr>
<tr>
<td>Utah</td>
<td>130</td>
<td>190</td>
<td>170</td>
</tr>
<tr>
<td>Nevada</td>
<td>76</td>
<td>62</td>
<td>#Missing</td>
</tr>
<tr>
<td>Western Avg</td>
<td>234.8</td>
<td>140</td>
<td>155</td>
</tr>
</tbody>
</table>

Western Avg for Caffeine Free Cola is 155 because the sales for all Western states is divided by 3, the number of states that do not have empty values for Caffeine Free Cola.

**BottomCount**

Returns a set of $n$ elements ordered from smallest to largest, optionally based on an evaluation.

This function ignores tuples that resulted in missing values after evaluating numeric value expression.

**Syntax**

```
BottomCount ( set, index [,numeric_value_expression] )
```

**Parameters**

- **set**
  The set from which the bottom $n$ elements are selected.

- **index**
  The number of elements to be included in the set ($n$).

- **numeric_value_expression**
  Optional. An expression further defining the selection criteria (see MDX Grammar Rules).

**Example**

The following expression

```
Bottomcount ( [Product].levels(0).members, 10, ( [Sales], [Actual] ) )
```

returns the set:

```
{ [200-40], [100-30], [400-30], [300-20], [200-30], [100-20], [100-20], [400-20], [400-10], [300-30] }
```
Therefore, the following query

```mdx
SELECT {[Year].levels(1).members} ON COLUMNS,
   BottomCount ( [Product].levels(0).members, 10, ( [Sales], [Actual] ) )
ON ROWS
FROM Sample.Basic
WHERE ( [Sales], [Actual] )
```

returns the grid:

Table 6-40  Output Grid from MDX Example

<table>
<thead>
<tr>
<th>(axis)</th>
<th>Qtr1</th>
<th>Qtr2</th>
<th>Qtr3</th>
<th>Qtr4</th>
</tr>
</thead>
<tbody>
<tr>
<td>200–40</td>
<td>2807</td>
<td>2922</td>
<td>2756</td>
<td>3265</td>
</tr>
<tr>
<td>100–30</td>
<td>3187</td>
<td>3182</td>
<td>3189</td>
<td>3283</td>
</tr>
<tr>
<td>400–30</td>
<td>3763</td>
<td>3962</td>
<td>3995</td>
<td>4041</td>
</tr>
<tr>
<td>300–20</td>
<td>4248</td>
<td>4638</td>
<td>4556</td>
<td>4038</td>
</tr>
<tr>
<td>200–30</td>
<td>4440</td>
<td>4562</td>
<td>4362</td>
<td>4195</td>
</tr>
<tr>
<td>100–20</td>
<td>7276</td>
<td>7957</td>
<td>8057</td>
<td>7179</td>
</tr>
<tr>
<td>100–10</td>
<td>7276</td>
<td>7957</td>
<td>8057</td>
<td>7179</td>
</tr>
<tr>
<td>400–20</td>
<td>7771</td>
<td>8332</td>
<td>8557</td>
<td>8010</td>
</tr>
<tr>
<td>400–10</td>
<td>8614</td>
<td>9061</td>
<td>9527</td>
<td>8957</td>
</tr>
<tr>
<td>300–30</td>
<td>8969</td>
<td>9105</td>
<td>9553</td>
<td>9342</td>
</tr>
</tbody>
</table>

See Also

TopCount

BottomPercent

Returns the smallest possible subset of a set for which the total results of a numeric evaluation are at least a given percentage. The result set is returned with elements listed from smallest to largest.

Syntax

```mdx
BottomPercent ( set, percentage, numeric_value_expression )
```

Parameters

- **set**
  The set from which the bottom-percentile elements are selected.

- **percentage**
  The percentile. This argument must be a value between 0 and 100.

- **numeric_value_expression**
  The expression that defines the selection criteria (see MDX Grammar Rules).

Notes

This function ignores negative and missing values.
Example

The following query returns data for products making up the lowest 5th percentile of all product sales in the Sample Basic database.

WITH
  SET [Lowest 5% products] AS
  'BottomPercent (
    { [Product].members },
    5,
    ([Measures].[Sales], [Year].[Qtr2])
  )'

MEMBER [Product].[Sum of all lowest prods] AS
  'Sum ( [Lowest 5% products] )'

MEMBER [Product].[Percent that lowest sellers hold of all product sales] AS
  'Sum ( [Lowest 5% products] ) / [Product] '

SELECT
  { [Year].[Qtr2].children}
on columns,
  {
    [Lowest 5% products],
    [Product].[Sum of all lowest prods],
    [Product],
    [Product].[Percent that lowest sellers hold of all product sales]
  }
on rows
FROM Sample.Basic
WHERE ([Measures].[Sales])

In the WITH section,

- The named set [Lowest 5% products] consists of those products accounting for the lowest 5 percent of sales in the second quarter. This set includes Birch Beer, Caffeine Free Cola, Strawberry, Sasparilla, and Vanilla Cream.
- The first calculated member, [Product].[Sum of all lowest prods], is used to show the sum of the sales of the products with sales in the lowest fifth percentile.
- The second calculated member, [Product].[Percent that lowest sellers hold of all product sales], is used to show, for each month, how the sales of lowest-selling products compare (as a percentage) to sales of all products in the Product dimension.

This query returns the following grid:

<table>
<thead>
<tr>
<th>Table 6-41</th>
<th>Output Grid from MDX Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>(axis)</td>
<td>Apr</td>
</tr>
<tr>
<td>Birch Beer</td>
<td>954</td>
</tr>
<tr>
<td>Caffeine Free Cola</td>
<td>1049</td>
</tr>
</tbody>
</table>
Table 6-41  (Cont.) Output Grid from MDX Example

<table>
<thead>
<tr>
<th>(axis)</th>
<th>Apr</th>
<th>May</th>
<th>Jun</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strawberry</td>
<td>1314</td>
<td>1332</td>
<td>1316</td>
</tr>
<tr>
<td>Sarsaparilla</td>
<td>1509</td>
<td>1552</td>
<td>1501</td>
</tr>
<tr>
<td>Vanilla Cream</td>
<td>1493</td>
<td>1533</td>
<td>1612</td>
</tr>
<tr>
<td>Sum of all lowest prods</td>
<td>6319</td>
<td>6399</td>
<td>6548</td>
</tr>
<tr>
<td>Product</td>
<td>32917</td>
<td>33674</td>
<td>35088</td>
</tr>
<tr>
<td>Percent that lowest sellers hold of all product sales</td>
<td>0.192</td>
<td>0.194</td>
<td>0.187</td>
</tr>
</tbody>
</table>

See Also

TopPercent

**BottomSum**

Returns the smallest possible subset of a set for which the total results of a numeric evaluation are at least a given sum. Elements of the result set are listed from smallest to largest.

**Syntax**

```
BottomSum ( set, numeric_value_expression, numeric_value_expression )
```

**Parameters**

- **set**
  The set from which the lowest-summing elements are selected.

- **numeric_value_expression1**
  The given sum (see MDX Grammar Rules).

- **numeric_value_expression2**
  The numeric evaluation (see MDX Grammar Rules).

**Notes**

- If the total results of the numeric evaluation do not add up to the given sum, an empty set is returned.
- This function ignores negative and missing values.

**Example**

The following query selects Qtr1 and Qtr2 sales for the lowest selling products in Qtr1 (where Sales totals at least 10000).

```
SELECT 
  {[Year].[Qtr1], [Year].[Qtr2]} 
ON COLUMNS,
```
This query returns the grid:

Table 6-42  Output Grid from MDX Example

<table>
<thead>
<tr>
<th>(axis)</th>
<th>Qtr1</th>
<th>Qtr2</th>
</tr>
</thead>
<tbody>
<tr>
<td>200-40</td>
<td>2807</td>
<td>2922</td>
</tr>
<tr>
<td>100-30</td>
<td>3187</td>
<td>3182</td>
</tr>
<tr>
<td>400-30</td>
<td>3763</td>
<td>3962</td>
</tr>
<tr>
<td>300-20</td>
<td>4248</td>
<td>4638</td>
</tr>
</tbody>
</table>

See Also

TopSum

Case

The CASE keyword begins a conditional expression. There are two types of conditional test you can perform using CASE: simple case expression and searched case expression.

Syntax

The simple case expression evaluates case_operand and returns a result based on its value, as specified by WHEN or ELSE clauses. The result of a case expression can be a value expression or a set. If no ELSE clause is specified, and none of the WHEN clauses is matched, an empty value/empty set is returned.

```
CASE
    case_operand
    simple_when_clause...
    [ else_clause ]
END
```

In searched case expression, each WHEN clause specifies a search condition and a result to be returned if that search condition is satisfied. The WHEN clauses are evaluated in the order specified. The result is returned from the first WHEN clause in which the search condition evaluates to TRUE. The result can be a value expression or a set. If no ELSE clause is specified, and none of the search conditions in the WHEN clauses evaluate to TRUE, an empty value/empty set is returned.

```
CASE
    searched_when_clause...
```
Parameters

**case_operand**
An expression to evaluate.

**simple_when_clause**
One or more WHEN/THEN statements. Syntax: WHEN when_operand THEN result

- **when_operand**: A value expression.
- **result**: A numeric value expression, a string value expression, or a set.

**else_clause**
Optional. Syntax:
ELSE numeric_value_expression | set | string_value_expression

**searched_when_clause**
One or more WHEN/THEN statements. Syntax: WHEN search_condition THEN result

- **search_condition**: A value expression.
- **result**: A numeric value expression, a string value expression, or a set.

Example

**Example for Simple Case Expression**

In the following query, the calculated member [Measures].[ProductOunces] is evaluated based on the value of the Ounce attribute for the current member of the Product dimension.

```
WITH MEMBER [Measures].[ProductOunces] AS
  'Case Product.CurrentMember.Ounces
    when 32 then 32
    when 20 then 20
    when 16 then 16
    when 12 then 12
    else 0
  end'
SELECT
  { [Measures].[ProductOunces] } ON COLUMNS,
  [Product].Members } ON ROWS
FROM Sample.Basic
```

This query returns the following result:

**Table 6-43  Output Grid from MDX Example**

<table>
<thead>
<tr>
<th>(axis)</th>
<th>ProductOunces</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product</td>
<td>0</td>
</tr>
<tr>
<td>Colas</td>
<td>0</td>
</tr>
</tbody>
</table>
Example for Searched Case Expression

The following query divides products into different profit categories based on Profit, and returns categories for each product.

```mdx
WITH MEMBER [Measures].[ProfitCategory] AS 'Case
    when Profit > 10000 then 4
    when Profit > 5000  then 3
    when Profit > 3000 then  2
    else 1
end'
SELECT
{ [Measures].[ProfitCategory] } ON COLUMNS,
{ [Product].Members } ON ROWS
FROM Sample.Basic
```

This query returns the following result:

Table 6-44  Output Grid from MDX Example

<table>
<thead>
<tr>
<th>Product</th>
<th>ProfitCategory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Colas</td>
<td>4</td>
</tr>
<tr>
<td>Diet Cola</td>
<td>0</td>
</tr>
<tr>
<td>Diet Root Beer</td>
<td>0</td>
</tr>
<tr>
<td>Diet Cream</td>
<td>0</td>
</tr>
<tr>
<td>Diet Drinks</td>
<td>0</td>
</tr>
<tr>
<td>Diet Cream</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 6-43  (Cont.) Output Grid from MDX Example

<table>
<thead>
<tr>
<th>(axis)</th>
<th>ProductOunces</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cola</td>
<td>12</td>
</tr>
<tr>
<td>Diet Cola</td>
<td>12</td>
</tr>
<tr>
<td>Caffeine Free Cola</td>
<td>16</td>
</tr>
<tr>
<td>Root Beer</td>
<td>0</td>
</tr>
<tr>
<td>Old Fashioned</td>
<td>12</td>
</tr>
<tr>
<td>Diet Root Beer</td>
<td>16</td>
</tr>
<tr>
<td>Sarsaparilla</td>
<td>12</td>
</tr>
<tr>
<td>Birch Beer</td>
<td>16</td>
</tr>
<tr>
<td>Cream Soda</td>
<td>0</td>
</tr>
<tr>
<td>Dark Cream</td>
<td>20</td>
</tr>
<tr>
<td>Vanilla Cream</td>
<td>20</td>
</tr>
<tr>
<td>Diet Cream</td>
<td>12</td>
</tr>
<tr>
<td>Fruit Soda</td>
<td>0</td>
</tr>
<tr>
<td>Grape</td>
<td>32</td>
</tr>
<tr>
<td>Orange</td>
<td>32</td>
</tr>
<tr>
<td>Strawberry</td>
<td>32</td>
</tr>
<tr>
<td>Diet Drinks</td>
<td>0</td>
</tr>
<tr>
<td>Diet Cola</td>
<td>0</td>
</tr>
<tr>
<td>Diet Root Beer</td>
<td>0</td>
</tr>
<tr>
<td>Diet Cream</td>
<td>0</td>
</tr>
</tbody>
</table>
Table 6-44   (Cont.) Output Grid from MDX Example

<table>
<thead>
<tr>
<th>(axis)</th>
<th>ProfitCategory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cola</td>
<td>4</td>
</tr>
<tr>
<td>Diet Cola</td>
<td>3</td>
</tr>
<tr>
<td>Caffeine Free Cola</td>
<td>1</td>
</tr>
<tr>
<td>Root Beer</td>
<td>4</td>
</tr>
<tr>
<td>Old Fashioned</td>
<td>3</td>
</tr>
<tr>
<td>Diet Root Beer</td>
<td>4</td>
</tr>
<tr>
<td>Sarsaparilla</td>
<td>2</td>
</tr>
<tr>
<td>Birch Beer</td>
<td>2</td>
</tr>
<tr>
<td>Cream Soda</td>
<td>4</td>
</tr>
<tr>
<td>Dark Cream</td>
<td>4</td>
</tr>
<tr>
<td>Vanilla Cream</td>
<td>1</td>
</tr>
<tr>
<td>Diet Cream</td>
<td>4</td>
</tr>
<tr>
<td>Fruit Soda</td>
<td>4</td>
</tr>
<tr>
<td>Grape</td>
<td>4</td>
</tr>
<tr>
<td>Orange</td>
<td>3</td>
</tr>
<tr>
<td>Strawberry</td>
<td>1</td>
</tr>
<tr>
<td>Diet Drinks</td>
<td>4</td>
</tr>
<tr>
<td>Diet Cola</td>
<td>3</td>
</tr>
<tr>
<td>Diet Root Beer</td>
<td>4</td>
</tr>
<tr>
<td>Diet Cream</td>
<td>4</td>
</tr>
</tbody>
</table>

See Also

IIF

CellValue

Returns the numeric value of the current cell.

Syntax

CellValue

Notes

• This function can be useful when defining format strings for a member. Most MDX expressions can be used to specify format strings; however, format strings cannot contain references to values of data cells other than the current cell value being formatted. Use this function to reference the current cell value.

• Enclose all format strings within the MdxFormat( ) directive as shown in the examples.
Example

Example 1

The following format string displays negative values for the current measure if the current [AccountTypes] member is of type “Expense”. CellValue refers to the current cell value that is being formatted. The CurrentMember function in the expression refers to the context of the cell being formatted.

```mdx
/* Display negative values if current Account is an Expense type account */
MdxFORMAT{
  IIF(IsUda(AccountTypes.CurrentMember, "Expense"),
    NumToStr(-CellValue()),
    NumToStr(CellValue()))
}
```

Example 2

The following format string displays negative cell values as positive values enclosed in parentheses.

```mdx
MdxFORMAT{
  IIF(
    CellValue() < 0,
    Concat(Concat("(" , numtostr(-CellValue())), ")" ),
    numtostr(CellValue())
  )
}
```

Example 3

This example illustrates a dynamic member [Variance %] along the [Scenario] dimension. [Variance %] has the following formula, which specifies how to calculate its value from [Actual] and [Budget].

[Variance %] Formula

```
IIF(Is(Measures.CurrentMember, Title) OR
  Is(Measures.CurrentMember, Performance),
  (Actual ± Budget) * 10, (Actual ± Budget)*100/Budget)
```

[Variance %] also has the following format string, which specifies how its values should be displayed. In this case, based on the percentage value computed for a [Variance %] cell, a text value is displayed which conveys the importance of the number.

[Variance %] Format String

```
MdxFORMAT{
  CASE
    WHEN CellValue() <= 5 THEN    "Low"
    WHEN CellValue() > 5 THEN    "High"
  END
}
```
WHEN CellValue() <= 10 THEN "Medium"
WHEN CellValue() <= 15 THEN "High"
ELSE "Very High"
END

Children

Returns a set of all child members of the specified member.

Syntax

\texttt{member}.Children

Parameters

\texttt{member}

A member specification.

Notes

If the input member does not have any children (is a level-0 member), this function returns an empty set.

Example

This example uses the following parts of the Sample Basic outline:

\[\text{West}\]

\[\text{California}\]

\[\text{Oregon}\]

\[\text{Washington}\]

\[\text{Utah}\]

\[\text{Nevada}\]

\[\text{Diet}\]

\[\text{100-20}\]

\[\text{200-20}\]

\[\text{300-30}\]

The following expression

\{(West).children\}

returns the set:

\{(California), (Oregon), (Washington), (Utah), (Nevada)\}

And the following expression

\{(Diet).children\}
returns the set:

{ [100-20], [200-20], [300-30] }

Therefore, the following query

```
SELECT
  {([West].children)}
ON COLUMNS,
  {([Diet].children)}
ON ROWS
FROM Sample.Basic
```

returns the grid:

<table>
<thead>
<tr>
<th>(axis)</th>
<th>California</th>
<th>Oregon</th>
<th>Washington</th>
<th>Utah</th>
<th>Nevada</th>
</tr>
</thead>
<tbody>
<tr>
<td>100-20</td>
<td>-1587</td>
<td>338</td>
<td>231</td>
<td>398</td>
<td>86</td>
</tr>
<tr>
<td>200-20</td>
<td>2685</td>
<td>1086</td>
<td>579</td>
<td>496</td>
<td>167</td>
</tr>
<tr>
<td>300-30</td>
<td>1328</td>
<td>288</td>
<td>1217</td>
<td>413</td>
<td>362</td>
</tr>
</tbody>
</table>

**ClosingPeriod**

Returns the last descendant of a layer, or the last child of the Time dimension.

**Syntax**

```
ClosingPeriod ( [ layer [, member ] ] )
```

**Parameters**

- **layer**
  Layer specification.

- **member**
  Optional member specification. If omitted, the last child of the Time dimension is assumed (for example, Qtr4 in Sample Basic).

**Notes**

The return value of this function varies depending on the input.

1. When both *layer* and *member* arguments are given as input, Closingperiod returns the last descendant of the input member at the input layer. For example,
   Closingperiod(Year.generations(3), Qtr3) returns Sep. If the input *member* and *layer* are the same layer, the output is the input member. For example,
   Closingperiod(Year.generations(3), Sep) returns Sep.

2. When only the *layer* argument is specified, the input member is assumed to be the current member of the dimension used in the layer argument. Closingperiod returns the last descendant of that dimension, at the input layer. For example,
   Closingperiod(Year.generations(3)) returns Dec.
3. When no arguments are specified, the input member is assumed to be the current member of the Time dimension, and ClosingPeriod returns the last child of that member. Do not use this function without arguments if there is no dimension tagged as Time.

Example

The following query

WITH
MEMBER [Measures].[Starting Inventory] AS
' IIF (IsLeaf (Year.CurrentMember),
  [Measures].[Opening Inventory],
  ([Measures].[Opening Inventory],
   OpeningPeriod {
     [Year].Levels(0),
     [Year].CurrentMember
   }
  )
)'

MEMBER [Measures].[Closing Inventory] AS
' IIF (IsLeaf (Year.CurrentMember),
  [Measures].[Ending Inventory],
  ([Measures].[Closing Inventory],
   ClosingPeriod {
     [Year].Levels(0),
     [Year].CurrentMember
   }
  )
)'

SELECT
CrossJoin (
  { [100-10] },
  { [Measures].[Starting Inventory], [Measures].[Closing Inventory] }
)
ON COLUMNS,
Hierarchize ([Year].Members, POST)
ON ROWS
FROM Sample.Basic

returns the grid:

<table>
<thead>
<tr>
<th></th>
<th>100-10</th>
<th>100-10</th>
</tr>
</thead>
<tbody>
<tr>
<td>(axis)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jan</td>
<td>14587</td>
<td>14039</td>
</tr>
<tr>
<td>Feb</td>
<td>14039</td>
<td>13566</td>
</tr>
</tbody>
</table>

Table 6-46  Output Grid from MDX Example
Table 6-46  (Cont.) Output Grid from MDX Example

<table>
<thead>
<tr>
<th>(axis)</th>
<th>Starting Inventory</th>
<th>Closing Inventory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mar</td>
<td>13566</td>
<td>13660</td>
</tr>
<tr>
<td>Qtr1</td>
<td>14587</td>
<td>13660</td>
</tr>
<tr>
<td>Apr</td>
<td>13660</td>
<td>14172</td>
</tr>
<tr>
<td>May</td>
<td>14172</td>
<td>15127</td>
</tr>
<tr>
<td>Jun</td>
<td>15127</td>
<td>15580</td>
</tr>
<tr>
<td>Qtr2</td>
<td>13660</td>
<td>15580</td>
</tr>
<tr>
<td>Jul</td>
<td>15580</td>
<td>14819</td>
</tr>
<tr>
<td>Aug</td>
<td>14819</td>
<td>14055</td>
</tr>
<tr>
<td>Sep</td>
<td>14055</td>
<td>13424</td>
</tr>
<tr>
<td>Qtr3</td>
<td>15580</td>
<td>13424</td>
</tr>
<tr>
<td>Oct</td>
<td>13424</td>
<td>13323</td>
</tr>
<tr>
<td>Nov</td>
<td>13323</td>
<td>13460</td>
</tr>
<tr>
<td>Dec</td>
<td>13460</td>
<td>12915</td>
</tr>
<tr>
<td>Qtr4</td>
<td>13424</td>
<td>12915</td>
</tr>
<tr>
<td>Year</td>
<td>14587</td>
<td>12915</td>
</tr>
</tbody>
</table>

See Also

OpeningPeriod
LastPeriods
ParallelPeriod
PeriodsToDate

CoalesceEmpty

Returns the first (from the left) non #Missing value from the given value expressions.

Syntax

CoalesceEmpty( numeric_value_expression1, numeric_value_expression2 )

Parameters

numeric_value_expression1
A numeric value expression (see MDX Grammar Rules).

numeric_value_expression2
A numeric value expression (see MDX Grammar Rules).

Notes

This function returns numeric_value_expression2 if numeric_value_expression1 is #MISSING; otherwise it returns numeric_value_expression1.
Example

CoalesceEmpty([Profit per Ounce], 0)

returns the [Profit per Ounce] value if it is not #MISSING; returns zero otherwise. This can be used inside the Order function to coalesce all #MISSING values to zero, as shown in the next example:

Order([Product].Members, CoalesceEmpty([Profit per Ounce], 0))

Without CoalesceEmpty in the value expression, the Order function would skip all [Product] members with MISSING values for [Profit per Ounce].

See Also

Order

Concat

Returns the concatenated input strings.

Syntax

Concat ( string [, string +] )

Parameters

string
A string.

string +
Optional. A second string, or a list of multiple additional strings. If omitted, this function returns the single input string.

Example

Concat("01", "01")

Contains

Returns TRUE if a tuple is found within a set; otherwise returns FALSE.

Syntax

Contains ( member_or_tuple, set )

Parameters

member_or_tuple
A member or a tuple.
set
The set to search.

Example
The following expression returns TRUE.

Contains([Oregon], {[California], [Oregon]})

Count
Returns the number of tuples in a set (the cardinality of the set). This function counts all tuples of the set regardless of empty values. If you wish to count only tuples that evaluate to nonempty values, use NonEmptyCount.

Syntax

Count ( set [, IncludeEmpty] )

Parameters

set
The set for which a tuple count is needed.

IncludeEmpty
Optional and default (empty values are counted even if this keyword is omitted).

Notes
This function returns a zero if the input set is empty.

Example

WITH MEMBER
[Measures].[Prod Count]
AS
'Count (Crossjoin {
   {[Measures].[Sales]},
   {[Product].children}
 })'

SELECT
{ [Scenario].[Actual], [Scenario].[Budget] } ON COLUMNS,
{ Crossjoin {
   {[Measures].[Sales]},
   {[Product].children}
 },
   {[Measures].[Prod Count], [Product]}
 } ON ROWS
FROM
WHERE
  ([Year].[Jan], [Market].[New York])
returns the grid:

Table 6-47  Output Grid from MDX Example

<table>
<thead>
<tr>
<th>(axis)</th>
<th>Actual</th>
<th>Budget</th>
</tr>
</thead>
<tbody>
<tr>
<td>Colas</td>
<td>678</td>
<td>640</td>
</tr>
<tr>
<td>Root Beer</td>
<td>551</td>
<td>530</td>
</tr>
<tr>
<td>Cream Soda</td>
<td>663</td>
<td>510</td>
</tr>
<tr>
<td>Fruit Soda</td>
<td>587</td>
<td>620</td>
</tr>
<tr>
<td>Diet Drinks</td>
<td>#Missing</td>
<td>#Missing</td>
</tr>
</tbody>
</table>

Prod Count | Product | 5 | 5 |

The WITH section of the query calculates the count of all products for which a data value exists. The SELECT section arranges the members shown on columns and rows. The entire query is sliced by January and New York in the WHERE section; though those members are not shown in the grid, the data is applicable to those members.

Cousin

Returns a child member at the same position as a member from another ancestor.

Syntax

Cousin ( member1, member2 )

Parameters

member1
A child member. For example, [Year].[Qtr1].

member2
An ancestor for which Cousin() should the return child member at the same position as member1.

Notes

Assuming a symmetric hierarchy, Cousin takes as input one member (member1) from one hierarchy and an ancestor member (member2) of another hierarchy, and returns the child of member2 that is at the same position as member1.

Example

This example uses the following parts of the Sample Basic outline:
The following expression

\{ Cousin ( \{Qtr2\}.\[Apr\], \{Qtr4\} ) \}

returns the member:

\{Qtr4\}.\[Oct\]

And the following expression

\{Product\}.generations(2).members

returns the set:

\{ 100, 200, 300, 400, Diet \}

Therefore, the following query

SELECT
   \{ Cousin ( \{Qtr2\}.\[Apr\], \{Qtr4\} ) \}
ON COLUMNS,
   \{Product\}.generations(2).members
ON ROWS
FROM Sample.Basic

returns the grid:

<table>
<thead>
<tr>
<th>(axis)</th>
<th>Oct</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>2317</td>
</tr>
<tr>
<td>200</td>
<td>2505</td>
</tr>
<tr>
<td>300</td>
<td>2041</td>
</tr>
<tr>
<td>400</td>
<td>1790</td>
</tr>
<tr>
<td>Diet</td>
<td>2379</td>
</tr>
</tbody>
</table>
CrossJoin

Returns the cross-product of two sets from different dimensions.

Syntax

CrossJoin ( set1, set2 )

Parameters

set1
A set to cross with set2.

set2
A set to cross with set1. Must not include any dimension used in set1.

Notes

This function returns the cross-product of two sets from different dimensions. If the two
sets share a common dimension, an error is returned.

If one of the input sets is empty, the output set will be empty as well. For example, the
output will be empty if the input set is [Root Beer].children but [Root Beer] has no
children.

The order of the sets (and their constituent tuples) provided to the CrossJoin function
have an effect on the order of the tuples in the result set. For example,

CrossJoin({a, b}, {c, d})

returns { (a, c), (a, d), (b, c), (b, d) }

CrossJoin({a, b, c}, {d, e, f})

returns { (a, d), (a, e), (a, f), (b, d), (b, e), (b, f), (c, d), (c, e),
          (c, f) }

Be aware of the order of the output set when using the results of CrossJoin with other
order-dependent set functions; for example, Head or Tail.

Example

Example 1
The following expression

CrossJoin({[Qtr1], [Qtr2]}, {[New York], [California]})
returns the set:

\[
\{([Qtr1], \{\text{New York}\}), ([Qtr1], \{\text{California}\}), ([Qtr2], \{\text{New York}\}), ([Qtr2], \{\text{California}\})\}
\]

Therefore, the following query

\[
\text{SELECT} \\
\text{CrossJoin}\{([Qtr1], [Qtr2]), ([\text{New York}], [\text{California}])\} \\
\text{ON COLUMNS} \\
\text{FROM sample.basic}
\]

returns the grid:

Table 6-49  Output Grid from MDX Example

<table>
<thead>
<tr>
<th>Qtr1</th>
<th>Qtr1</th>
<th>Qtr2</th>
<th>Qtr2</th>
</tr>
</thead>
<tbody>
<tr>
<td>New York</td>
<td>California</td>
<td>New York</td>
<td>California</td>
</tr>
<tr>
<td>1656</td>
<td>3129</td>
<td>2363</td>
<td>3288</td>
</tr>
</tbody>
</table>

Example 2

The following expression

\[
\text{CrossJoin}\{([Qtr1], [Qtr2], [Qtr3]), ([\text{New York}], [\text{California}], [\text{Texas}])\}
\]

returns the set

\[
\{([Qtr1], \{\text{New York}\}), ([Qtr1], \{\text{California}\}), ([Qtr1], \{\text{Texas}\}), ([Qtr2], \{\text{New York}\}), ([Qtr2], \{\text{California}\}), ([Qtr2], \{\text{Texas}\}), ([Qtr3], \{\text{New York}\}), ([Qtr3], \{\text{California}\}), ([Qtr3], \{\text{Texas}\})\}
\]

Therefore, the following query

\[
\text{SELECT} \\
\text{CrossJoin}\{([Qtr1], [Qtr2], [Qtr3]), ([\text{New York}], [\text{California}], [\text{Texas}])\} \\
\text{ON AXIS(0)} \\
\text{FROM Sample.Basic}
\]

returns the grid:

Table 6-50  Output Grid from MDX Example

<table>
<thead>
<tr>
<th>Qtr1</th>
<th>Qtr1</th>
<th>Qtr1</th>
<th>Qtr2</th>
<th>Qtr2</th>
<th>Qtr2</th>
<th>Qtr3</th>
<th>Qtr3</th>
<th>Qtr3</th>
</tr>
</thead>
<tbody>
<tr>
<td>New York</td>
<td>California</td>
<td>Texas</td>
<td>New York</td>
<td>California</td>
<td>Texas</td>
<td>New York</td>
<td>California</td>
<td>Texas</td>
</tr>
<tr>
<td>1656</td>
<td>3129</td>
<td>1582</td>
<td>2363</td>
<td>3288</td>
<td>1610</td>
<td>1943</td>
<td>3593</td>
<td>1703</td>
</tr>
</tbody>
</table>
Example 3

The following expression

CrossJoin ([100].children, [Profit].children)

returns the set:

\{([100-10], Margin), ([100-10], [Total Expenses]),
([100-20], Margin), ([100-20], [Total Expenses]),
([100-30], Margin), ([100-30], [Total Expenses])\}

Therefore, the following query

SELECT
  {([Market].levels(1).members)}
ON COLUMNS,
  CrossJoin ([100].children, [Profit].children)
ON ROWS
FROM Sample.Basic

returns the grid:

<table>
<thead>
<tr>
<th>(axis)</th>
<th>East</th>
<th>West</th>
<th>South</th>
<th>Central</th>
</tr>
</thead>
<tbody>
<tr>
<td>100–10</td>
<td>Margin</td>
<td>15762</td>
<td>5937</td>
<td>8124</td>
</tr>
<tr>
<td></td>
<td>Total Expenses</td>
<td>4633</td>
<td>2361</td>
<td>4645</td>
</tr>
<tr>
<td>100–20</td>
<td>Margin</td>
<td>1785</td>
<td>2767</td>
<td>7426</td>
</tr>
<tr>
<td></td>
<td>Total Expenses</td>
<td>671</td>
<td>1570</td>
<td>3495</td>
</tr>
<tr>
<td>100–30</td>
<td>Margin</td>
<td>871</td>
<td>#Missing</td>
<td>3975</td>
</tr>
<tr>
<td></td>
<td>Total Expenses</td>
<td>458</td>
<td>#Missing</td>
<td>1895</td>
</tr>
</tbody>
</table>

See Also

CrossJoinAttribute

CrossJoinAttribute

Returns the cross-product of two sets from different dimensions. This function is similar to CrossJoin, but skips calculation of non-existing intersections. For aggregate storage databases, CrossJoinAttribute can improve on CrossJoin's performance for queries on data intersections, because it checks the validity of data intersections before calculating them. Only valid intersections are calculated, while invalid intersections are set to #MISSING.
Syntax

`CrossJoinAttribute ( set1, set2 )`

Parameters

`set1`  
A set to cross with `set2`.

`set2`  
A set to cross with `set1`. Must not include any dimension used in `set1`.

Notes

In the case of data-less queries, only rows with existing intersections are returned. Data-less queries have the following form:

```sql
SELECT {} ON COLUMNS,  
CrossJoinAttribute (set),{set}) ON ROWS  
FROM <cube_specification>
```

Example

The following query based on ASOSamp.Sample

```sql
SELECT  
{} ON COLUMNS,  
CrossJoinAttribute({[Great Buys].Children}, {[Square Footage].Children} )  
ON ROWS  
FROM ASOSamp.Sample;
```

returns the grid

<table>
<thead>
<tr>
<th>(axis)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(004118, 10000)</td>
</tr>
<tr>
<td>(011683, 5000)</td>
</tr>
<tr>
<td>(017589, 10000)</td>
</tr>
</tbody>
</table>

See Also

`CrossJoin`

`AttributeEx`

`WithAttrEx`

**CurrentAxisMember**

Returns the current axis member in the context of a member value expression argument.
Syntax

CurrentAxisMember()

Notes

This function is intended for use only inside the member value expression argument of the PROPERTY_EXPR function. See MDX Property Expressions.

Example

See the example provided in MDX Property Expressions.

CurrentMember

Returns the current member in the input dimension.

The current member is evaluated in the context of query execution mechanics. Used in conjunction with iterative functions such as Filter, at every stage of iteration the member being operated upon is the current member.

Syntax

\[\text{dimension.CurrentMember}\]

Parameters

dimension

A dimension specification.

Example

The following query selects the quarters during which sales growth is 3% or more compared to the previous month.

\[
\text{SELECT Filter} \left\{ \begin{array}{l}
\text{[Year].Children, -- outer loop} \\
\text{Max (}
\quad \text{Except (}
\quad \quad \text{[Year].CurrentMember.Children, -- current in outer loop} \\
\quad \quad \quad \text{[Year].[Jan]}) \\
\quad \\
\text{)}, \\
\quad \quad \text{[Year].CurrentMember} \\
\quad \quad \quad \text{-- current in Max loop} \\
\quad \quad \quad \quad / \text{[Year].CurrentMember.PrevMember}) \\
\text{)} \quad \text{>= 1.03}
\end{array}\right\}
\]

ON axis(0)
FROM Sample.Basic
WHERE ([Measures].[Sales])
CurrentTuple

Returns the current tuple in a set. Current is in the context of query execution mechanics. Use in combination with iterative functions such as Filter.

Syntax

CurrentTuple ( set )

set.Current

set.CurrentTuple

Parameters

set
A set specification. This argument should be a named set, defined in the WITH section.

Example

The following example finds all Product, Market combinations for which Sales data exists.

WITH SET [NewSet]
SELECT
  Filter([NewSet], NOT IsEmpty([NewSet].CurrentTuple))
ON COLUMNS
FROM Sample.Basic
WHERE
  {[Sales]}

This query returns the following grid:

Table 6-54  Output Grid from MDX Example

<table>
<thead>
<tr>
<th>East</th>
<th>West</th>
<th>South</th>
<th>Central</th>
<th>East</th>
<th>...</th>
<th>Central</th>
<th>East</th>
<th>West</th>
<th>South</th>
<th>Central</th>
</tr>
</thead>
<tbody>
<tr>
<td>27740</td>
<td>28306</td>
<td>16280</td>
<td>33808</td>
<td>23672</td>
<td>...</td>
<td>33451</td>
<td>7919</td>
<td>36423</td>
<td>18676</td>
<td>42660</td>
</tr>
</tbody>
</table>
**DateDiff**

Returns the difference (number) between two input dates in terms of the specified date-parts, following a standard Gregorian calendar.

**Syntax**

DateDiff ( *date1*, *date2*, *date_part* )

**Parameters**

**date1**
A number representing the input date between January 1, 1970 and Dec 31, 2037. The number is the number of seconds elapsed since midnight, January 1, 1970. To retrieve this number, use any of the following functions: Today(), TodateEx(), GetFirstDate(), GetLastDate(), DateRoll(). Date-time attribute properties of a member can also be used to retrieve this number. For example,

- `Product.currentmember.[Intro Date]` returns the product introduction date for the current product in context.
- `[Cola].[Intro Date]` returns the product introduction date for Cola.

**date2**
A second input date. See *date1*.

**date_part**
Defined time components as per the standard calendar.

- DP_YEAR - Year of the input date.
- DP_QUARTER - Quarter of the input date.
- DP_MONTH - Month of the input date.
- DP_WEEK - Week of the input date.
- DP.DAY - Day of the input date.

**Notes**

Based on the input *date_part*, the difference between the two input dates is counted in terms of time component specified.

Example: For input dates June 14, 2005 and Oct 10, 2006,

- DP_YEAR returns the difference in the year component. (2006 - 2005 = 1)
- DP_QUARTER returns the distance between the quarters capturing the input dates. (Quarter 4, 2006 - Quarter 2, 2005 = 6)
- DP_MONTH returns the distance between the months capturing the input dates. (Oct 2006 - June 2005 = 16)
- DP_WEEK returns the distance between the weeks capturing the input dates. Each Standard calendar week is defined to start on Sunday and it spans 7 days. (Oct 10, 2006 - June 14, 2005 = 69)
• DP_DAY returns the difference between the input dates in terms of days. (483 days)

**Example**

The following query returns weekly sales for the last 6 months for the product Cola in the market California.

```mdx
SELECT
  {sales} ON COLUMNS,
  Filter(
    [Time dimension].Weeks.members,
    Datediff(
      GetFirstDate([Time dimension].CurrentMember),
      Today(),
      DP_MONTH
    ) < 6
  )
ON ROWS
FROM Mysamp.Basic
WHERE (Actual, California, Cola);
```

**DatePart**

This function returns the Year/Quarter/Month/Week/Weekday/DayOfYear/Day as a number, given the input date and a date part, following the standard Gregorian calendar.

**Syntax**

```mdx
DatePart ( date, date_part_ex )
```

**Parameters**

**date**

A number representing the input date between January 1, 1970 and Dec 31, 2037. The number is the number of seconds elapsed since midnight, January 1, 1970. To retrieve this number, use any of the following functions: Today(), TodateEx(), GetFirstDate(), GetLastDate(), DateRo

Date-time attribute properties of a member can also be used to retrieve this number. For example,

- Product.currentmember.[Intro Date] returns the product introduction date for the current product in context.
- [Cola].[Intro Date] returns the product introduction date for Cola.

**date_part_ex**

Defined time components as per the standard calendar.

- DP_YEAR - Year of the input date, in yyyy format.
- DP_QUARTER - Quarter of the year (1 to 4) for the input date.
• DP_MONTH - Month of the year (1 to 12) for the input date.
• DP_WEEK - Week of the year for the input date (1 to 54).
• DP_WEEKDAY - Week day of the input date. (1 - Sunday, 2 - Monday, ... 7 - Saturday).
• DP_DAYOFYEAR - Day of the year numbering (1 to 366).
• DP_DAY - Day of the month for the input date (1 to 31).

Notes
Based on the requested time component, the output is as follows:
• DP_YEAR returns the year of the input date in yyyy format.
• DP_QUARTER returns the quarter of the year (1 to 4) for the input date.
• DP_MONTH returns the month of the year (1 to 12) for the input date.
• DP_WEEK returns the week of the year for the input date (1 to 54).
• DP_WEEKDAY returns the week day of the input date. (1 - Sunday, 2 - Monday, ... 7 - Saturday).
• DP_DAYOFYEAR returns the day of the year numbering (1 to 366).
• DP_DAY returns the day of the month for the input date (1 to 31).

Example: For June 14, 2005,
DP_YEAR returns 2005 (the year member, in yyyy format).
DP_QUARTER returns 2 (Second quarter of the year)
DP_MONTH returns 6 (Sixth month of the year)
DP_WEEK returns 24 (24th week of the year)
DP_WEEKDAY returns 4 (for Wednesday. Sunday = 1)
DP_DAYOFYEAR returns 165 (165th day of the year)
DP_DAY returns 14 (14th day of the month)

Example
The following query returns the quarterly sales for the second quarter across all years for the product Cola in the market California.

SELECT
{[Sales]}
  ON COLUMNS,
{
  Filter(
    [Time dimension].Quarters.members,
    Datepart(
      getFirstDate([Time dimension].CurrentMember),
      DP_QUARTER
    ) = 2
  )
}
ON ROWS,
FROM MySamp.Basic
WHERE (Actual, Cola, California);

DateRoll

To the given date, rolls (adds or subtracts) a number of specific time intervals, returning another date. This function assumes a standard Gregorian calendar.

Syntax

DateRoll ( date, date_part, number )

Parameters

date
A number representing the date between January 1, 1970 and Dec 31, 2037. The number is the number of seconds elapsed since midnight, January 1, 1970. To retrieve this number, use any of the following functions: Today(), TodateEx(), GetFirstDate(), GetLastDate(). Date-time attribute properties of a member can also be used to retrieve this number. For example,

• Product.currentmember.[Intro Date] returns the product introduction date for the current product in context.
• [Cola].[Intro Date] returns the product introduction date for Cola.

date_part
Defined time components as per the standard calendar.

• DP_YEAR - Year of the input date.
• DP_QUARTER - Quarter of the input date.
• DP_MONTH - Month of the input date.
• DP_WEEK - Week of the input date.
• DP_DAY - Day of the input date.

number
Number of time intervals to add or subtract.

Notes

Based on input date_part and dateroll number, the date is moved forward or backward in time.

Example: For input date June 14, 2005 and input dateroll number 5,

• DP_YEAR adds 5 years to the input date. (June 14, 2010)
• DP_QUARTER adds 5 quarters to the input date. (June 14, 2005 + 5 quarters = June 14, 2005 + 15 months = Sept 14, 2006)
• DP_MONTH adds 5 months to the input date (June 14, 2005 + 5 months = Nov 14, 2005)
• DP_WEEK adds 5 weeks to the input date (June 14, 2005 + 5 weeks = June 14, 2005 + 35 days = July 19, 2005)
• DP_DAY adds 5 days to the input date. (June 14, 2005 + 5 days = June 19, 2005)

Example
The following query returns actual weekly sales, rolling back for six months from Apr 2005 (inclusive), for the product Cola in the market California.

```
SELECT
{[Sales]}
ON COLUMNS,
(DateToMember

{DateRoll(
   GetFirstDate ([Apr 2005]),
   DP_MONTH,
   6
),
   [Time dimension].Dimension,
   [Time dimension].[WEEKS]
): ClosingPeriod([Time dimension].[Weeks], [Apr 2005]))
} ON ROWS
FROM MySamp.Basic
WHERE (Actual, California, Cola);
```

DateToMember

Returns the date-time dimension member specified by the input date and the input layer.

Syntax

```
DateToMember ( date, dimension [,layer])
```

Parameters

date
A number representing the input date between January 1, 1970 and Dec 31, 2037. The number is the number of seconds elapsed since midnight, January 1, 1970. To retrieve this number, use any of the following functions: Today(), TodateEx(), GetFirstDate(), GetLastDate(), DateRoll(). Date-time attribute properties of a member can also be used to retrieve this number. For example,

• Product.currentmember.[Intro Date] returns the product introduction date for the current product in context.

• [Cola].[Intro Date] returns the product introduction date for Cola.

dimension
A date-time dimension specification.
layer
Optional. A date-time dimension layer specification. If not specified, defaults to the date-time dimension's leaf generation.

Notes
• This function is applicable only to aggregate storage databases.
• This function is only applicable if there is a date-time dimension in the outline.

Example
Consider the following Time-Date dimension hierarchy:

Time dimension (gen 1)
  Years (gen 2)
    Semesters (gen 3)
      Quarters (gen 4)
        Months (gen 5)
          Weeks (gen 6)
            Days (gen 7)

The following query returns sales for the week containing Dec 25, 2006 for the product Cola in the market California.

```
SELECT
  {Sales} ON COLUMNS,
  {
    DateToMember(
      TodateEx("Mon dd yyyy", "December 25 2006"),
      [Time dimension].Dimension,
      [Time dimension].[Weeks])
  } ON ROWS
FROM MySamp.Basic
WHERE (Actual, California, Cola);
```

DefaultMember

Returns the default member in the input dimension. In Essbase, the top member of the input dimension is returned.

Syntax

```
dimension.DefaultMember
```

Parameters

dimension
A dimension specification.
Example

DefaultMember ( [Market] )

returns the member [Market].

DefaultMember ( [Florida].Dimension )

returns the member [Market].

DefaultMember ( [Bottle] )

returns the member [Pkg Type].

Descendants

Returns the set of descendants of a member at a specified level or distance, optionally including or excluding descendants in other levels. The members are returned in hierarchized order; for example, parent members are followed by child members.

Syntax

Descendants ( member , {[ layer | index }[, Desc_flags ]))

Parameters

member
The member for which descendants are sought.

layer
Optional. Layer specification indicating the depth of the descendants to return.

index
Optional. A number of hierarchical steps down from member, locating the descendants you want returned.

Desc_flags
Optional. Keywords which further indicate which members to return. These keywords are available only if layer or index is specified. See Values for Desc_flags

Notes

Values for Desc_flags

For all flags, SELF refers to layer; therefore, BEFORE indicates “before the layer” and AFTER indicates “after the layer.”

• SELF—Include only members in layer, including member only if member is in layer.
• **AFTER**—Include members below *layer*, but not the members of *layer*.

• **BEFORE**—Include *member* and all its descendants that are higher in the hierarchy than *layer*, excluding *layer* and anything below it.

• **BEFORE_AND_AFTER**—Include *member* and all its descendants, down to level 0, but excluding members in *layer*.

• **SELF_AND_AFTER**—Include members in *layer* and all descendants below *layer*.

• **SELF_AND_BEFORE**—Include *member* and all its descendants, down to and including *layer*. 
• SELF BEFORE AFTER—Include member and all its descendants.

• LEAVES—Include only level-0 descendants between member and layer.

Example

The following query

```
SELECT Descendants ( [Year] )
ON COLUMNS
FROM sample.basic
```

returns the grid:

<table>
<thead>
<tr>
<th>Year</th>
<th>Qtr1</th>
<th>Jan</th>
<th>Feb</th>
<th>Mar</th>
<th>Qtr2</th>
<th>Apr</th>
<th>May</th>
<th>Jun</th>
<th>Qtr3</th>
<th>Jul</th>
<th>Aug</th>
<th>Sep</th>
<th>Qtr4</th>
<th>Oct</th>
<th>Nov</th>
<th>Dec</th>
</tr>
</thead>
<tbody>
<tr>
<td>1265</td>
<td>2747</td>
<td>924</td>
<td>888</td>
<td>935</td>
<td>3352</td>
<td>1011</td>
<td>1071</td>
<td>1270</td>
<td>3740</td>
<td>1334</td>
<td>1304</td>
<td>1102</td>
<td>2817</td>
<td>907</td>
<td>884</td>
<td>1026</td>
</tr>
</tbody>
</table>

The following expressions return the following sets

Descendants ( [Year], 2 )

returns {([Jan]:[Dec])}, which is the range of members found two steps below Year.

Descendants ( [Year], 2, BEFORE )

returns {([Year], [Qtr1], [Qtr2], [Qtr3], [Qtr4])}, which is the set of Year and its descendants that occur BEFORE the layer that is two steps below Year.

Descendants ( [Market], [West].level )
returns \{[East], [West], [South], [Central]\}, which is the set of Market's descendants found at the level of West.

Descendants([Market])

is equivalent to Descendants([Market], [Market].level, SELF_BEFORE_AFTER). It returns all descendants of Market:

\{[Market], [East], [New York], [Massachusetts], [Florida], [Connecticut], [New Hampshire], [West], [California], [Oregon], [Washington], [Utah], [Nevada], [South], [Texas], [Oklahoma], [Louisiana], [New Mexico], [Central], [Illinois], [Ohio], [Wisconsin], [Missouri], [Iowa], [Colorado] \}

Descendants([Market], [Region])

is equivalent to Descendants([Market], [Region]), SELF), where [Region] is an alias. It returns all members at [Region] level:

\{[East], [West], [South], [Central]\}

Descendants([Market], [State], SELF)

returns all descendants of [Market] at [State] level:

\{[New York], [Massachusetts], [Florida], [Connecticut], [New Hampshire], [California], [Oregon], [Washington], [Utah], [Nevada], [Texas], [Oklahoma], [Louisiana], [New Mexico], [Illinois], [Ohio], [Wisconsin], [Missouri], [Iowa], [Colorado] \}

Descendants([Market], [State], BEFORE)

returns all regions and [Market]:

\{[Market], [East], [West], [South], [Central]\}

Descendants([Market], [State], AFTER)

returns an empty set, because there are no levels below [State] level in the [Market] dimension:

\{\}

Descendants([Market], [Region], AFTER)
returns all states in the [Market] dimension:

{[New York], [Massachusetts], [Florida], [Connecticut], [New Hampshire],
[California], [Oregon], [Washington], [Utah], [Nevada], [Texas],
[Oklahoma], [Louisiana], [New Mexico], [Illinois], [Ohio], [Wisconsin],
[Missouri], [Iowa], [Colorado]}

Descendants([Market], [State], LEAVES)

returns all level-0 members between [Market] level and [State] level, including both levels:

{[New York], [Massachusetts], [Florida], [Connecticut], [New Hampshire],
[California], [Oregon], [Washington], [Utah], [Nevada], [Texas],
[Oklahoma], [Louisiana], [New Mexico], [Illinois], [Ohio], [Wisconsin],
[Missouri], [Iowa], [Colorado]}

Descendants([Market], 1)

The second argument specifies a distance of 1 from [Market] level, which is [Region] level. So this expression is equivalent to Descendants([Market], [Region]). It returns:

{[East], [West], [South], [Central]}

Descendants([Market], 2, SELF_BEFORE_AFTER)

is equivalent to Descendants([Market], [State], SELF_BEFORE_AFTER). It returns:

{[Market],
[East], [New York], [Massachusetts], [Florida], [Connecticut], [New Hampshire],
[West], [California], [Oregon], [Washington], [Utah], [Nevada],
[South], [Texas], [Oklahoma], [Louisiana], [New Mexico],
[Central], [Illinois], [Ohio], [Wisconsin], [Missouri], [Iowa],
[Colorado]}

Descendants([Market], -1, SELF_BEFORE_AFTER)

prints a warning in application log, because a negative distance argument is not valid. The expression returns an empty set:

{}
returns an empty set, because there are no descendants of [Market] at a distance of 10 from [Market] level.

Descendants([Market], 10, BEFORE)

returns all descendants of [Market]:

{[Market],
 [East], [New York], [Massachusetts], [Florida], [Connecticut], [New Hampshire],
 [West], [California], [Oregon], [Washington], [Utah], [Nevada],
 [South], [Texas], [Oklahoma], [Louisiana], [New Mexico],
 [Central], [Illinois], [Ohio], [Wisconsin], [Missouri], [Iowa],
 [Colorado] }

Descendants([Market], 10, LEAVES)

returns all level-0 descendants of [Market]:

{[New York], [Massachusetts], [Florida], [Connecticut], [New Hampshire],
 [California], [Oregon], [Washington], [Utah], [Nevada], [Texas],
 [Oklahoma], [Louisiana], [New Mexico], [Illinois], [Ohio], [Wisconsin],
 [Missouri], [Iowa], [Colorado] }

Distinct

Deletes duplicate tuples from a set.

**Syntax**

```
Distinct ( set )
```

**Parameters**

- **set**
  The set from which to remove duplicates.

**Notes**

- Duplicates are eliminated from the tail of the set.
- Distinct of an empty set returns an empty set.

**Example**

The expression

```
Distinct(([Colas], [Root Beer], [Cream Soda], [Colas]))
```
returns the set

{[Colas], [Root Beer], [Cream Soda]}

Note that the duplicate [Colas] is removed from the end of the set.

**Dimension**

Returns the dimension that contains the input element.

**Syntax**

```
member.Dimension
```

```
layer.Dimension
```

```
Dimension ( member | layer )
```

**Parameters**

- **member**
  - A member specification. The dimension returned is the dimension that this member belongs to.

- **layer**
  - A layer specification. The dimension returned is the dimension that this layer belongs to.

**Example**

- `[Colas].Dimension` returns `Product`.
- `[Market].[Region].Dimension` returns `Market`.

**DrilldownByLayer**

Drills down members of a set that are at a specified layer.

**Syntax**

```
DrilldownByLayer ( set [, layer | index ] )
```

**Parameters**

- **set**
  - The set in which the drilldown should occur.

- **layer**
  - The layer of the members that should be drilled down.
index
A number of hierarchical steps representing the location of members that should be drilled down.

Notes
This function returns the members of set to one level below the optionally specified layer (or index number of the level). If layer (or index) is omitted, the lowest level of set is returned. Members are returned in their hierarchical order as represented in the database outline.

Example
The following query

```
SELECT
DrilldownByLayer (  
  { ([Product],[California]), ([Product],[Oregon]),  
    ([Product],[New York]), ([Product],[South]),  
    ([Product],[Washington])), [Market].[Region]  
)  
ON COLUMNS  
FROM Sample.Basic
```

returns the grid:

<table>
<thead>
<tr>
<th>Product</th>
<th>California</th>
<th>Oregon</th>
<th>New York</th>
<th>South</th>
<th>Texas</th>
<th>Oklahoma</th>
<th>Louisiana</th>
<th>New Mexico</th>
<th>Washington</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>12964</td>
<td>5062</td>
<td>8202</td>
<td>13238</td>
<td>6425</td>
<td>3491</td>
<td>2992</td>
<td>330</td>
<td>4641</td>
</tr>
</tbody>
</table>

TO use index, note that index is the index number of the dimension to drill down on. In the example below, the function drills down on Market. If you change the 1 to a 0, it drills down on Product.

```
SELECT
DrilldownByLayer (  
  { ([Product],[East]), ([Product],[West])  
  )  
)  
ON COLUMNS  
FROM Sample.Basic
```

DrilldownMember
Drills down on any members or tuples of set1 that are also found in set2. The resulting set contains the drilled-down members or tuples, as well as the original members or tuples (whether they were expanded or not).
Syntax

DrilldownMember( set1, set2 [, RECURSIVE] )

Parameters

set1
The set containing members or tuples to drill down on if comparison with set2 tests positive for identical members or tuples.

set2
The set to compare with set1 before drilling down on members or tuples in set1.

RECURSIVE
Optional. A keyword to enable repeated comparisons of the sets.

Notes
This function drills down on all members of set1 that are also found in set2. The two sets are compared. Then the members or tuples of the first set that are also present in the second set are expanded to include their children.

If the first set is a list of tuples, then any tuples in the first set that contain members from the second set are expanded to their children, generating more tuples.

If the RECURSIVE keyword is used, multiple passes are made on the expanded result sets. Drilldownmember repeats the set comparison and resulting drilldown until there are no more unexpanded members or tuples of set1 that are also present in set2.

Example

Drilling Down on Members
The following examples drill down on members.

Example 1

Example 2
The following expression

DrilldownMember({Market, [New York]}, {Market, West}, RECURSIVE)

returns the set:

{Market, East, West, California, Oregon, Washington, Utah, Nevada, South, Central, [New York]}

The member Market is drilled down and then the West member of the resulting set is drilled down, because the RECURSIVE parameter was specified.

Drilling Down on Tuples
This example uses the following part of the Sample Basic outline:
The following example drills down on tuples.

The following expression

\[
\text{DrilldownMember} \left( \left\{ \left( [100], \text{California} \right), \left( [200], \text{Washington} \right) \right\}, \left\{ [100] \right\} \right)
\]

returns the set of tuples:

\[
\left\{ \left( [100], \text{California} \right), \left( [100-10], \text{California} \right), \left( [100-20], \text{California} \right), \left( [100-30], \text{California} \right), \left( [200], \text{Washington} \right) \right\}
\]

Therefore, the following query

\[
\text{SELECT} \\text{DrilldownMember} \left( \left\{ \left( [100], \text{California} \right), \left( [200], \text{Washington} \right) \right\}, \left\{ [100] \right\} \right) \\text{ON COLUMNS} \\text{FROM Sample.Basic}
\]

returns the grid:

<table>
<thead>
<tr>
<th></th>
<th>100</th>
<th>100-10</th>
<th>100-20</th>
<th>100-30</th>
<th>200</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>California</td>
<td>California</td>
<td>California</td>
<td>California</td>
<td>Washington</td>
</tr>
<tr>
<td>100</td>
<td>999</td>
<td>3498</td>
<td>-1587</td>
<td>-912</td>
<td>1091</td>
</tr>
</tbody>
</table>

**DrillupByLayer**

Drills up the members of a set that are below a specified layer.

**Syntax**

\[
\text{DrillupByLayer} \left( \text{set} [,\text{layer}] \right)
\]
Parameters

set
The set in which the drill-up should occur.

layer
The layer of the members that should be drilled up. If omitted, the set is drilled up to the second lowest level found in the set.

Notes
DrillupLevel can be used as a synonym for DrillupByLayer.

Example
These examples focus on the following hierarchy from the Sample Basic outline:

Example 1
The following query drills up the members of set to the second generation of the Measures dimension:

```mdx
SELECT
    DrillupByLayer
FROM
    Sample.Basic
    
ON COLUMNS

This query returns the grid:

<table>
<thead>
<tr>
<th>Measures</th>
<th>Profit</th>
</tr>
</thead>
<tbody>
<tr>
<td>105522</td>
<td>105522</td>
</tr>
</tbody>
</table>
Example 2

With no layer specified, the following query drills up the members of set to the second lowest level found in set:

```sql
SELECT
    DrillupByLayer
    (
        {[Measures],[Profit],
        [Margin], [Sales], [COGS]
    }
)
ON COLUMNS
FROM Sample.Basic
```

This query returns the grid:

**Table 6-59  Output Grid from MDX Example**

<table>
<thead>
<tr>
<th>Measures</th>
<th>Profit</th>
<th>Margin</th>
</tr>
</thead>
<tbody>
<tr>
<td>105522</td>
<td>105522</td>
<td>221519</td>
</tr>
</tbody>
</table>

**DrillupMember**

Tests two sets for common ancestors and drills up members of the first set to the level of the ancestors that are present in the second set.

**Syntax**

```sql
DrillupMember ( set1, set2 )
```

**Parameters**

- **set1**
  The set containing members to drill up if comparison with set2 tests positive for identical members or tuples.

- **set2**
  The set to compare with set1 before drilling up members in set1.

**Notes**

This function drills up any members of set1 whose ancestors are found in set2. The level to which members in set1 are drilled up depends on the level of the ancestor found in set2. The resulting set contains the ancestors of the drilled up member at the level found in set2, as well as any members of set1 that were not drilled up.
Example

Example 1

The following example

```
DrillupMember({East, South, West, California, Washington, Oregon},{West})
```

returns the set:

{East, South, West}

The following expression

```
DrillupMember
{
    {East, South, West, California,
        Washington, Oregon, Central, Nevada),
    {West}
}
```

returns the set:

{East, South, West, Central, Nevada}

The member Nevada is not drilled up to member West because another member Central interrupts the chain of West descendants.

Example 2

The following examples use the following part of the Sample Basic outline:

```
Product
    -100 (+)
    -100-10
    -100-20
    -100-30
    -200 (+)
```

The following expression

```
DrillupMember
((Product, [100], [100-10]),
 (Product))
```

returns the set:

{Product}
The following expression

\[
\text{DrillupMember} \\
\quad \left( ((\text{Product}, [100], [100-10]), \right.
\quad \left. ([100]) \right)
\]

returns the set:

\{\text{Product}, [100]\}

DTS

Calculates period-to-date values using built-in Dynamic Time Series functionality on block storage databases.

Syntax

\[\text{DTS} \ (\text{dts-operation-specification}, \ \text{member})\]

Parameters

**dts-operation-specification**

The Dynamic Time Series member for which to return values. Specify one of the following operations:

- HTD—History-to-date
- YTD—Year-to-date
- STD—Season-to-date
- PTD—Period-to-date
- QTD—Quarter-to-date
- MTD—Month-to-date
- WTD—Week-to-date
- DTD—Day-to-date

*Note:*

The operation you use for this parameter must have a corresponding Dynamic Time Series member enabled in the outline.

**member**

Member specification. Must be a level-0 member from the time dimension.

Notes

This function is applicable only to block storage databases.
Example

The following query returns year to date information for Sample Basic.

WITH MEMBER [Year].[QuarterToDate_April] AS 'DTS(QTD,Apr)'
SELECT {[Profit], [Opening Inventory],[Ratios]}
ON COLUMNS,
{[Jan],[Feb],[Mar],[Apr],[QuarterToDate_April]}
ON ROWS
FROM Sample.Basic;

This query returns the grid:

Table 6-60  Output Grid from MDX Example

<table>
<thead>
<tr>
<th>(axis)</th>
<th>Profit</th>
<th>Opening Inventory</th>
<th>Ratios</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jan</td>
<td>8024</td>
<td>117405</td>
<td>55.1017819772972</td>
</tr>
<tr>
<td>Feb</td>
<td>8346</td>
<td>116434</td>
<td>55.3868221647073</td>
</tr>
<tr>
<td>Mar</td>
<td>8333</td>
<td>115558</td>
<td>55.2665073107131</td>
</tr>
<tr>
<td>Apr</td>
<td>8644</td>
<td>119143</td>
<td>55.4181729805268</td>
</tr>
<tr>
<td>QuarterToDate_April</td>
<td>8644</td>
<td>119143</td>
<td>55.4181729805268</td>
</tr>
</tbody>
</table>

EnumText

Returns the text value corresponding to a numeric value in a text list.

Syntax

EnumText (textlistname, numeric_value_expression )

Parameters

textlistname
Name of a text list defined on the outline.

numeric_value_expression
Numeric value expression (see MDX Grammar Rules).

Example

EnumText(CSRatings, 1)

returns “Excellent” if there is a text list named CSRatings containing the text “Excellent” mapped to ID 1. This example returns an empty string if there is no text associated with the given numeric ID.
EnumValue

Returns the internal numeric value for a text value in a text list.

Syntax

EnumValue (enum_string)

Parameters

denum_string
Either textlistname.string_literal or textlistmembername.string_literal, where
• textlistname is the name of a text list defined on the outline
• textlistmembername is the name of a member that has an associated text list
• string_literal is the text value stored in the text list

Example

The following expression shows how EnumValue can be used to filter employees
based on their title, which is stored as a text list in [Measures].[Title].

FILTER([Employee].Levels[0].Members, [Measures].[Title] = EnumValue([Job Titles]."Manager") )

Except

Returns a subset containing the differences between two sets, optionally retaining
duplicates. The two input sets must have identical dimensionality.

Syntax

Except ( set1, set2 [,ALL] )

Parameters

set1
A set to compare with set2.

set2
A set to compare with set1.

ALL
The optional ALL flag retains duplicates. Matching duplicates in set1 and set2 are
eliminated.
Example

```
Except( {{New York}, [California], [Florida], [California]},
        {{Oregon}, [Washington], [California], [Florida]})
```

returns {{New York}}.

```
Except( {{New York}, [California], [Florida], [California]},
        {{Oregon}, [Washington], [California], [Florida]}, ALL)
```

returns {{New York}, [California]}.

The following query returns Actual Sales and Profit numbers for the level-0 markets that are not defined as "Major Market."

```
SELECT
    {[Measures].[Sales], [Measures].[Profit]}
ON COLUMNS,
    Except(
        [Market].Levels(0).Members,
        UDA (Market, "Major Market")
    ) ON ROWS
FROM Sample.Basic
WHERE {{{Year].[Qtr1], [Scenario].[Actual]}}
```

This query returns the grid:

Table 6-61  Output Grid from MDX Example

<table>
<thead>
<tr>
<th></th>
<th>Sales</th>
<th>Profit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connecticut</td>
<td>3472</td>
<td>920</td>
</tr>
<tr>
<td>New Hampshire</td>
<td>1652</td>
<td>202</td>
</tr>
<tr>
<td>Oregon</td>
<td>5058</td>
<td>1277</td>
</tr>
<tr>
<td>Washington</td>
<td>4835</td>
<td>1212</td>
</tr>
<tr>
<td>Utah</td>
<td>4209</td>
<td>744</td>
</tr>
<tr>
<td>Nevada</td>
<td>6516</td>
<td>775</td>
</tr>
<tr>
<td>Oklahoma</td>
<td>2961</td>
<td>718</td>
</tr>
<tr>
<td>Louisiana</td>
<td>2906</td>
<td>773</td>
</tr>
<tr>
<td>New Mexico</td>
<td>1741</td>
<td>4</td>
</tr>
<tr>
<td>Wisconsin</td>
<td>4073</td>
<td>913</td>
</tr>
<tr>
<td>Missouri</td>
<td>3062</td>
<td>399</td>
</tr>
<tr>
<td>Iowa</td>
<td>6175</td>
<td>2036</td>
</tr>
</tbody>
</table>

Exp

Returns the exponent of an expression; that is, the value of e (the base of natural logarithms) raised to the power of the expression.
Syntax

Exp ( numeric_value_expression )

Parameters

numeric_value_expression
A numeric value (see MDX Grammar Rules).

Notes

• Exp returns the inverse of Ln, the natural logarithm.
• The constant e is the base of the natural logarithm. e is approximately 2.71828182845904.

Example

The calculated member Index is created to represent e raised to the power of [Variance %]/100. In the example, [Variance %] divided by 100 is the numeric value expression provided to the Exp function.

WITH MEMBER [Scenario].[Index]
AS
'Exp(' + [Scenario].[Variance %]/100 + ')
SELECT
{[Scenario].[Variance %], [Scenario].[Index]}
ON COLUMNS,
{[Market].children}
ON ROWS
FROM
Sample.Basic
WHERE
{{Sales}}

This query returns the grid:

Table 6-62  Output Grid from MDX Example

<table>
<thead>
<tr>
<th>(axis)</th>
<th>Variance %</th>
<th>Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>East</td>
<td>10.700</td>
<td>1.113</td>
</tr>
<tr>
<td>West</td>
<td>10.914</td>
<td>1.115</td>
</tr>
<tr>
<td>South</td>
<td>3.556</td>
<td>1.036</td>
</tr>
<tr>
<td>Central</td>
<td>3.595</td>
<td>1.037</td>
</tr>
</tbody>
</table>

See Also

Ln
Extract

Returns a set of tuples with members from the specified dimensions of the input set.

Syntax

Extract ( set [ , dimension ... ] )

Parameters

set
The set from which to extract tuples belonging to the specified dimension.

dimension
One or more dimensions from which to extract a set.

Notes

This function always removes duplicates. The dimension argument should specify dimensions present in the input set. It is an error to specify a dimension that is not present in the input set. The members in the tuples of the output set are ordered based on the dimension order specified in the input set.

Example

In the following example, Extract returns a subset of only those tuples belonging to the Year dimension.

SELECT
  Extract( 
    { 
      ([Year].[Qtr1], [Market].[California]),
      ([Year].[Qtr1], [Market].[Oregon]),
      ([Year].[Qtr2], [Market].[Oregon])
    }, Year
  )
ON COLUMNS
FROM Sample.basic

Table 6-63  Output Grid from MDX Example

<table>
<thead>
<tr>
<th>Qtr1</th>
<th>Qtr2</th>
</tr>
</thead>
<tbody>
<tr>
<td>24703</td>
<td>27107</td>
</tr>
</tbody>
</table>

Factorial

Returns the factorial of a number.

Syntax

Factorial ( index )
Parameters

_index_
A numeric value. The fractional part of _index_ is ignored.

Example

Factorial(5) returns 120 (which is 5 * 4 * 3 * 2 * 1).

Factorial(3.5) returns 6 (which is 3 * 2 * 1). The fractional part of _index_ is ignored.

Filter

Returns the tuples of a set that meet the criteria of a search condition.

Syntax

FILTER (set, search_condition)

Parameters

set
The set through which to iterate.

search_condition
A Boolean expression (see MDX Grammar Rules). The search condition is evaluated in the context of every tuple in the set.

Notes

This function returns the subset of tuples in _set_ for which the value of the search condition is TRUE. The order of tuples in the returned set is the same as in the input set.

Example

Example 1

The following _unfiltered_ query returns profit for all level-0 products:

SELECT
  { [Profit] }
ON COLUMNS,
  [Product].levels(0).members
ON ROWS
FROM Sample.Basic

This query returns the grid:

<table>
<thead>
<tr>
<th>(axis)</th>
<th>Profit</th>
</tr>
</thead>
<tbody>
<tr>
<td>100-10</td>
<td>22777</td>
</tr>
</tbody>
</table>
Table 6-64  (Cont.) Output Grid from MDX Example

<table>
<thead>
<tr>
<th>(axis)</th>
<th>Profit</th>
</tr>
</thead>
<tbody>
<tr>
<td>100-20</td>
<td>5708</td>
</tr>
<tr>
<td>100-30</td>
<td>1983</td>
</tr>
<tr>
<td>200-10</td>
<td>7201</td>
</tr>
<tr>
<td>200-20</td>
<td>12025</td>
</tr>
<tr>
<td>200-30</td>
<td>4636</td>
</tr>
<tr>
<td>200-40</td>
<td>4092</td>
</tr>
<tr>
<td>300-10</td>
<td>12195</td>
</tr>
<tr>
<td>300-20</td>
<td>2511</td>
</tr>
<tr>
<td>300-30</td>
<td>11093</td>
</tr>
<tr>
<td>400-10</td>
<td>11844</td>
</tr>
<tr>
<td>400-20</td>
<td>9851</td>
</tr>
<tr>
<td>400-30</td>
<td>-394</td>
</tr>
<tr>
<td>100-20</td>
<td>5708</td>
</tr>
<tr>
<td>200-20</td>
<td>12025</td>
</tr>
<tr>
<td>300-30</td>
<td>11093</td>
</tr>
</tbody>
</table>

To filter the above results to only show negative Profit, use the Filter function, passing it the original set and a search condition. Filter will only return the set of members for which the search condition is true (for which Profit is less than zero).

```
SELECT
    { Profit }
ON COLUMNS,
    Filter( [Product].levels(0).members, Profit < 0)
ON ROWS
FROM Sample.Basic
```

The resulting query returns only the products with negative profit:

Table 6-65  Output Grid from MDX Example

<table>
<thead>
<tr>
<th>(axis)</th>
<th>Profit</th>
</tr>
</thead>
<tbody>
<tr>
<td>400-30</td>
<td>-394</td>
</tr>
</tbody>
</table>

Example 2

The search expression in Example 1 compared a value expression (Profit) with a value. You can also filter using a member attribute as the search condition. For example, you can use the Filter function to only select members whose Caffeinated attribute is TRUE.

```
SELECT
    { [Profit] }
ON COLUMNS,
    Filter( [Product].levels(0).members, Product.CurrentMember.[Caffeinated])
```

The resulting query returns only the products with Caffeinated attribute set to TRUE.
ON ROWS
FROM Sample.Basic

This query returns profit for the members that are caffeinated:

Table 6-66  Output Grid from MDX Example

<table>
<thead>
<tr>
<th>(axis)</th>
<th>Profit</th>
</tr>
</thead>
<tbody>
<tr>
<td>100-10</td>
<td>22777</td>
</tr>
<tr>
<td>100-20</td>
<td>5708</td>
</tr>
<tr>
<td>200-10</td>
<td>7201</td>
</tr>
<tr>
<td>200-20</td>
<td>12025</td>
</tr>
<tr>
<td>300-10</td>
<td>12195</td>
</tr>
<tr>
<td>300-20</td>
<td>2511</td>
</tr>
<tr>
<td>300-30</td>
<td>11093</td>
</tr>
</tbody>
</table>

To understand the search condition, Product.CurrentMember.[Caffeinated], it may be helpful to read it right to left: Filter is searching for presence of the Caffeinated property on the current member, for each member in the input set, which happens to be from the Product dimension (The CurrentMember function requires the dimension name as its argument).

Filter is an iterative function, meaning that at every member or tuple in the set being evaluated, the member being operated upon is the “current member,” until Filter has looped through the entire input set and evaluated the search condition for each tuple. So to see how the previous query results were generated, it would be useful to see which members actually have the Caffeinated attribute set to true. The following unfiltered query uses a calculated member to reveal which of the level-0 product members is caffeinated. The IIF function returns a value of 1 for each member whose Caffeinated attribute is set to TRUE, and returns a value of 0 otherwise.

WITH MEMBER Measures.IsCaffeinated
AS 'IIF(Product.CurrentMember.[Caffeinated], 1, 0)'
SELECT
  { IsCaffeinated }
ON COLUMNS,
  [Product].levels(0).members
ON ROWS
FROM Sample.Basic

This query returns the grid:

Table 6-67  Output Grid from MDX Example

<table>
<thead>
<tr>
<th>(axis)</th>
<th>IsCaffeinated</th>
</tr>
</thead>
<tbody>
<tr>
<td>100-10</td>
<td>1</td>
</tr>
<tr>
<td>100-20</td>
<td>1</td>
</tr>
<tr>
<td>100-30</td>
<td>0</td>
</tr>
<tr>
<td>200-10</td>
<td>1</td>
</tr>
<tr>
<td>200-20</td>
<td>1</td>
</tr>
</tbody>
</table>
Looking at the results for the second query, you can begin to see that the search condition is evaluated for each tuple in the input set, and that only the tuples meeting the search condition are returned.

**Example 3**

Example 2 introduced the `CurrentMember` function. Even when `CurrentMember` is not explicitly called, `Filter` operates in the context of “the current member” while it iterates through a set. `Filter` and other iterative functions are processed in a nested context.

By default, `Filter` operates in the current-member context of top dimension members. You make the MDX context smaller by using a slicer (the `Where` clause), which overrides the built-in top-dimensional context. Additionally, you can override the slicer context by specifying context in the search condition argument for `Filter`.

The following query returns the Profit values for Western Region, for Qtr1. Note that the MDX context is West, Qtr1.

```
SELECT
    { [Profit] }
ON COLUMNS,
    [Product].levels(0).members
ON ROWS
FROM Sample.Basic
WHERE (West, Qtr1)
```

When adding a filter to the above query, the values for Profit are still evaluated as `(Profit, West, Qtr1)`, because the sub-context for `Filter` is based on the main context.

```
SELECT
    { [Profit] }
ON COLUMNS,
    Filter( [Product].levels(0).members, Profit < 0 )
ON ROWS
```
FROM Sample.Basic
Where (West, Qtr1)

In the next query, the values for Profit are evaluated as (Profit, West, Qtr1), even though the outer context is (Profit, Market, Qtr1). This is because the inner context in the Filter function overrides the outer context of the slicer (West replaces Market).

SELECT
   { [Sales] }
ON COLUMNS,
Filter( [Product].levels(0).members, (Profit, West) < 0)
ON ROWS
FROM Sample.Basic
Where (Market, Qtr1)

The above query returns the Sales values for West, Qtr1 for members of Product whose Profit for West, Qtr1 was less than 0.

Table 6-68  Output Grid from MDX Example

<table>
<thead>
<tr>
<th>(axis)</th>
<th>Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>100-20</td>
<td>2153</td>
</tr>
<tr>
<td>400-30</td>
<td>1862</td>
</tr>
<tr>
<td>100-20</td>
<td>2153</td>
</tr>
</tbody>
</table>

Additional Examples

The following query on Sample Basic returns Qtr2 sales figures for products where the sales have increased by at least 10% since Qtr1.

SELECT
   { Filter {
      [Product].Members,
      [Measures].[Sales] >
      1.1 *
      ( [Measures].[Sales], [Year].CurrentMember.PrevMember )
   } }
on columns
FROM sample.basic
WHERE ([Year].[Qtr2], [Measures].[Sales])

Table 6-69  Output Grid from MDX Example

<table>
<thead>
<tr>
<th>Cola</th>
<th>Dark Cream</th>
</tr>
</thead>
<tbody>
<tr>
<td>16048</td>
<td>11993</td>
</tr>
</tbody>
</table>
The following query on Sample Basic returns sales figures for product family "100" where the monthly sales of that product family are greater than 8,570. The filtering logic is stored as a named set in the WITH section.

WITH SET [High-Sales Months] as
  , Filter(
      [Year].Levels(0).members,
      [Measures].[Sales] > 8570
     )
 , SELECT
   { [Measures].[Sales] }
 ON COLUMNS,
   { [High-Sales Months] }
 ON ROWS
 FROM
 sample.basic
 WHERE
 ([Product].[100])

<table>
<thead>
<tr>
<th>(axis)</th>
<th>Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apr</td>
<td>8685</td>
</tr>
<tr>
<td>May</td>
<td>8945</td>
</tr>
<tr>
<td>Jun</td>
<td>9557</td>
</tr>
<tr>
<td>Jul</td>
<td>9913</td>
</tr>
<tr>
<td>Aug</td>
<td>9787</td>
</tr>
<tr>
<td>Sep</td>
<td>8844</td>
</tr>
<tr>
<td>Dec</td>
<td>8772</td>
</tr>
</tbody>
</table>

**FirstChild**

Returns the first child of the input member.

**Syntax**

`member.FirstChild`

**Parameters**

- **member**
  A member specification. If a level-0 member, the output of FirstChild is an empty member.
Example

```mdx
SELECT
  {[Qtr1].firstchild}
ON COLUMNS,
  {[Market].[Central].lastchild}
ON ROWS
FROM Sample.Basic
```

Table 6-71  Output Grid from MDX Example

<table>
<thead>
<tr>
<th>(axis)</th>
<th>Jan</th>
</tr>
</thead>
<tbody>
<tr>
<td>Colorado</td>
<td>585</td>
</tr>
</tbody>
</table>

See Also
- `LastChild`
- `FirstChild`

**FirstChild**

Returns the first child of the input member's parent.

**Syntax**

```mdx
FirstChild ( member [, hierarchy ])
```

```
member.FirstSibling ([hierarchy])
```

**Parameters**

- `member`  
  A member specification.

- `hierarchy`  
  Optional. A specific hierarchy within the time dimension.

**Notes**

If `member` is the top member of a dimension, then `member` itself is returned.

**Example**

**Example 1**

Year.FirstSibling returns Year.

Qtr3.firstsibling returns Qtr1.
Example 2

For every month, the following query displays the change in inventory level since the beginning of the quarter.

WITH MEMBER
[Measures].[Inventory Level since beginning of Quarter]
AS
'([Ending Inventory] - ([Opening Inventory],
[Year].CurrentMember.FirstSibling))'
SELECT
([Measures].[Inventory Level since beginning of Quarter])
ON COLUMNS,
Year.Levels(0).Members ON ROWS
FROM Sample.Basic

This query returns the grid:

Table 6-72  Output Grid from MDX Example

<table>
<thead>
<tr>
<th>(axis)</th>
<th>Inventory Level Since Beginning of Quarter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jan</td>
<td>-971</td>
</tr>
<tr>
<td>Feb</td>
<td>-1847</td>
</tr>
<tr>
<td>Mar</td>
<td>1738</td>
</tr>
<tr>
<td>Apr</td>
<td>6740</td>
</tr>
<tr>
<td>May</td>
<td>17002</td>
</tr>
<tr>
<td>Jun</td>
<td>24315</td>
</tr>
<tr>
<td>Jul</td>
<td>-871</td>
</tr>
<tr>
<td>Aug</td>
<td>-1243</td>
</tr>
<tr>
<td>Sep</td>
<td>-1608</td>
</tr>
<tr>
<td>Oct</td>
<td>2000</td>
</tr>
<tr>
<td>Nov</td>
<td>5308</td>
</tr>
<tr>
<td>Dec</td>
<td>4474</td>
</tr>
</tbody>
</table>

See Also

LastSibling
FirstChild

**FormatDate**

Returns a formatted date-string.

**Syntax**

FormatDate ( date, internal-date-format )
Parameters

date
A number representing the input date between January 1, 1970 and Dec 31, 2037. The number is the number of seconds elapsed since midnight, January 1, 1970. To retrieve this number, use any of the following functions: Today(), TodateEx(), GetFirstDate(), GetLastDate(), DateRoll(). Date-time attribute properties of a member can also be used to retrieve this number. For example,

• `Product.currentmember.[Intro Date]` returns the product introduction date for the current product in context.

• `[Cola].[Intro Date]` returns the product introduction date for Cola.

internal-date-format
One of the following literal strings (excluding ordered-list numbers and parenthetical examples) indicating a supported date format.

1. "mon dd yyyy" *(Example: mon = Aug)*
2. "Month dd yyyy" *(Example: Month = August)*
3. "mm/dd/yy"
4. "mm/dd/yyyy"
5. "yy.mm.dd"
6. "dd/mm/yy"
7. "dd.mm.yy"
8. "dd-mm-yy"
9. "dd Month yy"
10. "dd mon yy"
11. "Month dd, yy"
12. "mon dd, yy"
13. "mm-dd-yy"
14. "yy/mm/dd"
15. "yymmdd"
16. "dd Month yyyy"
17. "dd mon yyyy"
18. "yyyy-mm-dd"
19. "yyyy/mm/dd"
20. "Long format" *(Example: "WeekDay, Mon dd, yyyy")*
21. "Short format" *(Example: "m/d/yy")*

Notes

• Using an invalid input date returns an error.

• Using extra whitespace not included in the internal format strings returns an error.
• This function interprets years in the range 1970 to 2029 for yy format. Therefore, if the function is invoked using a date format mm/dd/yy for June 20, 2006, the returned date string is "06/20/06".

Example
The following query returns the first 10 day sales for all Colas products since their release date in the market California.

WITH MEMBER
  Measures.[first 10 days sales] AS
  'SUM(
    LastPeriods(-10,
      StrToMbr(
        FormatDate("Mon dd yyyy", Product.CurrentMember.[Intro Date])
      )
    )
  )
```
ON COLUMNS,
  {Colas.Children}
ON ROWS
FROM MySamp.basic
WHERE (California, Actual);

Generate

Returns a set formed by evaluating a set expression. For each tuple in set1, return set2.

Syntax

Generate ( set1, set2 [, [ALL]] )

Parameters

set1
The set to loop through.

set2
The set expression to evaluate for every tuple in set1.

ALL
If the optional ALL flag is used, duplicate tuples are retained.

Notes
The set expression set2 is evaluated in the context of each of the tuples from set1. The resulting sets are combined, in the same order as of the tuples in set1, to produce the output. Duplicates are not included by default.
Example

For each region of the market, return its top-selling 3 products. Display the sales data by quarter.

WITH SET [Top3BevsPerRegion]
AS
'Generate {{[Market].children},
    Crossjoin
    {
        {[Market].Currentmember},
        TopCount
        {
            [Product].Members, 3, [Measures].[Sales]
        }
    }
}"
SELECT
    {[Top3BevsPerRegion]}
ON COLUMNS,
    {[Year].children}
ON ROWS
FROM Sample.Basic
WHERE ([Scenario].[Actual], [Measures].[Sales])

Table 6-73  Output Grid from MDX Example

<table>
<thead>
<tr>
<th>(axis)</th>
<th>East</th>
<th>Colas</th>
<th>Root Beer</th>
<th>West</th>
<th>Diet Drinks</th>
<th>Cream Soda</th>
<th>South</th>
<th>Root Beer</th>
<th>Diet Drinks</th>
<th>Produ ct</th>
<th>Diet Drinks</th>
<th>Colas</th>
</tr>
</thead>
<tbody>
<tr>
<td>Qtr1</td>
<td>20621</td>
<td>6292</td>
<td>5726</td>
<td>31674</td>
<td>8820</td>
<td>8043</td>
<td>12113</td>
<td>5354</td>
<td>4483</td>
<td>31412</td>
<td>10544</td>
<td>8074</td>
</tr>
<tr>
<td>Qtr2</td>
<td>224499</td>
<td>7230</td>
<td>5902</td>
<td>33572</td>
<td>9086</td>
<td>8982</td>
<td>12602</td>
<td>5535</td>
<td>4976</td>
<td>33056</td>
<td>10809</td>
<td>8701</td>
</tr>
<tr>
<td>Qtr3</td>
<td>22976</td>
<td>7770</td>
<td>5863</td>
<td>35130</td>
<td>9518</td>
<td>9616</td>
<td>13355</td>
<td>5690</td>
<td>4497</td>
<td>33754</td>
<td>10959</td>
<td>8894</td>
</tr>
<tr>
<td>Qtr4</td>
<td>21352</td>
<td>6448</td>
<td>6181</td>
<td>32555</td>
<td>8999</td>
<td>8750</td>
<td>12776</td>
<td>5429</td>
<td>4450</td>
<td>31458</td>
<td>10348</td>
<td>8139</td>
</tr>
</tbody>
</table>

Generation

Returns the generation of the input member.

Syntax

`member.Generation`

Parameters

`member`
Member specification.
Example

The following query

```
SELECT
  [Year].[Qtr1].Generation.Members
ON COLUMNS,
  [Product].Generations(2).Members
ON ROWS
FROM Sample.Basic
```

returns the grid:

<table>
<thead>
<tr>
<th>(axis)</th>
<th>Qtr1</th>
<th>Qtr2</th>
<th>Qtr3</th>
<th>Qtr4</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>7048</td>
<td>7872</td>
<td>8511</td>
<td>7037</td>
</tr>
<tr>
<td>200</td>
<td>6721</td>
<td>7030</td>
<td>7005</td>
<td>7198</td>
</tr>
<tr>
<td>300</td>
<td>5929</td>
<td>6769</td>
<td>6698</td>
<td>6403</td>
</tr>
<tr>
<td>400</td>
<td>5005</td>
<td>5436</td>
<td>5698</td>
<td>5162</td>
</tr>
<tr>
<td>Diet</td>
<td>7017</td>
<td>7336</td>
<td>7532</td>
<td>6941</td>
</tr>
</tbody>
</table>

See Also

Generations
Level
IsGeneration

Generations

Returns the generation specified by the input generation number.

Syntax

```
dimension.Generations { index }
```

Generations ( dimension, index )

Parameters

- **dimension**  
The dimension specification.

- **index**  
The numerical depth from the top member of the outline, where the top member is 1.
Example

The following query

```sql
SELECT  
    [Year].[Qtr1].Generation.Members
ON COLUMNS,
    [Product].Generations(2).Members
ON ROWS
FROM Sample.Basic
```

returns the grid:

<table>
<thead>
<tr>
<th>(axis)</th>
<th>Qtr1</th>
<th>Qtr2</th>
<th>Qtr3</th>
<th>Qtr4</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>7048</td>
<td>7872</td>
<td>8511</td>
<td>7037</td>
</tr>
<tr>
<td>200</td>
<td>6721</td>
<td>7030</td>
<td>7005</td>
<td>7198</td>
</tr>
<tr>
<td>300</td>
<td>5929</td>
<td>6769</td>
<td>6698</td>
<td>6403</td>
</tr>
<tr>
<td>400</td>
<td>5005</td>
<td>5436</td>
<td>5698</td>
<td>5162</td>
</tr>
<tr>
<td>Diet</td>
<td>7017</td>
<td>7336</td>
<td>7532</td>
<td>6941</td>
</tr>
</tbody>
</table>

See Also

Generation

Levels

GetFirstDate

Returns the start date for a date-time dimension member.

Syntax

```sql
GetFirstDate ( member )
```

Parameters

`member`

A member from a date-time dimension.

Notes

- This function returns #MISSING if the input member is not from a date hierarchy in a Time-Date tagged dimension.
- The return value is a number representing the input date. The number is the number of seconds elapsed since midnight, January 1, 1970.
- This function is applicable only to aggregate storage databases.
Example

The following query returns sales for the first week of April, 2004.

```
SELECT
  {[Sales]}
ON COLUMNS,
  {DateToMember(
    GetFirstDate ([Apr 2004]),
    [Time dimension].Dimension,
    [Time dimension].[Weeks]
  )}
ON ROWS
FROM MySamp.basic;
```

GetFirstDay

For a given `date_part`, this function returns the first day of the time interval for the input date, following a standard Gregorian calendar.

Syntax

```
GetFirstDay ( date, date_part )
```

Parameters

date
A number representing the input date between January 1, 1970 and Dec 31, 2037. The number is the number of seconds elapsed since midnight, January 1, 1970. To retrieve this number, use any of the following functions: Today(), TodateEx(), GetFirstDate(), GetLastDate(), DateRoll().

Date-Time type attribute properties of a member can also be used to retrieve this number. For example: `Product.currentmember.[Intro Date]` returns the Introduction or release date for the current product in context. `[Cola].[Intro Date]` returns the Introduction or release date for the “Cola” product.

date_part
Defined time components of the standard calendar.

• DP_YEAR - year of the input date.
• DP_QUARTER – quarter of the input date.
• DP_MONTH - month of the input date.
• DP_WEEK - week of the input date.

Notes

This function can be used for getting the truncated date of an input date for a given date part, following a standard Gregorian calendar.
Example

Assuming today’s date is April 15 2007, consider the following scenarios.

GetFirstDay(Today(), DP_YEAR)

returns the first day of the year, Jan 1 2007

GetFirstDay(Today(), DP_QUARTER)

returns the first day of the quarter, Apr 1 2007

GetFirstDay(Today(), DP_MONTH)

returns the first day of the month, Apr 1 2007

GetFirstDay(Today(), DP_WEEK)

returns the first day of the week, Apr 15 2007

See Also

GetNextDay
GetLastDay
Today

GetLastDate

Returns the end date for a date-time dimension member.

Syntax

GetLastDate ( member )

Parameters

member
A member from a date-time tagged dimension.

Notes

• This function returns #MISSING if the input member is not from a date hierarchy in a Time-Date tagged dimension.
• The return value is a number representing the input date. The number is the number of seconds elapsed since midnight, January 1, 1970.
• This function is applicable only to aggregate storage databases.
Example
The following query returns sales for the last week of April, 2004.

SELECT
  {[Sales]}
ON COLUMNS,
  {DateToMember(
    GetLastDate ([Apr 2004]),
    [Time dimension].Dimension,
    [Time dimension].[Weeks]
  )}
ON ROWS
FROM MySamp.basic;

GetLastDay

For a given date_part, this function returns the last day of the time interval for the input date, following a standard Gregorian calendar.

Syntax

GetLastDay ( date, date_part )

Parameters

date
A number representing the input date between January 1, 1970 and Dec 31, 2037. The number is the number of seconds elapsed since midnight, January 1, 1970. To retrieve this number, use any of the following functions: Today(), TodateEx(), GetFirstDate(), GetLastDate(), DateRoll(). Date-Time type attribute properties of a member can also be used to retrieve this number. For example: Product.currentmember.[Intro Date] returns the Introduction or release date for the current product in context. [Cola].[Intro Date] returns the Introduction or release date for the “Cola” product.

date_part
Defined time components of the standard calendar.
  * DP_YEAR - year of the input date.
  * DP_QUARTER – quarter of the input date.
  * DP_MONTH - month of the input date.
  * DP_WEEK - week of the input date.

Notes
This function can be used for getting the truncated date of an input date for a given date part, following a standard Gregorian calendar.
Example
Assuming today’s date is April 15 2007, consider the following scenarios.

GetLastDay(Today(), DP_YEAR)
returns the last day of the year, Dec 31 2007

GetLastDay(Today(), DP_QUARTER)
returns the last day of the quarter, Jun 30 2007

GetLastDay(Today(), DP_MONTH)
returns the last day of the month, Apr 30 2007

GetLastDay(Today(), DP_WEEK)
returns the last day of the week, Apr 21 2007

See Also
GetFirstDay
GetNextDay
Today

GetNextDay

To the given date and the week day, get the next date after input date that corresponds to the week day.

Syntax

GetNextDay ( date, week_day, [0|1] )

Parameters

date
A number representing the input date between January 1, 1970 and Dec 31, 2037. The number is the number of seconds elapsed since midnight, January 1, 1970. To retrieve this number, use any of the following functions: Today(), TodateEx(), GetFirstDate(), GetLastDate(), DateRoll(). Date-Time type attribute properties of a member can also be used to retrieve this number. For example: Product.currentmember.[Intro Date] returns the Introduction or release date for the current product in context. [Cola].[Intro Date] returns the Introduction or release date for the “Cola” product.

week_day
A number between 1 (Sunday) and 7 (Saturday) representing the week day.
**0 or 1**
Optional. Indicates whether to include the date itself or not. Default behavior is 1: to include the date itself.

**Example**

GetNextDay(Today(), 2, 0)

returns the next Monday following today.

GetNextDay(Today(), 2, 1)

returns the next Monday following today, or today if today is Monday.

GetNextDay(Today(), 2)

returns the next Monday following today, or today if today is Monday.

**See Also**

GetFirstDay

GetLastDay

Today

**GetRoundDate**

For a given `date_part`, this function returns the rounded date of the input date to the input time interval, following a standard Gregorian calendar.

**Syntax**

GetRoundDate ( `date`, `date_part` )

**Parameters**

`date`
A number representing the input date between January 1, 1970 and Dec 31, 2037. The number is the number of seconds elapsed since midnight, January 1, 1970. To retrieve this number, use any of the following functions: Today(), TodateEx(), GetFirstDate(), GetLastDate(), DateRoll(). Date-Time type attribute properties of a member can also be used to retrieve this number. For example: `Product.currentmember.[Intro Date]` returns the Introduction or release date for the current product in context. `[Cola].[Intro Date]` returns the Introduction or release date for the “Cola” product.

`date_part`
Defined time components of the standard calendar.

- **DP_YEAR** - year of the input date.
• DP_QUARTER – quarter of the input date.
• DP_MONTH - month of the input date.
• DP_WEEK - week of the input date.

Example
Assuming today’s date is April 15 2007, consider the following scenarios.

GetRoundDate(Today(), DP_YEAR)
returns the rounded date to the year, Jan 1 2007

GetRoundDate(Today(), DP_QUARTER)
returns the rounded date to the quarter, Apr 1 2007

GetRoundDate(Today(), DP_MONTH)
returns the rounded date to the month, Apr 1 2007

GetRoundDate(Today(), DP_WEEK)
returns the rounded date to the week, Apr 15 2007

See Also
GetNextDay
GetFirstDay
GetLastDay
Today

Head

Returns the first \( n \) members or tuples present in a set.

Syntax

Head ( set [, numeric value expression ] )

Parameters

set
The set from which to take items.

numeric value expression
The count of items to take from the beginning of the set. If omitted, the default is 1. If less than 1, an empty set is returned. If the value exceeds the number of tuples in the input set, the original set is returned.
Example

Example 1
This example uses the following part of the Sample Basic outline:

```
- Product
  - 100
  - 200
  - 300
  - 400
  - Diet
```

The following expression

```
[Product].children
```

returns the set:

```
{ [100], [200], [300], [400], [Diet] }
```

Therefore, the following expression

```
Head ( [Product].children, 2)
```

returns the first two members of the previous result set:

```
{ [100], [200] }
```

Example 2
This example uses the following parts of the Sample Basic outline:

```
- 100 (+)
  --- 100-10
  --- 100-20
  --- 100-30

- South (+)
  --- Texas
  --- Oklahoma
  --- Louisiana
  --- New Mexico
```
The following expression

```mdx
CrossJoin ([100].children, [South].children )
```

returns the set:

```
{ ([100-10], Texas), ([100-10], Oklahoma), ([100-10], Louisiana),
  ([100-20], Texas), ([100-20], Oklahoma), ([100-20], Louisiana),
  ([100-20], [New Mexico]),
  ([100-30], Texas), ([100-30], Oklahoma), ([100-30], Louisiana),
  ([100-30], [New Mexico]) }
```

And the following expression

```mdx
Head ( CrossJoin ([100].children, [South].children), 8 )
```

returns the first 8 tuples of the previous result set:

```
{ ([100-10], Texas), ([100-10], Oklahoma), ([100-10], Louisiana),
  ([100-20], Texas), ([100-20], Oklahoma), ([100-20], Louisiana),
  ([100-20], [New Mexico]),
  ([100-30], Texas), ([100-30], Oklahoma), ([100-30], Louisiana),
  ([100-30], [New Mexico]) }
```

Additionally, the following expression

```mdx
([Year].generations(2).members)
```

returns the set of members comprising the second generation of the Year dimension:

```
{ [Qtr1], [Qtr2], [Qtr3], [Qtr4] }
```

Therefore, the following query

```
SELECT
  {([Year].generations(2).members)}
ON COLUMNS,
Head ( CrossJoin ( [100].children, [South].children), 8 )
ON ROWS
FROM Sample.Basic
```
returns the grid:

### Table 6-76  Output Grid from MDX Example

<table>
<thead>
<tr>
<th>(axis)</th>
<th>Qtr1</th>
<th>Qtr2</th>
<th>Qtr3</th>
<th>Qtr4</th>
</tr>
</thead>
<tbody>
<tr>
<td>100–10</td>
<td>Texas</td>
<td>489</td>
<td>536</td>
<td>653</td>
</tr>
<tr>
<td></td>
<td>Oklahoma</td>
<td>87</td>
<td>92</td>
<td>128</td>
</tr>
<tr>
<td></td>
<td>Louisiana</td>
<td>93</td>
<td>106</td>
<td>128</td>
</tr>
<tr>
<td></td>
<td>New Mexico</td>
<td>76</td>
<td>101</td>
<td>122</td>
</tr>
<tr>
<td>100–20</td>
<td>Texas</td>
<td>206</td>
<td>199</td>
<td>152</td>
</tr>
<tr>
<td></td>
<td>Oklahoma</td>
<td>84</td>
<td>66</td>
<td>55</td>
</tr>
<tr>
<td></td>
<td>Louisiana</td>
<td>119</td>
<td>158</td>
<td>171</td>
</tr>
<tr>
<td></td>
<td>New Mexico</td>
<td>–103</td>
<td>–60</td>
<td>–98</td>
</tr>
</tbody>
</table>

See Also

Tail

**Hierarchize**

Returns members of a set in their hierarchical order as represented in the database outline.

**Syntax**

```
Hierarchize ( set [,POST] )
```

**Parameters**

- **set**
  Set specification.

- **POST**
  If this keyword is used, child members are returned before their parents.

**Notes**

This function returns members of a set in their hierarchical order as represented in the database outline (viewed from top-down by default, meaning that parent members are returned before their children).

If **POST** is used, child members are returned before their parents (the view changes to bottom-up). For example,

Hierarchize({Child, Grandparent, Parent})

returns {Grandparent, Parent, Child}.

Hierarchize({Child, Grandparent, Parent}, POST)

returns {Child, Parent, Grandparent}.
Example

Example 1
The following expression

Hierarchize({May, Apr, Jun})

returns the set:

{Apr, May, Jun}

Therefore, the following query

Select
Hierarchize({May, Apr, Jun})
on columns from sample.basic

returns the grid:

Table 6-77  Output Grid from MDX Example

<table>
<thead>
<tr>
<th>Apr</th>
<th>May</th>
<th>Jun</th>
</tr>
</thead>
<tbody>
<tr>
<td>8644</td>
<td>8929</td>
<td>9534</td>
</tr>
</tbody>
</table>

Example 2
The following expression

Hierarchize({May, Qtr2, Apr, Jun})

returns the set:

{ Qtr2 Apr May Jun }

Therefore, the following query

Select
Hierarchize({May, Qtr2, Apr, Jun})
on columns from sample.basic

returns the grid:

Table 6-78  Output Grid from MDX Example

<table>
<thead>
<tr>
<th>Qtr2</th>
<th>Apr</th>
<th>May</th>
<th>Jun</th>
</tr>
</thead>
<tbody>
<tr>
<td>27107</td>
<td>8644</td>
<td>8929</td>
<td>9534</td>
</tr>
</tbody>
</table>
Example 3

The following expression

Hierarchize({May, Qtr2, Apr, Jun}, POST)

returns the set:

{Apr, May, Jun, Qtr2}

Therefore, the following query

Select
Hierarchize({May, Qtr2, Apr, Jun}, POST)
on columns from sample.basic

returns the grid:

Table 6-79  Output Grid from MDX Example

<table>
<thead>
<tr>
<th>Apr</th>
<th>May</th>
<th>Jun</th>
<th>Qtr2</th>
</tr>
</thead>
<tbody>
<tr>
<td>8644</td>
<td>8929</td>
<td>9534</td>
<td>27107</td>
</tr>
</tbody>
</table>

Example 4

The following query

Select
Hierarchize({Dec, Year, Feb, Apr, Qtr1, Jun, Qtr2}, POST)
on columns,
Hierarchize({Margin, Sales})
on rows
from sample.basic

returns the grid:

Table 6-80  Output Grid from MDX Example

<table>
<thead>
<tr>
<th>(axis)</th>
<th>Feb</th>
<th>Qtr1</th>
<th>Apr</th>
<th>Jun</th>
<th>Qtr2</th>
<th>Dec</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>Margin</td>
<td>17762</td>
<td>52943</td>
<td>18242</td>
<td>19457</td>
<td>56317</td>
<td>18435</td>
<td>221519</td>
</tr>
<tr>
<td>Sales</td>
<td>32069</td>
<td>95820</td>
<td>32917</td>
<td>35088</td>
<td>101679</td>
<td>33342</td>
<td>400855</td>
</tr>
</tbody>
</table>

IIF

Performs a conditional test, and returns an appropriate numeric expression or set depending on whether the test evaluates to true or false.
Syntax

\[ \text{IIF ( search\_condition, true\_part, false\_part )} \]

Parameters

**search\_condition**
An expression to evaluate as true or false (see MDX Grammar Rules).

**true\_part**
A \textit{value\_expression} or a set. \textit{IIF} returns this expression if the search condition evaluates to TRUE (something other than zero).
The \textit{value\_expression} can be a numeric value expression or a string value expression.

**false\_part**
A \textit{value\_expression} or a set. \textit{IIF} returns this expression if the search condition evaluates to FALSE (zero).
The \textit{value\_expression} can be a numeric value expression or a string value expression.

Example

**Example 1**
The company plans an expensive promotion of its caffeinated drinks. For the Caffeinated products only, the following query calculates a Revised Budget that is 110% of the regular budget.

\[
\text{WITH MEMBER}
\quad \text{[Scenario].[Revised Budget]}
\quad \text{AS}
\quad \quad \text{'IIF (}
\quad \quad \quad \text{[Product].CurrentMember.Caffeinated,}
\quad \quad \quad \quad \text{Budget} \ast 1.1, \text{Budget}
\quad \quad \quad \text{)}'
\]

\text{SELECT}
\quad \{\text{[Scenario].[Budget], [Scenario].[Revised Budget]}\}
\quad \text{ON COLUMNS,}
\quad \text{[Product].Levels(0).Members}
\quad \text{ON ROWS}
\quad \text{FROM Sample.Basic}
\quad \text{WHERE ([Measures].[Sales], [Year].[Qtr3])}
\]

This query returns the grid:

<table>
<thead>
<tr>
<th>(axis)</th>
<th>Budget</th>
<th>Revised Budget</th>
</tr>
</thead>
<tbody>
<tr>
<td>100-10</td>
<td>18650</td>
<td>20515</td>
</tr>
<tr>
<td>100-20</td>
<td>8910</td>
<td>9801</td>
</tr>
<tr>
<td>100-30</td>
<td>3370</td>
<td>3370</td>
</tr>
</tbody>
</table>
### Table 6-81  (Cont.) Output Grid from MDX Example

<table>
<thead>
<tr>
<th>(axis)</th>
<th>Budget</th>
<th>Revised Budget</th>
</tr>
</thead>
<tbody>
<tr>
<td>200-10</td>
<td>11060</td>
<td>12166</td>
</tr>
<tr>
<td>200-20</td>
<td>9680</td>
<td>10648</td>
</tr>
<tr>
<td>200-30</td>
<td>3880</td>
<td>3880</td>
</tr>
<tr>
<td>200-40</td>
<td>2660</td>
<td>2660</td>
</tr>
<tr>
<td>300-10</td>
<td>10600</td>
<td>11660</td>
</tr>
<tr>
<td>300-20</td>
<td>3760</td>
<td>4136</td>
</tr>
<tr>
<td>300-30</td>
<td>8280</td>
<td>9108</td>
</tr>
<tr>
<td>400-10</td>
<td>7750</td>
<td>7750</td>
</tr>
<tr>
<td>400-20</td>
<td>6800</td>
<td>6800</td>
</tr>
<tr>
<td>400-30</td>
<td>3290</td>
<td>3290</td>
</tr>
<tr>
<td>100-20</td>
<td>8910</td>
<td>8910</td>
</tr>
<tr>
<td>200-20</td>
<td>9680</td>
<td>9680</td>
</tr>
<tr>
<td>300-30</td>
<td>8280</td>
<td>8280</td>
</tr>
</tbody>
</table>

#### Example 2

The following query calculates a Revised Budget equaling Budget for caffeinated products, and Actual for non-caffeinated products.

```sql
WITH MEMBER [Scenario].[Revised Budget] AS 'StrToMbr(IIF ([Product].CurrentMember.Caffeinated, "Budget", "Actual"))'
SELECT {[Scenario].[Budget], [Scenario].[Revised Budget]} ON COLUMNS,
       Children([100]) ON ROWS
FROM Sample.Basic
WHERE ([Measures].[Sales], [Year].[Qtr3])
```

This query returns the grid:

### Table 6-82  Output Grid from MDX Example

<table>
<thead>
<tr>
<th>(axis)</th>
<th>Budget</th>
<th>Revised Budget</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cola</td>
<td>18650</td>
<td>18650</td>
</tr>
<tr>
<td>Diet Cola</td>
<td>8910</td>
<td>8910</td>
</tr>
<tr>
<td>Caffeine Free Cola</td>
<td>3370</td>
<td>3189</td>
</tr>
</tbody>
</table>
InStr

Returns a number specifying the position of the first occurrence of one string within another. This function includes a required "start" parameter.

Syntax

InStr ( start, string1, string2 [,compare] )

Parameters

start
Character position to begin search in string1. For example, a position value of 1 indicates that the search begins at the first character in the string. This parameter is required.

string1
String expression or literal string in which to search.

string2
String expression or literal string for which to search.

compare
Optional search mode. Values: 0 for case sensitive, 1 for case insensitive. Default is case sensitive.

Notes

If a matching string is not found, the return value is 0.
If you require an optional "start" argument, then use the InString function instead.

Example

InStr (5, "Year2000_promotional", "promotional", 1)

returns 10

InString

Returns a number specifying the position of the first occurrence of one string within another.

Syntax

InString (string1, string2, [start] [,compare])

Parameters

string1
String expression or literal string in which to search.
**string2**  
String expression or literal string for which to search.

**start**  
Optional character position to begin search in *string1*. The default value is 1. A position value of 1 indicates the very first character in the string. If omitted, search begins at first character in *string1*.

**compare**  
Optional search mode. Values: 0 for case sensitive, 1 for case insensitive. Default is case sensitive.

**Notes**  
If a matching string is not found, the return value is 0.

**Example**

```
InString ("Year2000_promotional", "promotional", 5,1)
returns 10
```

If the start parameter is omitted, the comma before the compare parameter is still required:

```
InString ("Year2000_promotional", "promotional", ,1)
```

If the compare parameter is omitted, the comma before the start parameter is still required:

```
InString ("Year2000_promotional", "promotional", 5)
```

**Int**

Returns the next lowest integer value of an expression.

**Syntax**

```
Int ( numeric_value_expression )
```

**Parameters**

`numeric_value_expression`  
A numeric value or an expression that returns a numeric value (see MDX Grammar Rules).

**Example**

**Example 1**

```
Int (104.504) returns 104.
```
Example 2

The following query

```
WITH MEMBER [Market].[West_approx]
AS
'Int(
  Sum(
    Children([Market].[West])
  )
)
SELECT
  {[Year].[Qtr1].Children}
ON COLUMNS,
  {[Market].[West].children,
    [Market].[West_approx]}
ON ROWS
FROM
  Sample.Basic
WHERE ([Measures].[Profit %], [Product].[Cola], [Scenario].[Actual])
```

returns the grid:

Table 6-83  Output Grid from MDX Example

<table>
<thead>
<tr>
<th>(axis)</th>
<th>Jan</th>
<th>Feb</th>
<th>Mar</th>
</tr>
</thead>
<tbody>
<tr>
<td>California</td>
<td>38.643</td>
<td>37.984</td>
<td>38.370</td>
</tr>
<tr>
<td>Oregon</td>
<td>17.500</td>
<td>16.129</td>
<td>16.107</td>
</tr>
<tr>
<td>Washington</td>
<td>29.231</td>
<td>30.986</td>
<td>32.000</td>
</tr>
<tr>
<td>Utah</td>
<td>23.077</td>
<td>23.077</td>
<td>20.968</td>
</tr>
<tr>
<td>Nevada</td>
<td>-3.947</td>
<td>-6.757</td>
<td>-5.333</td>
</tr>
<tr>
<td>West_approx</td>
<td>104.000</td>
<td>101.00</td>
<td>102.000</td>
</tr>
</tbody>
</table>

Intersect

Returns the intersection of two input sets, optionally retaining duplicates.

Syntax

```
Intersect ( set1, set2 [,ALL] )
```

Parameters

- **set1**
  A set to intersect with set2.

- **set2**
  A set to intersect with set1.
The optional ALL keyword retains matching duplicates in set1 and set2.

Notes
Duplicates are eliminated by default from the tail of the set. The optional ALL keyword retains duplicates. The two input sets must have identical dimension signatures. For example, if set1 consists of dimensions Product and Market, in that order, then set2 should also consist of Product followed by Market.

Example

Example 1
The following expression

\[
\text{Intersect}([\text{New York}, \text{California}, \text{Oregon}], \\
[\text{California}, \text{Washington}, \text{Oregon}])
\]

returns the set:

\[
[\text{California}, \text{Oregon}]
\]

Therefore, the following query

\[
\text{SELECT} \\
\text{Intersect}([\text{New York}, \text{California}, \text{Oregon}], \\
[\text{California}, \text{Washington}, \text{Oregon}]) \\
\text{ON COLUMNS} \\
\text{FROM Sample.Basic}
\]

returns the grid:

Table 6-84 Output Grid from MDX Example

<table>
<thead>
<tr>
<th>California</th>
<th>Oregon</th>
</tr>
</thead>
<tbody>
<tr>
<td>12964</td>
<td>5062</td>
</tr>
</tbody>
</table>

Example 2
The following expression

\[
\text{Intersect}([\text{New York}, \text{California}, \text{Florida}, \text{California}], \\
[\text{Oregon}, \text{Washington}, \text{California}, \text{Florida}, \text{California}], \text{ALL})
\]
returns the set:

\{ \text{[California]}, \text{[Florida]}, \text{[California]} \}

Therefore, the following query

\text{SELECT Intersect( \{ \text{[New York]}, \text{[California]}, \text{[Florida]}, \text{[California]} \}, \{ \text{[Oregon]}, \text{[Washington]}, \text{[California]}, \text{[Florida]}, \text{[California]} \}, \text{ALL}) ON COLUMNS FROM Sample.Basic}

returns the grid:

\begin{tabular}{|l|l|l|}
\hline
\text{California} & \text{Florida} & \text{California} \\
\hline
12964 & 5029 & 12964 \\
\hline
\end{tabular}

The matching duplicate element \text{[California]} is duplicated in the result.

However, the following expression

\text{Intersect( \{ \text{[New York]}, \text{[California]}, \text{[Florida]}, \text{[California]} \}, \{ \text{[Oregon]}, \text{[Washington]}, \text{[California]}, \text{[Florida]} \}, \text{ALL})}

would return only

\{ \text{[California]}, \text{[Florida]} \}

because only one match exists between \text{[California]} in \text{set1} and \text{[California]} in \text{set2}.

\text{IS}

Returns TRUE if two members are identical.

\text{Syntax}

\text{IS( member1, member2 )}

\text{member1 IS member2}
Parameters

**member1**
First member specification.

**member2**
Second member specification.

Example

IS([Year].CurrentMember.Parent, [Qtr1])

returns TRUE if the parent of the current member in [Year] dimension is [Qtr1].

Filter([Year].Levels(0).members, IS([Year].CurrentMember.Parent, [Qtr1]))

returns children of [Qtr1].

The following query returns all members of [Market] that have the parent [East]; in other words, children of [East].

```
SELECT
  {
    Filter {
      [Market].members,
      [Market].CurrentMember.Parent IS [East]
    }
  }

on columns
FROM sample.basic
```

This query returns the following grid:

<table>
<thead>
<tr>
<th>New York</th>
<th>Massachusetts</th>
<th>Florida</th>
<th>Connecticut</th>
<th>New Hampshire</th>
</tr>
</thead>
<tbody>
<tr>
<td>8202</td>
<td>6712</td>
<td>5029</td>
<td>3093</td>
<td>1125</td>
</tr>
</tbody>
</table>

**IsAccType**

Returns TRUE if the member has the associated accounts tag. Account tags apply only to dimensions marked as Accounts dimensions. A FALSE value is returned for all other dimensions.

Syntax

IsAccType ( member , AcctTag )
Parameters

**member**
A member specification.

**AcctTag**
Valid values (defined in the database outline):
- First
- Last
- Average
- Expense
- TwoPass

Example

```sql
SELECT Filter([Measures].Members, IsAccType([Measures].CurrentMember, First))
ON COLUMNS
FROM Sample.Basic
```

This query returns the following grid:

<table>
<thead>
<tr>
<th>Opening Inventory</th>
<th>117405</th>
</tr>
</thead>
</table>

IsAncestor

Returns TRUE if the first member is an ancestor of the second member and, optionally, if the first member is equal to the second member.

Syntax

```
IsAncestor ( member1 , member2 [, INCLUDEMEMBER])
```

Parameters

**member1**
A member specification.

**member2**
A member specification.

**INCLUDEMEMBER**
Optional. Use this keyword if you want IsAncestor to return TRUE if the first member is equal to the second member.
Example

Example 1

The following query returns all Market dimension members for which the expression
IsAncestor([Market].CurrentMember, [Florida]) returns TRUE; in other words, the
query returns all ancestors of Florida.

```
SELECT
  Filter([Market].Members, IsAncestor([Market].CurrentMember, [Florida]))
ON COLUMNS
FROM Sample.Basic
```

Table 6-88  Output Grid from MDX Example

<table>
<thead>
<tr>
<th>Market</th>
<th>East</th>
</tr>
</thead>
<tbody>
<tr>
<td>105522</td>
<td>24161</td>
</tr>
</tbody>
</table>

Example 2

The following query is the same as the above query, except that it uses
INCLUDEMEMBER. It returns all Market dimension members for which the expression
IsAncestor([Market].CurrentMember, [Florida], INCLUDEMEMBER)) returns TRUE;
in other words, the query returns Florida and all ancestors of Florida.

```
SELECT
  Filter([Market].Members, IsAncestor([Market].CurrentMember, [Florida],
  INCLUDEMEMBER))
ON COLUMNS
FROM Sample.Basic

{[Market], [East], [Florida]}
```

Table 6-89  Output Grid from MDX Example

<table>
<thead>
<tr>
<th>Market</th>
<th>East</th>
<th>Florida</th>
</tr>
</thead>
<tbody>
<tr>
<td>105522</td>
<td>24161</td>
<td>5029</td>
</tr>
</tbody>
</table>

IsChild

Returns TRUE if the first member is a child of the second member and, optionally, if
the first member is equal to the second member.

Syntax

IsChild ( member1 , member2 [, INCLUDEMEMBER])
Parameters

**member1**
A member specification.

**member2**
A member specification.

**INCLUDEMEMBER**
Optional. Use this keyword if you want IsChild to return TRUE if the first member is equal to the second member.

**Example**

**Example 1**

The following query returns all Market dimension members for which the expression IsChild([Market].CurrentMember, [East]) returns TRUE; in other words, the query returns all children of East.

```plaintext
SELECT Filter([Market].Members, IsChild([Market].CurrentMember, [East]))
ON COLUMNS
FROM Sample.Basic
```

**Table 6-90  Output Grid from MDX Example**

<table>
<thead>
<tr>
<th>New York</th>
<th>Massachusetts</th>
<th>Florida</th>
<th>Connecticut</th>
<th>New Hampshire</th>
</tr>
</thead>
<tbody>
<tr>
<td>8202</td>
<td>6712</td>
<td>5029</td>
<td>3093</td>
<td>1125</td>
</tr>
</tbody>
</table>

**Example 2**

The following query is the same as the above query, except that it uses INCLUDEMEMBER. It returns all Market dimension members for which the expression IsChild([Market].CurrentMember, [East]) returns TRUE; in other words, the query returns East and all children of East.

```plaintext
SELECT Filter([Market].Members, IsChild([Market].CurrentMember, [East], INCLUDEMEMBER))
ON COLUMNS
FROM Sample.Basic
```

**Table 6-91  Output Grid from MDX Example**

<table>
<thead>
<tr>
<th>East</th>
<th>New York</th>
<th>Massachusetts</th>
<th>Florida</th>
<th>Connecticut</th>
<th>New Hampshire</th>
</tr>
</thead>
<tbody>
<tr>
<td>24161</td>
<td>8202</td>
<td>6712</td>
<td>5029</td>
<td>3093</td>
<td>1125</td>
</tr>
</tbody>
</table>
IsEmpty

Returns True if the value of an input numeric-value-expression evaluates to #MISSING, and returns FALSE otherwise.

Syntax

IsEmpty ( value_expression )

Parameters

value_expression
A set returning values to check for emptiness.

Notes

Zero is not equivalent to #MISSING. isEmpty(0) returns TRUE.

Example

The following example finds all Product, Market combinations for which Sales data exists.

SELECT
  Filter([NewSet], NOT IsEmpty([NewSet].CurrentTuple))
ON COLUMNS
FROM Sample.Basic
WHERE
  {[Sales]}

This query returns the following grid:

<table>
<thead>
<tr>
<th>East</th>
<th>West</th>
<th>South</th>
<th>Central</th>
<th>...</th>
<th>East</th>
<th>West</th>
<th>Central</th>
<th>East</th>
<th>West</th>
<th>South</th>
<th>Central</th>
</tr>
</thead>
<tbody>
<tr>
<td>27740</td>
<td>28306</td>
<td>16280</td>
<td>33808</td>
<td>...</td>
<td>15745</td>
<td>35034</td>
<td>33451</td>
<td>7919</td>
<td>36423</td>
<td>18676</td>
<td>42660</td>
</tr>
</tbody>
</table>

IsGeneration

Returns TRUE if the member is in a specified generation.

Syntax

IsGeneration { member, index }
Parameters

*member*
A member specification.

*index*
A generation number.

Example

IsGeneration([Market].CurrentMember, 2)

returns TRUE if the current member of the Market dimension is at generation 2.

Therefore, the following query

```
SELECT
  Filter([Market].Members, IsGeneration([Market].CurrentMember, 2))
ON COLUMNS
FROM Sample.Basic
```

returns

<table>
<thead>
<tr>
<th>Table 6-93</th>
<th>Output Grid from MDX Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>East</td>
<td>West</td>
</tr>
<tr>
<td>24161</td>
<td>29861</td>
</tr>
</tbody>
</table>

See Also

*Generation*

*IsLevel*

**IsLeaf**

Returns TRUE if the member is a level-0 member.

Syntax

```
IsLeaf ( member )
```

Parameters

*member*
A member specification.

Notes

*IsLeaf(member) is the same as IsLevel(member, 0).*
Example

IsLeaf([Market].CurrentMember)

returns TRUE if the current member of the Market dimension is at level 0.

Therefore, the following query

SELECT
    Filter([Market].Members, IsLeaf([Market].CurrentMember))
ON COLUMNS
FROM Sample.Basic

returns

<table>
<thead>
<tr>
<th>New York</th>
<th>Massachusetts</th>
<th>Florida</th>
<th>...</th>
<th>Missouri</th>
<th>Iowa</th>
<th>Colorado</th>
</tr>
</thead>
<tbody>
<tr>
<td>8202</td>
<td>6712</td>
<td>5029</td>
<td></td>
<td>1466</td>
<td>9061</td>
<td>7227</td>
</tr>
</tbody>
</table>

IsLevel

Returns TRUE if the member is in a specified level.

Syntax

IsLevel ( member , index )

Parameters

member
A member specification.

index
A level number.

Example

IsLevel([Market].CurrentMember, 1)

returns TRUE if the current member of the Market dimension is at level 1.

Therefore, the following query

SELECT
    Filter([Market].Members, IsLevel([Market].CurrentMember, 1))
ON COLUMNS
FROM Sample.Basic
returns

Table 6-95  Output Grid from MDX Example

<table>
<thead>
<tr>
<th>East</th>
<th>West</th>
<th>South</th>
<th>Central</th>
</tr>
</thead>
<tbody>
<tr>
<td>24161</td>
<td>29861</td>
<td>13238</td>
<td>38262</td>
</tr>
</tbody>
</table>

See Also
Level
IsGeneration

IsMatch

Performs wild-card search / pattern matching to check if a string matches a given pattern. The input string can be a member name, an alias, an attribute value, or any relevant string. This function searches for strings matching the pattern you specify, and returns the artifacts it finds.

Syntax

`IsMatch(string, patternstring, {MATCH_CASE | IGNORE_CASE})`

Parameters

`string`
The string that should be tested against the pattern.

`patternstring`
The pattern to search for. Must be in POSIX Extended Regular Expression Syntax. See the syntax specification at The Open Group. See the Notes in this topic for additional rules regarding special characters.

`MATCH_CASE`
Optional. Consider `patternstring` to be case sensitive. If `MATCH_CASE / IGNORE_CASE` are omitted, Essbase defaults to the case-sensitive setting of the outline properties.

`IGNORE_CASE`
Optional. Do not consider `patternstring` to be case sensitive. If `MATCH_CASE / IGNORE_CASE` are omitted, Essbase defaults to the case-sensitive setting of the outline properties.

Notes

- To search for a member name containing $, you must precede it with three backslash (\) escape characters in the `patternstring`. For example, to search for member a$bc in Market, you must use `IsMatch(Market.CurrentMember.MEMBER_NAME, "a\\\\$bc")`.
- To search for a character at the end of a line, you must precede the POSIX end-of-line anchor, which is a dollar sign ($), with one backslash (\) escape character in the `patternstring`. For example, to search for a member name that ends with a c in Market, you must use `IsMatch(Market.CurrentMember.MEMBER_NAME, "c\$")`. 
• To search for any other special characters besides $, you must precede them with two backslash (\) escape characters in the patternstring. For example, to search for member a?bc in Market, you must use
   IsMatch(Market.CurrentMember.MEMBER_NAME, "a\?bc").

Example

The following query searches for members whose names start with "new":

```sql
SELECT
  Filter(Market.Levels(0).Members,
    IsMatch(Market.CurrentMember.MEMBER_NAME, "^new")
  )
ON COLUMNS
FROM Sample.Basic
```

The following query searches for members whose names start with at least an "n":

```sql
SELECT
  Filter(Market.Levels(0).Members,
    ISMATCH(Market.CurrentMember.MEMBER_NAME, "^n+")
  )
ON COLUMNS
FROM Sample.Basic
```

The following query searches for members whose names contain an "*":

```sql
SELECT
  Filter(Year.Members,
    ISMATCH(Year.CurrentMember.MEMBER_NAME, "\*")
  )
ON COLUMNS
FROM Sample.Basic
```

The following query searches for members whose names contain zero or an "a":

```sql
SELECT
  Filter(Year.Members,
    ISMATCH(Year.CurrentMember.MEMBER_NAME, "a?")
  )
ON COLUMNS
FROM Sample.Basic
```

IsSibling

Returns TRUE if the first member is a sibling of the second member and, optionally, if the first member is equal to the second member.
Syntax

\texttt{IsSibling( member1, member2 [, INCLUDEMEMBER] )}

Parameters

\textbf{member1}

A member specification.

\textbf{member2}

A member specification.

\textbf{INCLUDEMEMBER}

Optional. Use this keyword if you want \texttt{IsSibling} to return TRUE if the first member is equal to the second member.

Example

Example 1

The following query returns all Market dimension members for which the expression \texttt{IsSibling([Market].CurrentMember, [California])} returns TRUE; in other words, the query returns all states that are siblings of California.

\begin{verbatim}
SELECT
  Filter([Market].Members, IsSibling([Market].CurrentMember, [California]))
ON COLUMNS
FROM Sample.Basic
\end{verbatim}

Table 6-96  Output Grid from MDX Example

<table>
<thead>
<tr>
<th>Oregon</th>
<th>Washington</th>
<th>Utah</th>
<th>Nevada</th>
</tr>
</thead>
<tbody>
<tr>
<td>5062</td>
<td>4641</td>
<td>3155</td>
<td>4039</td>
</tr>
</tbody>
</table>

Example 2

The following query is the same as the above query, except that it uses \texttt{INCLUDEMEMBER}. It returns all Market dimension members for which the expression \texttt{IsSibling([Market].CurrentMember, [California])} returns TRUE; in other words, the query returns all states that are siblings of California, including California itself.

\begin{verbatim}
SELECT
  Filter([Market].Members, IsSibling([Market].CurrentMember, [California],
  INCLUDEMEMEMBER))
ON COLUMNS
FROM Sample.Basic
\end{verbatim}

Table 6-97  Output Grid from MDX Example

<table>
<thead>
<tr>
<th>California</th>
<th>Oregon</th>
<th>Washington</th>
<th>Utah</th>
<th>Nevada</th>
</tr>
</thead>
<tbody>
<tr>
<td>12964</td>
<td>5062</td>
<td>4641</td>
<td>3155</td>
<td>4039</td>
</tr>
</tbody>
</table>
IsUda

Returns TRUE if the member has the associated UDA tag (user-defined attribute).

Syntax

IsUda ( member, string_value_expression )

Parameters

member
A member specification.

string_value_expression
A user-defined attribute (UDA) name string, defined in the database outline.

Example

IsUda([Market].CurrentMember, "Major Market")

returns TRUE if the current member of the Market has the user-defined attribute "Major Market."

Therefore, the following query

SELECT Filter([Market].Members, IsUda([Market].CurrentMember, "Major Market")) ON COLUMNS FROM Sample.Basic

returns

| Table 6-98  Output Grid from MDX Example |
|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|
| East | New York | Massachusetts | Florida | Californi a | Texas | Central | Illinois | Ohio | Colorado |
| 24161 | 8202 | 6712 | 5029 | 12964 | 6425 | 38262 | 12577 | 4384 | 7227 |

IsValid

Returns TRUE if the specified element validates successfully.

Syntax

IsValid ( member | tuple | set | layer | property )
Parameters

**member**
A member specification.

**tuple**
A tuple specification.

**set**
A set specification.

**layer**
A layer specification.

**property**
A property specification (see MDX Grammar Rules).

Example

**Example 1**
The following example shows how IsValid can be used to check whether a given property value is valid. It returns all Product dimension members that have an Ounces attribute value of 12.

SELECT
Filter([Product].members,
IsValid([Product].CurrentMember.Ounces)
AND
[Product].CurrentMember.Ounces = 12)
ON COLUMNS
FROM Sample.Basic

The expression `IsValid([Product].CurrentMember.Ounces)` returns TRUE for only those members in the Product dimension that have a valid property value for [Ounces]. This eliminates ancestral members such as [Product] and [Colas] that do not have the [Ounces] property defined because they are not level-0 members of the Product dimension.

The second part of the AND condition in the filter selects only those members with a value of 12 for [Ounces].

This query returns the following grid:

<table>
<thead>
<tr>
<th>Table 6-99</th>
<th>Output Grid from MDX Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>100-10</td>
<td>22777</td>
</tr>
<tr>
<td>100-20</td>
<td>5708</td>
</tr>
<tr>
<td>200-10</td>
<td>7201</td>
</tr>
<tr>
<td>200-30</td>
<td>4636</td>
</tr>
<tr>
<td>300-30</td>
<td>11093</td>
</tr>
</tbody>
</table>

**Example 2**

IsValid([Jan].FirstChild)
returns FALSE, because [Jan] is a level-0 member, therefore it does not have any children.

**Item**

Extracts a member from a tuple.

Extracts a tuple from a set.

**Syntax**

Syntax that Returns a Member—one of the following:

\[tuple[.Item] \{ \text{index} \}\]

\[\text{Item} \{ \text{tuple, index} \}\]

Syntax that Returns a Tuple—one of the following:

\[\text{set[.Item]} \{ \text{index} \}\]

\[\text{Item} \{ \text{set, index} \}\]

**Parameters**

- **tuple**
  The tuple from which to get a member.

- **index**
  The usage depends upon whether you are returning a member or a tuple:
  - Returning a member: Numeric position (starting from 0) of the member to extract from the tuple. A valid value for \text{index} is from 0 to 1 less than the size of the input tuple. A value of less than 0, or greater than or equal to size of the input tuple, results in an empty member.
  - Returning a tuple: Numeric position (starting from 0) of the tuple to extract from the set. A valid value for \text{index} is from 0 to 1 less than the size of the input set. A value of less than 0, or greater than or equal to size of the input set, results in an empty tuple.

- **set**
  The set from which to get a tuple.

**Example**

**Example 1, Extracting a Member from a Tuple**

```xml
SELECT
{( [Qtr1], [Sales], [Cola], [Florida], [Actual] ).Item(3)}
ON COLUMNS
FROM Sample.Basic
```
returns:

**Table 6-100  Output Grid from MDX Example**

| Florida | 5029 |

SELECT
   {Item({ [Qtr1], [Sales], [Cola], [Florida], [Actual] }, 2)}
ON COLUMNS
FROM Sample.Basic

returns:

**Table 6-101  Output Grid from MDX Example**

| Cola | 22777 |

Example 2, Extracting a Tuple from a Set

The following query

SELECT
   {CrossJoin ([Market].CHILDREN, [Product].CHILDREN).ITEM(0)}
ON COLUMNS
FROM Sample.Basic

returns the first tuple in the set CrossJoin([Market].CHILDREN, [Product].CHILDREN), which is ([East], [Colas]):

The above query can also be written as:

SELECT
   {CrossJoin ([Market].CHILDREN, [Product].CHILDREN)(0)}
ON COLUMNS
FROM Sample.Basic

because the ITEM keyword is optional.
Example 3, Extracting Member from a Set

Consider the following crossjoined set of Market and Product members:

```
{ ([East],[100]), ([East],[200]), ([East],[300]), ([East],[400]), ([East],[Diet]),
  ([West],[100]), ([West],[200]), ([West],[300]), ([West],[400]), ([West],[Diet]),
  ([South],[100]), ([South],[200]), ([South],[300]), ([South],[400]), ([South],[Diet]),
  ([Central],[100]), ([Central],[200]), ([Central],[300]), ([Central],[400]), ([Central],[Diet])
}
```

The following example

```
CrossJoin([Market].CHILDREN, [Product].CHILDREN).item(0)
```

returns the first tuple of the crossjoined set, ([East],[100]), and the following example

```
CrossJoin([Market].CHILDREN, [Product].CHILDREN).item(0).item(1)
```

returns [100], the second member of the first tuple of the crossjoined set.

### JulianDate

To the given UNIX date, get its Julian date.

**Syntax**

```
JulianDate ( date )
```

**Parameters**

- **date**
  A number representing the input date between January 1, 1970 and Dec 31, 2037. The number is the number of seconds elapsed since midnight, January 1, 1970. To retrieve this number, use any of the following functions: Today(), TodateEx(), GetFirstDate(), GetLastDate(), DateRoll().
  Date-Time type attribute properties of a member can also be used to retrieve this number. For example: `Product.currentmember.[Intro Date]` returns the Introduction or release date for the current product in context. `[Cola].[Intro Date]` returns the Introduction or release date for the “Cola” product.

**Notes**

- This function is useful in converting the UNIX date to Julian Date or the 1900 Date system recognized by Microsoft Excel.
• In the 1900 date system, the first day that is supported is January 1, 1900. When
you enter a date, the date is converted into a serial number that represents the
number of elapsed days since January 1, 1900. For example, if you enter July 5,
1998, Microsoft Excel converts the date to the serial number 35981. By default,
Microsoft Excel for Windows uses the 1900 date system.

Return Value

This function returns juliandate, a number representing the Julian date. This number
is a continuous count of days and fractions elapsed since noon Universal Time on
January 1, 4713 BC in the proleptic Julian calendar.

Note:

For Excel workbooks using 1900 date system, \( (\text{JulianDate} - 2415018.50) \)
gets the sequential serial number as per 1900 date system.

Example

The following query returns the total monthly sales for all Colas along with their
release dates as in 1900 Date system in market “California” for “March 2007.”

WITH MEMBER
  Measures.[Product Intro Date]
AS
  \'\text{JulianDate(Product.CurrentMember.[Intro Date])} - 2415018.50\'
SELECT
  (Measures.[Product Intro Date], Measures.Sales)
ON COLUMNS,
  (Colas.Children)
ON ROWS
FROM Sample.Basic
WHERE
  (California, [March 2007], Actual);

See Also

UnixDate

Lag

Using the order of members existing in a database outline, returns a member that is \( n \)
steps behind a given member, along the same generation or level (as defined by
layertype).

Syntax

\[
\text{member.Lag } \{ \text{index [,layertype ] [, hierarchy ] } \}
\]

\[
\text{Lag } \{ \text{member, index [, hierarchy ] } \}
\]
Parameters

**member**
The starting member from which .LAG counts to a given number of previous members.

**index**
A number \( n \) representing how many steps prior to <member> to count.

**layertype**
GENERATION or LEVEL. Generation is the default.

**hierarchy**
Optional. A specific hierarchy within the time dimension.

Notes

- If the member specified by the Lag function does not exist, the result is an empty member. For example, using Sample Basic, \([\text{Jun}]\.lag\ (12)\) returns an empty member.
- When multiple hierarchies are enabled, this function returns NULL when the source member is in one hierarchy and the result member belongs to a different hierarchy.

Example

The following expression:

\([\text{Jun}]\.lag\ (3)\)

returns the member that is 3 steps prior to Jun:

\([\text{Mar}]\)

The following expression:

\([\text{Jun}]\.lag\ (-3)\)

returns the member that is 3 steps following Jun:

\([\text{Sep}]\)

For every month, the following query displays the sales and average over the last three months.

```sql
WITH MEMBER
[Measures].[Average Sales in Last 3 months]
AS
'Avg(
    ([Year].CurrentMember,
    [Year].CurrentMember.Lag(1),
    [Year]. CurrentMember.Lag(2)
```
This query returns the grid:

**Table 6-102  Output Grid from MDX Example**

<table>
<thead>
<tr>
<th>(axis)</th>
<th>Sales</th>
<th>Average Sales in Last 3 Months</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jan</td>
<td>31538</td>
<td>31538</td>
</tr>
<tr>
<td>Feb</td>
<td>23069</td>
<td>31803.500</td>
</tr>
<tr>
<td>March</td>
<td>32213</td>
<td>31940</td>
</tr>
<tr>
<td>April</td>
<td>32917</td>
<td>32399.667</td>
</tr>
<tr>
<td>May</td>
<td>33674</td>
<td>32934.667</td>
</tr>
<tr>
<td>Jun</td>
<td>35088</td>
<td>33893</td>
</tr>
<tr>
<td>Jul</td>
<td>36134</td>
<td>34965.333</td>
</tr>
<tr>
<td>Aug</td>
<td>36008</td>
<td>35743.333</td>
</tr>
<tr>
<td>Sep</td>
<td>33073</td>
<td>35071.667</td>
</tr>
<tr>
<td>Oct</td>
<td>32828</td>
<td>33969.667</td>
</tr>
<tr>
<td>Nov</td>
<td>31971</td>
<td>32624</td>
</tr>
<tr>
<td>Dec</td>
<td>33342</td>
<td>32713.667</td>
</tr>
</tbody>
</table>

See Also

**Lead**

**PrevMember**

**LastChild**

Returns the last child of the input member.

**Syntax**

```
member.LastChild
```

```
LastChild ( member )
```
Parameters

member
A member specification.

Example

SELECT
   [[Qtr1].firstchild]
ON COLUMNS,
   [[Market].[Central].lastchild]
ON ROWS
FROM Sample.Basic

Table 6-103  Output Grid from MDX Example

<table>
<thead>
<tr>
<th>(axis)</th>
<th>Jan</th>
</tr>
</thead>
<tbody>
<tr>
<td>Colorado</td>
<td>585</td>
</tr>
</tbody>
</table>

See Also

FirstChild
LastSibling

LastPeriods

Returns a set of members ending either at the specified member or at the current member in the time dimension.

Syntax

LastPeriods ( numeric value expression [, member [, hierarchy ] ] )

Parameters

numeric value expression
The number of members to return (see MDX Grammar Rules). If negative, member is treated as the starting point.

member
Optional. A member expression.

hierarchy
Optional. A specific hierarchy within the time dimension.

Example

Lastperiods(3, Apr) returns the set {Feb, Mar, Apr}.
Lastperiods(-3, Apr) returns the set {Apr, May, Jun}.
Lastperiods(1, Apr) returns a set of one member: {Apr}. 
Lastperiods(0, Apr) returns an empty set.

Lastperiods(5, Apr) returns the set \{Jan, Feb, Mar, Apr\}. Note that the output set has only four members.

The following query:

```
WITH MEMBER [Measures].[Rolling Sales] AS 'Avg ( LastPeriods (3, [Year].Currentmember ), [Measures].[Sales] )'
SELECT {[Measures].[Sales], [Measures].[Rolling Sales]} ON COLUMNS,
Descendants ([Year].[Qtr2]) ON ROWS
FROM Sample.Basic
WHERE [Product].[Root Beer]
```

returns the grid:

<table>
<thead>
<tr>
<th>(axis)</th>
<th>Sales</th>
<th>Rolling Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>Qtr2</td>
<td>27401</td>
<td>27014</td>
</tr>
<tr>
<td>Apr</td>
<td>8969</td>
<td>8960</td>
</tr>
<tr>
<td>May</td>
<td>9071</td>
<td>8997</td>
</tr>
<tr>
<td>Jun</td>
<td>9361</td>
<td>9133.667</td>
</tr>
</tbody>
</table>

See Also

PeriodsToDate
OpeningPeriod
ClosingPeriod
ParallelPeriod

**LastSibling**

Returns the last child of the input member's parent.

**Syntax**

```
LastSibling ( member [, hierarchy ])
```

```
member.LastSibling [(hierarchy)]
```
Parameters

**member**
A member specification.

**hierarchy**
Optional. A specific hierarchy within the time dimension.

Notes
If *member* is the top member of a dimension, then *member* itself is returned.

Example

Year.Lastsibling returns Year.
Qtr3.Lastsibling returns Qtr4.

See Also

FirstSibling
LastChild

Lead

Using the order of members existing in a database outline, returns a member that is \( n \) steps past a given member, along the same generation or level (as defined by *layertype*).

Syntax

```mdx
member.Lead (index [,layertype] [, hierarchy])
```

```mdx
Lead ( member, index [, hierarchy] )
```

Parameters

**member**
The starting member from which .LEAD counts a given number of following members.

**index**
A number \( n \) representing how many steps away from <member> to count.

**layertype**

GENERATION or LEVEL.

**hierarchy**
Optional. A specific hierarchy within the time dimension.

Notes

- If the member specified by the Lead function does not exist, the result is an empty member. For example, using Sample Basic, `[Jun].lead (12)` returns an empty member.
• When multiple hierarchies are enabled, this function returns NULL when the source member is in one hierarchy and the result member belongs to a different hierarchy.

**Example**

The following expression:

```
[Jan].lead (11)
```

returns the member that is 11 steps past Jan:

```
[Dec]
```

The following expression:

```
[Dec].lead (-11)
```

returns the member that is 11 steps prior to Dec:

```
[Jan]
```

For every month, the following query displays the marketing expenses and budgeted sales for the next month.

```mdx
WITH MEMBER
[Measures].[Expected Sales in Next month]
AS
'([Measures].[Sales], [Year].CurrentMember.Lead(1))'
SELECT
{ ([Scenario].[Actual], [Measures].[Marketing]),
  ([Scenario].[Budget], [Measures].[Expected Sales in Next month])
} ON COLUMNS,
[Year].Levels(0).Members
ON ROWS
FROM Sample.Basic
```

This query returns the grid:

**Table 6-105  Output Grid from MDX Example**

<table>
<thead>
<tr>
<th></th>
<th>Actual</th>
<th>Budget</th>
</tr>
</thead>
<tbody>
<tr>
<td>(axis)</td>
<td>(axis)</td>
<td></td>
</tr>
<tr>
<td>Jan</td>
<td>5223</td>
<td>30000</td>
</tr>
<tr>
<td>Feb</td>
<td>5289</td>
<td>30200</td>
</tr>
<tr>
<td>Mar</td>
<td>5327</td>
<td>30830</td>
</tr>
<tr>
<td>Apr</td>
<td>5421</td>
<td>31510</td>
</tr>
</tbody>
</table>
### Table 6-105 (Cont.) Output Grid from MDX Example

<table>
<thead>
<tr>
<th>(axis)</th>
<th>Actual Marketing</th>
<th>Budget Expected Sales in Next Month</th>
</tr>
</thead>
<tbody>
<tr>
<td>May</td>
<td>5530</td>
<td>32900</td>
</tr>
<tr>
<td>Jun</td>
<td>5765</td>
<td>33870</td>
</tr>
<tr>
<td>Jul</td>
<td>5985</td>
<td>33820</td>
</tr>
<tr>
<td>Aug</td>
<td>6046</td>
<td>31000</td>
</tr>
<tr>
<td>Sep</td>
<td>5491</td>
<td>29110</td>
</tr>
<tr>
<td>Oct</td>
<td>5388</td>
<td>29540</td>
</tr>
<tr>
<td>Nov</td>
<td>5263</td>
<td>30820</td>
</tr>
<tr>
<td>Dec</td>
<td>5509</td>
<td>#Missing</td>
</tr>
</tbody>
</table>

**See Also**

Lag

NextMember

---

**Leaves**

Returns the set of level 0 (leaf) members that contribute to the value of the specified member.

The Leaves function compactly describes large sets of members or tuples while avoiding pre-expansion of the set before retrieval. Because large sets tend to be very sparse, only a few members contribute to the input member (have non #Missing values) and are returned. As a result, Leaves consumes less memory resources than the equivalent nonempty Descendants function call, allowing for better scalability, especially in concurrent user environments.

Members with #MISSING values are not included in the return set.

When `member` is on the primary hierarchy, the return set is the set of descendants at level 0 that are nonempty.

The set returned by Leaves is the set of nonempty descendants at level 0, with a few differences. For example, when `member` is from an alternate hierarchy, the return set contains all primary, stored, level 0 members whose values are aggregated into `member`'s value. These contributing members may be either:

- Direct descendants of `member` along the alternate hierarchy
- Members that contribute value to a direct descendant of `member` by means of a shared member

In most cases, the Leaves function does not pre-expand the set prior to retrieval. Thus it requires less memory resources than the Descendants function, allowing for more scalability in dealing with large sets, especially in a high-concurrency user environment. Large sets tend to be very sparse; therefore, very few members are returned given the current point of view as defined by the MDX current member stack.

For example, a healthcare provider may have a database containing Doctor and Geography dimensions. While there may be hundreds of thousands, even millions, of
doctors, only a fraction have data associated with them for a given geographic location. Leaves is ideal for queries where the set is large but is sparse at a given point of view:

Select {[Copayments]} ON COLUMNS
CrossJoin(Leaves ([Doctors]), Leaves([Santa Clara County]) ON ROWS

The Leaves function is beneficial for queries on large dimensions.
In some cases, Leaves does require pre-expansion of sets, limiting the memory savings. Pre-expansion of sets likely will occur when the input member to Leaves is:

- On an Accounts dimension
- On a Time dimension
- On a dimension with fewer than 10,000 members

Syntax

Leaves ( member )

Parameters

member
The member for which contributing leaf members are sought

Notes

- This function is applicable only to aggregate storage databases. Using Leaves() with a non aggregate-storage input member returns an error.
- Leaves() is supported only for members in stored hierarchies. Using Leaves with a member in a dynamic hierarchy returns an error.
- If you modify the return set of Leaves with a metadata function such as Head, Tail, or Subset, then the query is not optimized. For example, querying for half of the Leaves set reduces performance to about the same as for the nonempty Descendants function call.
- Leaves() is recommended for use on large, sparse dimensions. In general, use Leaves() to optimize performance when the input set contains 10,000 members or more. For smaller, denser input sets, using the NON EMPTY keyword on an axis with CrossJoin might improve performance.

Example

The following examples are based on the Asosamp.Sample database.

Example 1 (Leaves)

The following query returns the Units (items per package) for all level 0 Personal Electronics products for which the Units data is not #MISSING:

SELECT
{Units} ON COLUMNS,
Leaves([Personal Electronics]) ON ROWS
FROM [Asosamp.Sample]

Because Leaves returns nonempty, level 0 descendants, the above query is identical to the following query:

SELECT
{Units} ON COLUMNS,
NON EMPTY Descendants([Personal Electronics], [Products].Levels(0), SELF)
ON ROWS
FROM [Asosamp.Sample]

These queries return the following grid:

Table 6-106  Output Grid from MDX Example

<table>
<thead>
<tr>
<th>(axis)</th>
<th>Items Per Package</th>
</tr>
</thead>
<tbody>
<tr>
<td>Digital Cameras</td>
<td>3041</td>
</tr>
<tr>
<td>Camcorders</td>
<td>3830</td>
</tr>
<tr>
<td>Photo Printers</td>
<td>6002</td>
</tr>
<tr>
<td>Memory</td>
<td>23599</td>
</tr>
<tr>
<td>Other Accessories</td>
<td>117230</td>
</tr>
<tr>
<td>Boomboxes</td>
<td>10380</td>
</tr>
<tr>
<td>Radios</td>
<td>20009</td>
</tr>
</tbody>
</table>

[Handhelds] was omitted from the result set because it has a value of #MISSING for the measure Units.

Example 2 (Leaves)

For this example, a third hierarchy called [Small Items] was added to the Products dimension.

- Products
  - Multiple Hierarchies Enabled <3> (Label Only)
  - All Merchandise Stored # Default # (+) <3>
    - Personal Electronics (+) <3>
      + Digital Cameras/Camcorders (+) <3>
    - Handhelds/PDAs (+) <3>
      - Handhelds (+)
      - Memory (+)
      - Other Accessories (+)
      + Portable Audio (+) <2>
    - Home Entertainment (+) <2>
    - Other (+) <1>
  - High End Merchandise Stored # Default # (~) <4>
  - Small Items Stored # Default # (~) <3>
    - Digital Cameras (+) (Shared Member)
    - Camcorders (+) (Shared Member)
    - Handhelds/PDAs (+) (Shared Member)
The following query

```
SELECT
  {Units} ON COLUMNS,
Leaves {[Small Items]} ON ROWS
FROM [Asosamp.Sample]
```

Returns the the following grid:

<table>
<thead>
<tr>
<th>(axis)</th>
<th>Items Per Package</th>
</tr>
</thead>
<tbody>
<tr>
<td>Digital Cameras</td>
<td>3041</td>
</tr>
<tr>
<td>Camcorders</td>
<td>3830</td>
</tr>
<tr>
<td>Memory</td>
<td>23599</td>
</tr>
<tr>
<td>Other Accessories</td>
<td>117230</td>
</tr>
</tbody>
</table>

In addition to the primary members [Digital Cameras] and [Camcorders], Leaves also returned the primary members [Memory] and [Other Accessories], because these level-0 members contributed to [Small Items] via [Handhelds/PDAs].

**Left**

Returns a specified number (length) of characters from the left side of the string.

**Syntax**

```
Left ( string ,length )
```

**Parameters**

- **string**
  Input string.

- **length**
  The number of characters to return from the left side of the input string.

**Example**

```
Left ("Northwind", 5)
```

returns North.

**Len**

Returns length of a string in terms of number of characters.

**Syntax**

```
Len ( string )
```
Parameters
string
A string.

Level

Returns the level of the input member.

Syntax

member.Level

Parameters

member
A member specification.

Example

The following query

SELECT
    [Year].[Qtr1].Level.Members
ON COLUMNS,
    [Product].Levels(0).Members
ON ROWS
FROM Sample.Basic

returns the grid:

<table>
<thead>
<tr>
<th>(axis)</th>
<th>Qtr1</th>
<th>Qtr2</th>
<th>Qtr3</th>
<th>Qtr4</th>
</tr>
</thead>
<tbody>
<tr>
<td>100-10</td>
<td>5096</td>
<td>5892</td>
<td>6583</td>
<td>5206</td>
</tr>
<tr>
<td>100-20</td>
<td>1359</td>
<td>1534</td>
<td>1528</td>
<td>1287</td>
</tr>
<tr>
<td>100-30</td>
<td>593</td>
<td>446</td>
<td>400</td>
<td>544</td>
</tr>
<tr>
<td>200-10</td>
<td>1697</td>
<td>1734</td>
<td>1883</td>
<td>1887</td>
</tr>
<tr>
<td>200-20</td>
<td>2963</td>
<td>3079</td>
<td>3149</td>
<td>2834</td>
</tr>
<tr>
<td>200-30</td>
<td>1153</td>
<td>1231</td>
<td>1159</td>
<td>1093</td>
</tr>
<tr>
<td>200-40</td>
<td>908</td>
<td>986</td>
<td>814</td>
<td>1384</td>
</tr>
<tr>
<td>300-10</td>
<td>2544</td>
<td>3231</td>
<td>3355</td>
<td>3065</td>
</tr>
<tr>
<td>300-20</td>
<td>690</td>
<td>815</td>
<td>488</td>
<td>518</td>
</tr>
<tr>
<td>300-30</td>
<td>2695</td>
<td>2723</td>
<td>2855</td>
<td>2820</td>
</tr>
<tr>
<td>400-10</td>
<td>2838</td>
<td>2998</td>
<td>3201</td>
<td>2807</td>
</tr>
<tr>
<td>400-20</td>
<td>2283</td>
<td>2522</td>
<td>2642</td>
<td>2404</td>
</tr>
<tr>
<td>400-30</td>
<td>-116</td>
<td>-84</td>
<td>-145</td>
<td>-49</td>
</tr>
<tr>
<td>100-20</td>
<td>1359</td>
<td>1534</td>
<td>1528</td>
<td>1287</td>
</tr>
<tr>
<td>200-20</td>
<td>2963</td>
<td>3079</td>
<td>3149</td>
<td>2834</td>
</tr>
<tr>
<td>300-30</td>
<td>2695</td>
<td>2723</td>
<td>2855</td>
<td>2820</td>
</tr>
</tbody>
</table>
See Also
Generation
Levels
IsLevel

Levels

Returns the level specified by the input level number.

Syntax

\[ \text{dimension}.\text{Levels\, (\,index\,)} \]

\[ \text{Levels\, (\,dimension,\,index\,)\,} \]

Parameters

**dimension**
The dimension specification.

**index**
The number of steps up from the lowest level-0 member of the dimension. The count begins with zero at leaf members.

Example

The following query

\[
\text{SELECT}\ 
\text{[Year].[Qtr1].Level.Members}\ 
\text{ON COLUMNS,}\ 
\text{[Product].Levels(0).Members}\ 
\text{ON ROWS}\ 
\text{FROM Sample.Basic}
\]

returns the grid:

<table>
<thead>
<tr>
<th>(axis)</th>
<th>Qtr1</th>
<th>Qtr2</th>
<th>Qtr3</th>
<th>Qtr4</th>
</tr>
</thead>
<tbody>
<tr>
<td>100-10</td>
<td>5096</td>
<td>5892</td>
<td>6583</td>
<td>5206</td>
</tr>
<tr>
<td>100-20</td>
<td>1359</td>
<td>1534</td>
<td>1528</td>
<td>1287</td>
</tr>
<tr>
<td>100-30</td>
<td>593</td>
<td>446</td>
<td>400</td>
<td>544</td>
</tr>
<tr>
<td>200-10</td>
<td>1697</td>
<td>1734</td>
<td>1883</td>
<td>1887</td>
</tr>
<tr>
<td>200-20</td>
<td>2963</td>
<td>3079</td>
<td>3149</td>
<td>2834</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>300-30</td>
<td>2695</td>
<td>2723</td>
<td>2855</td>
<td>2820</td>
</tr>
</tbody>
</table>
See Also
Level
Generations

LinkMember

Returns a member's shared member along a given hierarchy.
This function can be used instead of passing hierarchy arguments to Parent, Ancestor, FirstSibling, and LastSibling functions. This function works well in conjunction with Is* functions such as IsAncestor, IsChild, IsSibling, IsLevel, IsGeneration, and IsLeaf.

Syntax

member.LinkMember(hierarchy)

LinkMember(member, hierarchy)

Parameters

member
A member specification

hierarchy
Optional. A specific hierarchy within the time dimension.

Notes

• This function is applicable only to aggregate storage databases.
• If the primary hierarchy is passed to this function, it returns the primary member.
• If there is no shared member along the given hierarchy, this function returns an empty member.
• If a calculated member is passed to this function, the calculated member itself is returned.

Example

The following examples are based on ASOSamp.Sample.
The following MDX returns the member [HDTV] along the [High End Merchandise] hierarchy. By default, the primary instance of [HDTV] is used.

LinkMember([HDTV], [High End Merchandise])

The following MDX also returns the member [HDTV] along the [High End Merchandise] hierarchy. In this example, the input member is on the input hierarchy.

LinkMember([High End Merchandise].[HDTV], [High End Merchandise])
The following MDX returns the member [HDTV] along the [All Merchandise] hierarchy.

```
LinkMember([All Merchandise].[HDTV], [All Merchandise])
```

The following MDX returns an empty member, because there is no instance of [Digital Cameras] along the [High End Merchandise] hierarchy. The empty member has a value of #MISSING.

```
LinkMember([Digital Cameras], [High End Merchandise])
```

The following MDX also returns an empty member.

```
LinkMember([All Merchandise], [High End Merchandise])
```

The following MDX also returns an empty member.

```
LinkMember([Products], [High End Merchandise])
```

The following MDX returns [High End Merchandise].

```
LinkMember([High End Merchandise], [High End Merchandise])
```

---

**Ln**

Returns the natural logarithm (base e) of an expression.

**Syntax**

```
Ln ( numeric_value_expression )
```

**Parameters**

- `numeric_value_expression` A numeric value (see MDX Grammar Rules).

**Notes**

- Ln returns the inverse of Exp.
- The constant e is the base of the natural logarithm. e is approximately 2.71828182845904.

**Example**

```
WITH MEMBER [Measures].[Ln_Sales] AS 'Ln([Measures].[Sales])' 
SELECT 
([Year].levels(0).members) ON COLUMNS, 
([Measures].[Sales], [Measures].[Ln_Sales]) ON ROWS
```

---
ON ROWS
FROM
Sample.Basic
WHERE
([Market].[East], [Product].[Cola])

returns the following grid:

**Table 6-110 Output Grid from MDX Example**

<table>
<thead>
<tr>
<th>(axis)</th>
<th>Jan</th>
<th>Feb</th>
<th>...</th>
<th>Nov</th>
<th>Dec</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sales</td>
<td>1812</td>
<td>1754</td>
<td></td>
<td>1708</td>
<td>1841</td>
</tr>
<tr>
<td>Ln_Sales</td>
<td>7.502</td>
<td>7.470</td>
<td></td>
<td>7.443</td>
<td>7.518</td>
</tr>
</tbody>
</table>

See Also

Log
Log10
Exp

Log

Returns the logarithm of an expression to a specified base.

Syntax

Log ( numeric_value_expression [,base] )

Parameters

**numeric_value_expression**
A numeric value or an expression that returns a numeric value (see MDX Grammar Rules).

**base**
Optional. A number representing the base to use for the logarithm. If less than zero, zero, or close to 1, the Log function returns #MISSING. If omitted, the Log function calculates the base-10 logarithm. Log (Sales, 10) is equivalent to Log(Sales), and is also equivalent to Log10(Sales).

Example

Log(9,3) returns 2.

Log10

Returns the base-10 logarithm of an expression.

Syntax

Log10 ( numeric_value_expression )
Parameters

`numeric_value_expression`
A numeric value or an expression that returns a numeric value (see MDX Grammar Rules).

Example

`Log10(1000)` returns 3.

Lower

Converts upper-case string to lower-case.

Syntax

```
Lower ( string )
```

Parameters

`string`
Input string.

Example

```
Lower(STRING)
```

returns `string`

See Also

Upper

LTrim

Trims all whitespace on the left side of the string.

Syntax

```
LTrim ( string )
```

Parameters

`string`
Input string.

Example

```
LTrim(" STRING")
```

returns "STRING"
Max

Returns the maximum of values found in the tuples of a set.

Syntax

\[ \text{Max} \left( \text{set} [,, \text{numeric\_value\_expression}] \right) \]

Parameters

\textit{set}

The set to search for values.

\textit{numeric\_value\_expression}

Optional numeric value expression (see MDX Grammar Rules).

Notes

The return value of Max is #MISSING if either of the following is true:

- The input set is empty.
- All tuple evaluations result in #MISSING values.

Example

WITH

MEMBER [Measures].[Max Qtr2 Sales] AS

'Max {
   ([Year].[Qtr2]),
   [Measures].[Sales]
}

SELECT

{ [Measures].[Max Qtr2 Sales] } on columns,
{ [Product].children } on rows
FROM Sample.Basic

<table>
<thead>
<tr>
<th>(axis)</th>
<th>Max Qtr2 Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>Colas</td>
<td>27187</td>
</tr>
<tr>
<td>Root Beer</td>
<td>27401</td>
</tr>
<tr>
<td>Cream Soda</td>
<td>25736</td>
</tr>
<tr>
<td>Fruit Soda</td>
<td>21355</td>
</tr>
<tr>
<td>Diet Drinks</td>
<td>26787</td>
</tr>
</tbody>
</table>

Median

Orders the set according to the numeric value expression, and then returns the value of the set's median tuple.
Syntax

Median ( set, numeric_value_expr )

Parameters

set
The set from which to get a median tuple value.

numeric_value_expr
A numeric value or an expression that returns a numeric value.

Notes

This function is a special case of the Percentile function where \( n = 50 \).

Example

The following query returns the median price for radios paid in all states last year.

```sql
WITH MEMBER
  [Geography].[Median Mkt Price]
AS
  'Median ( [Geography].Levels(2).Members, [Measures].[Price Paid])'
SELECT
  { [Geography].[Median Mkt Price] } ON COLUMNS
FROM
  ASOSamp.Sample
WHERE ([Products].[Radios], [Years].[Prev Year])

MemberRange

Using the order of members existing in a database outline, returns a range of members inclusive of and between two members in the same generation or level.

Syntax

MemberRange ( member1, member2 [, layertype] [, hierarchy ] )

member1:member2

Parameters

member1
The beginning point of the member range.

member2
The endpoint of the member range.
**layertype**
GENERATION or LEVEL. Available only with function-style `MemberRange()` syntax. If omitted or if operator-style `member:member` syntax is used, the range of members returned is inclusive of and between two specified members of the same generation. If `MemberRange(member, member, LEVEL)` is used, the range of members returned is inclusive of and between two specified members of the same level.

**hierarchy**
Optional. A specific hierarchy within the time dimension.

**Notes**
- If the two input members are not from the same generation or level, the result is an empty set.
- If the two input members are not from the same dimension, an error is returned.
- The order of the output resembles the order of the input. See Example 2.
- If the hierarchy argument is passed, `member1` and `member2` should belong to the same hierarchy. Otherwise, an empty set is returned.
- When multiple hierarchies are enabled, this function returns NULL when the range begins in one hierarchy and terminates in another hierarchy.

**Example**

**Example 1 (MemberRange)**
The following set:

```
{ [Year].[Qtr1], [Year].[Qtr2], [Year].[Qtr3], [Year].[Qtr4] }
```

is returned by both of the following examples:

```
MemberRange ( [Year].[Qtr1], [Year].[Qtr4] )
```

```
( [Year].[Qtr1] : [Year].[Qtr4] )
```

**Example 2 (MemberRange)**

```
[Jan] : [Mar]
```

returns:

```
{ [Jan], [Feb], [Mar] }
```

```
[Mar] : [Jan]
```

returns:

```
{ [Mar], [Feb], [Jan] }
```
Example 3 (MemberRange)

The following query

```sql
SELECT
    {[Measures].[Sales], [Measures].[Profit]}
ON COLUMNS,
    MemberRange([Year].[Feb], [Year].[Nov])
ON ROWS
FROM Sample.Basic
```

returns the grid:

### Table 6-112  Output Grid from MDX Example

<table>
<thead>
<tr>
<th>(axis)</th>
<th>Sales</th>
<th>Profit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feb</td>
<td>32069</td>
<td>8346</td>
</tr>
<tr>
<td>Mar</td>
<td>32213</td>
<td>8333</td>
</tr>
<tr>
<td>Apr</td>
<td>32917</td>
<td>8644</td>
</tr>
<tr>
<td>May</td>
<td>33674</td>
<td>8929</td>
</tr>
<tr>
<td>Jun</td>
<td>35088</td>
<td>9534</td>
</tr>
<tr>
<td>Jul</td>
<td>36134</td>
<td>9878</td>
</tr>
<tr>
<td>Aug</td>
<td>36008</td>
<td>9545</td>
</tr>
<tr>
<td>Sep</td>
<td>33073</td>
<td>8489</td>
</tr>
<tr>
<td>Oct</td>
<td>32828</td>
<td>8653</td>
</tr>
<tr>
<td>Nov</td>
<td>31971</td>
<td>8367</td>
</tr>
</tbody>
</table>

See Also

**RelMemberRange**

Members

Returns all members of the specified dimension or layer.

**Syntax**

```
dimension.Members | Members ( dimension )
```

```
layer.Members | Members ( layer )
```

**Parameters**

**dimension**

A dimension specification.

**layer**

A layer specification.
Example

This example focuses on the following part of the Sample Basic outline:

```
- Market
  - East
  - West
  - South
  - Central
```

The following expression:

```
{([Market].members)}
```

returns the following set, which includes all descendant members of the Market dimension:

```
{Market, [New York], Massachusetts, Florida, Connecticut, [New Hampshire], East, California, Oregon, Washington, Utah, Nevada, West, Texas, Oklahoma, Louisiana, [New Mexico], South, Illinois, Ohio, Wisconsin, Missouri, Iowa, Colorado, Central}
```

The following expression:

```
{([Market].levels(1).members)}
```

returns the following set, which includes one level of descendant members of the Market dimension:

```
{East, West, South, Central}
```

The following query assumes that level 1 of the Market dimension has an alias of Region:

```
Select
  {{ [Market].[Region].members } }
on columns
from Sample.Basic
```

This query returns the following grid:
Table 6-113  Output Grid from MDX Example

<table>
<thead>
<tr>
<th></th>
<th>East</th>
<th>West</th>
<th>South</th>
<th>Central</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>24161</td>
<td>29861</td>
<td>13238</td>
<td>38262</td>
</tr>
</tbody>
</table>

Min

Returns the minimum of values found in the tuples of a set.

Syntax

Min ( set [, numeric_value_expression ] )

Parameters

set
The set to search for values.

numeric_value_expression
Optional numeric value expression (see MDX Grammar Rules).

Notes

The return value of Min is #MISSING if either of the following is true:

- The input set is empty.
- All tuple evaluations result in #MISSING values.

Example

For every quarter, the following query displays the minimum monthly sales value.

WITH MEMBER
   [Measures].[Minimum Sales in Quarter]
AS
   'Min ([Year].CurrentMember.Children, [Measures].[Sales])'
SELECT
   {[Measures].[Minimum Sales in Quarter]}
ON COLUMNS,
   [Year].Children
ON ROWS
FROM Sample.Basic

This query returns the grid:

Table 6-114  Output Grid from MDX Example

<table>
<thead>
<tr>
<th>(axis)</th>
<th>Minimum Sales in Quarter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Qtr1</td>
<td>31538</td>
</tr>
<tr>
<td>Qtr2</td>
<td>32917</td>
</tr>
<tr>
<td>Qtr3</td>
<td>33073</td>
</tr>
</tbody>
</table>
Table 6-114  (Cont.) Output Grid from MDX Example

<table>
<thead>
<tr>
<th>(axis)</th>
<th>Minimum Sales in Quarter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Qtr4</td>
<td>31971</td>
</tr>
</tbody>
</table>

Mod

Returns the modulus (remainder value) of a division operation.

Syntax

Mod ( numeric_value_expr_1, numeric_value_expr_2 )

Parameters

numeric_value_expr_1
The number for which to find the remainder. Must be a numeric value or an expression that returns a numeric value (see MDX Grammar Rules).

numeric_value_expr_2
The divisor. Must be a numeric value or an expression that returns a numeric value (see MDX Grammar Rules).

Notes

The Essbase implementation of the function Mod returns the following values, which may be different from other vendors' implementations:

Mod(n,k) = - Mod(-n,k) , where n < 0
Mod(n,k) =  Mod(n,-k) , where k < 0

Example

WITH MEMBER [Measures].[Factor] AS
'Mod ([Measures].[Margin %],[Measures].[Profit %])'
SELECT
{
  [Measures].[Margin %],
  [Measures].[Profit %],
  [Measures].[Factor]
}
ON COLUMNS,
{[Year].[Qtr1].Children}
ON ROWS
FROM sample.basic

returns:
Table 6-115  Output Grid from MDX Example

<table>
<thead>
<tr>
<th>(axis)</th>
<th>Margin %</th>
<th>Profit %</th>
<th>Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jan</td>
<td>55.102</td>
<td>25.44</td>
<td>4.217</td>
</tr>
<tr>
<td>Feb</td>
<td>55.387</td>
<td>26.025</td>
<td>3.337</td>
</tr>
<tr>
<td>Mar</td>
<td>55.267</td>
<td>25.868</td>
<td>3.530</td>
</tr>
</tbody>
</table>

NextMember

Using the order of members existing in a database outline, returns the next member along the same generation or level.

**Syntax**

```
member.NextMember [( layertype ) ]
```

**Parameters**

- **member**
  The starting member from which .NEXTMEMBER counts one member forward.

- **layertype**
  GENERATION or LEVEL. The default is Generation.

**Notes**

- If the next member is not found, this function returns an empty member. For example, using Sample Basic, these would return an empty member:
  Qtr4.nextmember and Year.nextmember.

- When multiple hierarchies are enabled, this function returns NULL when the source member is in one hierarchy and the result member belongs to a different hierarchy.

**Example**

**Example 1**

The following expression:

```
[Jun].nextmember
```

returns the member that is one step further than Jun:

```
[Jul]
```
Example 2

The following query

/*
For January, PrevMember doesn't exist
For December, NextMember doesn't exist
*/

WITH

MEMBER [Measures].[Delta from Previous Month]
AS
  ' [Measures].[Sales] -
  ([Measures].[Sales],[Year].CurrentMember.PrevMember)
 ,

MEMBER [Measures].[Delta from Next Month]
AS
  ' [Measures].[Sales] -
  ([Measures].[Sales],[Year].CurrentMember.NextMember)
 ,

SELECT
  { [Measures].[Sales],
    [Measures].[Delta from Previous Month],
    [Measures].[Delta from Next Month]
  }
ON COLUMNS,

[Year].Levels(0).Members
ON ROWS

FROM Sample.Basic
WHERE
  ( [Scenario].[Actual],
    [Market].[East],
    [Product].[100] )

returns the grid:

Table 6-116  Output Grid from MDX Example

<table>
<thead>
<tr>
<th>(axis)</th>
<th>Sales</th>
<th>Delta from Previous Month</th>
<th>Delta from Next Month</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jan</td>
<td>2105</td>
<td>2105</td>
<td>44</td>
</tr>
<tr>
<td>Feb</td>
<td>2061</td>
<td>-44</td>
<td>-65</td>
</tr>
<tr>
<td>Mar</td>
<td>2126</td>
<td>65</td>
<td>-132</td>
</tr>
<tr>
<td>Apr</td>
<td>2258</td>
<td>132</td>
<td>-89</td>
</tr>
</tbody>
</table>
Table 6-116  (Cont.) Output Grid from MDX Example

<table>
<thead>
<tr>
<th>(axis)</th>
<th>Sales</th>
<th>Delta from Previous Month</th>
<th>Delta from Next Month</th>
</tr>
</thead>
<tbody>
<tr>
<td>May</td>
<td>2347</td>
<td>89</td>
<td>-278</td>
</tr>
<tr>
<td>Jun</td>
<td>2625</td>
<td>278</td>
<td>-110</td>
</tr>
<tr>
<td>Jul</td>
<td>2735</td>
<td>110</td>
<td>62</td>
</tr>
<tr>
<td>Aug</td>
<td>2673</td>
<td>-62</td>
<td>311</td>
</tr>
<tr>
<td>Sep</td>
<td>2362</td>
<td>-311</td>
<td>268</td>
</tr>
<tr>
<td>Oct</td>
<td>2094</td>
<td>-268</td>
<td>28</td>
</tr>
<tr>
<td>Nov</td>
<td>2066</td>
<td>-28</td>
<td>-222</td>
</tr>
<tr>
<td>Dec</td>
<td>2288</td>
<td>222</td>
<td>2288</td>
</tr>
</tbody>
</table>

See Also

PrevMember

Lead

NonEmptyCount

Returns the count of the number of tuples in a set that evaluate to non-#Missing values. Each tuple is evaluated and included in the count returned by this function. If the numeric value expression is specified, it is evaluated in the context of every tuple, and the count of non-#Missing values is returned.

On aggregate storage databases, the NonEmptyCount MDX function is optimized so that the calculation of the distinct count for all cells can be performed by scanning the database only once. Without this optimization, the database is scanned as many times as the number of cells corresponding to the distinct count. The NONEMPTYCOUNT optimization is triggered when an outline member formula has the following syntax:

NONEMPTYCOUNT(set, measure, exclude_missing)

Syntax

NonEmptyCount { set [, numeric_value_expression [, exclude_missing ] ] }

Parameters

set
The set in which to count tuples.

numeric_value_expression
Optional. (See MDX Grammar Rules.)

exclude_missing
Optional. A flag that indicates that the count value returned is missing when the Measure value is missing for members in Set.

Where:
• **Set**: Is a one dimensional set from a stored dimension.

• **Measure**: Is a stored measure.

The **exclude_missing** parameter supports the NonEmptyCount optimization on aggregate databases by improving the performance of a query that queries metrics that perform a distinct count calculation. See Example 2 in this topic for more information.

By default, a value of zero is returned when the **Measure** value is missing for all members in the **Set**.

**Example**

**Example 1**

The following query

```
With
Member [Measures].[Number Of Markets] as 'NonEmptyCount (Market.Levels(0).Members, Sales)'

Select
{[Measures].[Number Of Markets]} on Columns,
{[100].Children, [200].Children} on Rows
FROM Sample.Basic
```

Returns the grid:

**Table 6-117  Output Grid from MDX Example**

<table>
<thead>
<tr>
<th>(axis)</th>
<th>Number of Markets</th>
</tr>
</thead>
<tbody>
<tr>
<td>100-10</td>
<td>20</td>
</tr>
<tr>
<td>100-20</td>
<td>16</td>
</tr>
<tr>
<td>100-30</td>
<td>8</td>
</tr>
<tr>
<td>200-10</td>
<td>20</td>
</tr>
<tr>
<td>200-20</td>
<td>17</td>
</tr>
<tr>
<td>200-30</td>
<td>9</td>
</tr>
<tr>
<td>200-40</td>
<td>3</td>
</tr>
</tbody>
</table>

**Example 2**

In an aggregate storage database, it is common to count the distinct number of entities (such as customers and products). You can perform a distinct count by defining a formula member or a calculated member. For example, you can add a formula member, [DistinctCustomerCnt], to use with the following formula to calculate the count of distinct customers who bought a Product.

```
NONEMPTYCOUNT(Customer.Levels(0).Members, [Units])
```
The following MDX query scans the database as many times as the number of Products, evaluating the distinct customer count for each Product separately:

```
SELECT
    {[DistinctCustomerCnt]} on COLUMNS,
    Products.Levels(0).Members on ROWS
```

**NonEmptySubset**

Given an input set, `NonEmptySubset` returns a subset of that input set in which all tuples evaluate to nonempty. An optional value expression may be specified for the nonempty check.

This function can help optimize queries that are based on a large set for which the set of nonempty combinations is known to be small. `NonEmptySubset` reduces the size of the set in the presence of a metric; for example, you might request the nonempty subset of descendants for specific Units.

`NonEmptySubset` is used to reduce the size of a set before a subsequent analytical retrieval.

**Syntax**

```
NonEmptySubset (set [, value_expression [, dimension...]])
```

**Parameters**

- **set**
  - The set to reduce

- **value_expression**
  - A value expression--ideally, a stored member or a simple formula. For each tuple in `set`, if `value_expression` is nonempty, the tuple is returned as part of the subset. Otherwise, it is removed.

- **dimension**
  - One or more (comma-separated) dimensions from which to return the non-empty subset

**Notes**

`Value_expression`, if used, should be a stored member or simple formula. If `value_expression` is a complex formula, the retrieval of the nonempty subset is not optimized.

**Example**

The following example gets the bottom 10 products in terms of Units (items per package), and then returns the CrossJoin of that set and the level 0 members (zip codes) of [Albany - NY].

```
WITH SET Bottom_10 AS 'BottomCount (Leaves(Products),

```

6-231
10, Units

SELECT
  {Units}
ON COLUMNS,
  NonEmptySubset(CrossJoin(Bottom_10, Leaves([Albany - NY])))
ON ROWS
FROM Asosamp.Sample

This query returns the following grid:

**Table 6-118  Output Grid from MDX Example**

<table>
<thead>
<tr>
<th>(axis)</th>
<th>Items Per Package</th>
</tr>
</thead>
<tbody>
<tr>
<td>Digital Cameras,12201</td>
<td>4</td>
</tr>
<tr>
<td>Camcorders,12201</td>
<td>3</td>
</tr>
<tr>
<td>Photo Printers, 12201</td>
<td>2</td>
</tr>
<tr>
<td>Digital Recorders, 12201</td>
<td>2</td>
</tr>
<tr>
<td>Desktops,12201</td>
<td>3</td>
</tr>
<tr>
<td>Digital Cameras,12212</td>
<td>5</td>
</tr>
<tr>
<td>Camcorders,12212</td>
<td>2</td>
</tr>
<tr>
<td>Photo Printers, 12212</td>
<td>3</td>
</tr>
<tr>
<td>Flat Panel, 12212</td>
<td>1</td>
</tr>
<tr>
<td>HDTV,12212</td>
<td>1</td>
</tr>
<tr>
<td>Home Theater, 12212</td>
<td>1</td>
</tr>
<tr>
<td>Desktops, 12212</td>
<td>2</td>
</tr>
<tr>
<td>Notebooks,12212</td>
<td>1</td>
</tr>
<tr>
<td>Digital Cameras,12223</td>
<td>1</td>
</tr>
<tr>
<td>Camcorders,12223</td>
<td>1</td>
</tr>
<tr>
<td>Photo Printers,12223</td>
<td>4</td>
</tr>
<tr>
<td>HTDV,12223</td>
<td>1</td>
</tr>
<tr>
<td>Notebooks,12223</td>
<td>1</td>
</tr>
<tr>
<td>Camcorders,12229</td>
<td>4</td>
</tr>
<tr>
<td>HDTV,12229</td>
<td>1</td>
</tr>
<tr>
<td>Home Theater,12229</td>
<td>3</td>
</tr>
<tr>
<td>Desktops,12229</td>
<td>1</td>
</tr>
<tr>
<td>Digital Cameras,12249</td>
<td>2</td>
</tr>
<tr>
<td>Photo Printers,12249</td>
<td>3</td>
</tr>
<tr>
<td>Projection TVs,12249</td>
<td>1</td>
</tr>
<tr>
<td>HDTV,12249</td>
<td>2</td>
</tr>
<tr>
<td>Home Theater,12249</td>
<td>1</td>
</tr>
<tr>
<td>Digital Recorders,12249</td>
<td>1</td>
</tr>
<tr>
<td>Notebooks,12249</td>
<td>1</td>
</tr>
<tr>
<td>Camcorders,12257</td>
<td>2</td>
</tr>
<tr>
<td>Photo Printers,12257</td>
<td>4</td>
</tr>
<tr>
<td>Projection TVs,12257</td>
<td>2</td>
</tr>
<tr>
<td>HDTV,12257</td>
<td>1</td>
</tr>
</tbody>
</table>
NTile

Returns a division number of a tuple in a set. This function only applies to aggregate storage databases.

Syntax

\[
\text{NTile ( member_or_tuple, set, number_of_divisions, numeric_value_expr )}
\]

Parameters

**member_or_tuple**

A member or a tuple.

**set**

The set to order.

**number_of_divisions**

The number of divisions to use in ordering the set.

**numeric_value_expr**

A numeric value or an expression that returns a numeric value.

Notes

- This function is applicable only to aggregate storage databases.
- This function orders the set by a numeric value, divides it into \( n \) equal divisions, and returns the division number that the given tuple is in.

Example

WITH

 MEMBER [Measures].[7tile] AS
   '"NTile
      ([Measures].[Price Paid],
       { [Products].Levels(0).Members },
       7,
       [Measures].[Price Paid]
     )'
 SELECT
 { [Measures].[Price Paid], [Measures].[7tile] } on columns,
 { [Products].Levels(0).Members } on rows
 FROM ASOSamp.Sample
NumToStr

Converts a double-precision floating-point value into a decimal string. The number is formatted according to locale-specific conventions.

Syntax

\`NumToStr (numeric_value_expression)\`

Parameters

\textit{numeric_value_expression}

Numeric value expression (see MDX Grammar Rules).

Example

\`NumToStr(1)\`

returns "1.00".

OpeningPeriod

Returns the first descendant of a layer, or the first child of the Time dimension.

Syntax

\`OpeningPeriod ([layer [,member]])\`

Parameters

\textit{layer}

A layer specification. If omitted, the first descendant of \textit{member} is used. If \textit{member} is omitted, the first child of the Time dimension is assumed.

\textit{member}

Optional. A member specification. If omitted, the first child of the Time dimension is assumed (for example, \textit{Qtr1} in Sample Basic).

Notes

The return value of this function varies depending on the input.

1. When no arguments are specified, the input member is assumed to be the current member of the Time dimension, and \textit{OpeningPeriod} returns the first child of that member. Do not use this function without arguments if there is no dimension tagged as Time.

2. When both \textit{layer} and \textit{member} arguments are given as input, \textit{OpeningPeriod} returns the first descendant of the input member at the input layer. For example, \textit{OpeningPeriod(Year.generations(3), Qtr3)} returns Jul. If the input \textit{member} and \textit{layer} are the same layer, the output is the input member. For example, \textit{OpeningPeriod(Year.generations(3), Jul)} returns Jul.
3. When only the layer argument is specified, the input member is assumed to be the current member of the dimension used in the layer argument. Openingperiod returns the first descendant of that dimension, at the input layer. For example, Openingperiod(Year.generations(3)) returns Oct.

See Also
ClosingPeriod
LastPeriods
ParallelPeriod
PeriodsToDate

Order

Sorts members of a set in order based on an expression.

Syntax

Order ( set, string_expr | numeric_value_expression [,BASC | BDESC] )

Parameters

set
The set to sort.

string_expr
String sorting criteria.

t Numeric_value_expression
Numeric sorting criteria (see MDX Grammar Rules).

BASC
If this keyword is used, the returned set is arranged in ascending order. Ascending order is the default even if no keyword is used.

BDESC
If this keyword is used, the returned set is arranged in descending order.

Notes
This function ignores missing values.

Example
The following query displays budgeted Sales and Marketing in Qtr2, and the display of products is sorted based on ascending Actual Sales in Qtr1.

SELECT
   CrossJoin(
      {[Scenario].[Budget]},
      {[Measures].[Marketing], [Measures].[Sales]}
   )
ON COLUMNS,
   Order(
[Product].Levels(0).Members,
   ([Year].[Qtr1], [Scenario].[Actual])
)
ON ROWS
FROM Sample.Basic
WHERE ([Year].[Qtr2])

This query returns the grid:

**Table 6-119  Output Grid from MDX Example**

<table>
<thead>
<tr>
<th></th>
<th>Budget</th>
<th>Budget</th>
</tr>
</thead>
<tbody>
<tr>
<td>(axis)</td>
<td>(axis)</td>
<td></td>
</tr>
<tr>
<td>400-30</td>
<td>510</td>
<td>3240</td>
</tr>
<tr>
<td>100-30</td>
<td>450</td>
<td>3400</td>
</tr>
<tr>
<td>300-20</td>
<td>550</td>
<td>3800</td>
</tr>
<tr>
<td>200-40</td>
<td>310</td>
<td>2830</td>
</tr>
<tr>
<td>200-30</td>
<td>550</td>
<td>4060</td>
</tr>
<tr>
<td>100-20</td>
<td>1160</td>
<td>8800</td>
</tr>
<tr>
<td>100-20</td>
<td>1160</td>
<td>8800</td>
</tr>
<tr>
<td>200-10</td>
<td>2090</td>
<td>10330</td>
</tr>
<tr>
<td>400-20</td>
<td>880</td>
<td>6590</td>
</tr>
<tr>
<td>300-10</td>
<td>1450</td>
<td>10080</td>
</tr>
<tr>
<td>300-30</td>
<td>1080</td>
<td>7880</td>
</tr>
<tr>
<td>100-10</td>
<td>1800</td>
<td>17230</td>
</tr>
</tbody>
</table>

**Ordinal**

Returns a generation number or level number.

**Syntax**

```
Ordinal ( layer )
```

**Parameters**

**layer**

A layer specification for which to determine the ordinal.

**Example**

The following example prints generation number and level number for each member in the Product dimension. The value of calculated member [ProdGen] is a generation number because the input argument to the Ordinal function is a generation. The value
of calculated member [ProdLev] is a level number because the input argument to the Ordinal function is a level.

WITH

MEMBER [Measures].[ProdGen] AS
  'Ordinal([Product].CurrentMember.Generation)'
MEMBER [Measures].[ProdLev] AS
  'Ordinal([Product].CurrentMember.Level)'

SELECT
  {[ProdGen], [ProdLev]} ON COLUMNS,
  [Product].Members ON ROWS
FROM Sample.Basic

This query returns the following grid:

Table 6-120  Output Grid from MDX Example

<table>
<thead>
<tr>
<th>(axis)</th>
<th>ProdGen</th>
<th>ProdLev</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>100</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>100-10</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>100-20</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>100-30</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>200</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>200-10</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>200-20</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>200-30</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>200-40</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>300</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>300-10</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>300-20</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>300-30</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>400</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>400-10</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>400-20</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>400-30</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>Diet</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>100-20</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>200-20</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>300-30</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

ParallelPeriod

Returns a member from a prior time period as the specified or default time member.

Syntax

ParallelPeriod ( [layer [, index [, member [, hierarchy ]]]] )
Parameters

layer
Optional layer specification. If omitted, the same layer is assumed.

index
Number of time periods to count back in the specified layer.

member
Optional member specification. If omitted, the default member is assumed (for more information, see DefaultMember).

hierarchy
Optional. A specific hierarchy within the time dimension.

Notes
If layer, index, and member are present, this function determines the member ANCESTOR1, which is computed as

\[ \text{Ancestor(member, layer)} \]

The member ANCESTOR2 is then computed as

\[ \text{Lag(ANCESTOR1, index)} \]

The return value of this function is then computed as

\[ \text{Cousin(member, ANCESTOR2)} \]

If layer and index are present and member is absent, member is taken to be the current member along the dimension associated with layer. The returned value is determined as above.

If only layer is present, index is taken to be 1, and member is taken to be the current member along the dimension associated with layer. The returned value is determined as above.

If layer, index, and member are all absent, member is taken to be CurrentMember along TIME Dimension, index is taken to be 1, and layer is taken to be the generation of the parent of member. The returned value is determined as above.

See Also
LastPeriods
PeriodsToDate
ClosingPeriod
OpeningPeriod

Parent

Returns a member's parent.
Syntax

\texttt{member.Parent ((hierarchy) )}

Parent ( \texttt{member [, hierarchy ]})

Parameters

\textbf{member}

A member specification.

\textbf{hierarchy}

Optional. A specific hierarchy within the time dimension.

Example

Example 1

\texttt{SELECT}
\texttt{\{Parent \{[100-10]\}\}}
\texttt{ON COLUMNS}
\texttt{FROM}
\texttt{sample.basic}

returns the parent of 100-10:

\begin{table}[h]
\centering
\begin{tabular}{|c|c|}
\hline
\textbf{100} & 30468 \\
\hline
\end{tabular}
\caption{Output Grid from MDX Example}
\end{table}

Example 2

The following query uses \texttt{Filter} to find the months in which Sales for [Product].[100] are higher than 8,570. The Parent function is used with \texttt{Generate} to create a set consisting of the parents (quarters) of the high-sales months.

\begin{verbatim}
WITH SET [High-Sales Months] as
' Filter(
  [Year].Levels(0).members,
  [Measures].[Sales] > 8570
 )',

SELECT
  [[Measures].[Sales]]
ON COLUMNS,
  Generate([High-Sales Months], ( Parent([Year].CurrentMember) ))
ON ROWS
FROM
  sample.basic
\end{verbatim}
WHERE
([Product].[100])

This query returns the grid:

<table>
<thead>
<tr>
<th>(axis)</th>
<th>Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>Qtr2</td>
<td>27187</td>
</tr>
<tr>
<td>Qtr3</td>
<td>28544</td>
</tr>
<tr>
<td>Qtr4</td>
<td>25355</td>
</tr>
</tbody>
</table>

Percentile

Orders the set according to the numeric value expression, and then returns the value of the tuple that is at the given percentile.

This function only applies to aggregate storage databases.

Syntax

Percentile ( set, numeric_value_expr, percentile )

Parameters

set
The set from which to get a tuple value.

numeric_value_expr
A numeric value or an expression that returns a numeric value.

percentile
A percentile. Must be between 0 and 100.

Notes

- This function is applicable only to aggregate storage databases.
- The returned value is such that \( n \) percent of the of the set members are smaller than it.

Example

WITH MEMBER [Measures].[Perc] AS
'Percentile(Products.Levels(0).Members, [Measures].[Price Paid], 10)'
SELECT ([Measures].[Price Paid], [Measures].[Perc]) ON COLUMNS,
{ Products.Levels(0).Members } ON ROWS
FROM AsoSamp.Sample
PeriodsToDate

Returns a set of single-member tuples from a specified layer up to a given member in that layer (or up to the default member), or, returns members up to the current member of the Time dimension.

Syntax

PeriodsToDate ( [layer [, member [, hierarchy ]]] )

Parameters

layer
The layer to use as a beginning point.

member
The member to use as an ending point.

hierarchy
Optional. A specific hierarchy within the time dimension.

Notes

• If layer and member are present, this function determines the ANCESTOR of member, computed as Ancestor(member, layer).

Consider the subtree rooted at the ANCESTOR. This function returns the set of all members along the same generation between the first descendant of ANCESTOR at input member's generation and the input member (inclusive of both.)

The return value of this function is the set of single-member tuples constructed from the members in the subtree rooted at ANCESTOR which are in the same layer as member and which are at or before the position of member within its layer. The order of tuples in the returned set is the same as the order of the members included in the input layer.

• If layer is present and member is absent, member is considered to be CurrentMember of the dimension that layer is associated with.

• If layer and member are both absent, member is considered to be the current member of the Time dimension, and layer is assumed to be the generation of the member's parent. Hence the return value is a set containing the left siblings of member and member itself.

• Using PeriodsToDate(layer, member) has the same effect as using the following nested functions:

MemberRange(
    OpeningPeriod(
        member.GENERATION,
        Ancestor (member, layer)
    )
    : member
)


Example
PeriodsToDate (Year.Generations(1), May) returns the set:
{ Jan, Feb, Mar, Apr, May }

PeriodsToDate (Year.Generations(2), May) returns the set:
{ Apr, May }

PeriodsToDate (Year.Generations(3), May) returns the set:
{ May }

See Also
OpeningPeriod
ClosingPeriod
ParallelPeriod
LastPeriods

Power
Returns the result of raising a number to a given power.

Syntax
Power ( numeric_value_expression, power )

Parameters
numeric_value_expression
An expression that returns a value (see MDX Grammar Rules).

power
The power to which the numeric value expression is raised.

Example
Power(9, 2.5) returns 243.

PrevMember
Using the order of members existing in a database outline, returns the previous member along the same generation or level.
Note:

When multiple hierarchies are enabled, this function returns NULL when the source member is in one hierarchy and the result member belongs to a different hierarchy.

Syntax

```
member.PrevMember [( layertype )]
```

```
PrevMember ( member [,layertype ])
```

Parameters

- **member**
  The starting member from which PrevMember counts one member back.

- **layertype**
  GENERATION or LEVEL. The default is Generation.

Example

Example 1

The following expression

```
[Jun].prevmember
```

returns the member that is 1 step prior to Jun:

```
[May]
```

Example 2

The following query

```
/*
For January, PrevMember doesn't exist
For December, NextMember doesn't exist
*/

WITH

MEMBER

[Measures].[Delta from Previous Month]
AS

'
[Measures].[Sales] -
(([Measures].[Sales],[Year].CurrentMember.PrevMember)
,'
MEMBER [Measures].[Delta from Next Month]
AS
' [Measures].[Sales] -
([Measures].[Sales], [Year].CurrentMember.NextMember)
',

SELECT
{ [Measures].[Sales],
  [Measures].[Delta from Previous Month],
  [Measures].[Delta from Next Month]
}
ON COLUMNS,
[Year].Levels(0).Members
ON ROWS
FROM Sample.Basic
WHERE
(
  [Scenario].[Actual],
  [Market].[East],
  [Product].[100]
)

Returns the grid:

**Table 6-123  Output Grid from MDX Example**

<table>
<thead>
<tr>
<th>(axis)</th>
<th>Sales</th>
<th>Delta from Previous Month</th>
<th>Delta from Next Month</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jan</td>
<td>2105</td>
<td>2105</td>
<td>44</td>
</tr>
<tr>
<td>Feb</td>
<td>2061</td>
<td>-44</td>
<td>-65</td>
</tr>
<tr>
<td>Mar</td>
<td>2126</td>
<td>65</td>
<td>-132</td>
</tr>
<tr>
<td>Apr</td>
<td>2258</td>
<td>132</td>
<td>-89</td>
</tr>
<tr>
<td>May</td>
<td>2347</td>
<td>89</td>
<td>-278</td>
</tr>
<tr>
<td>Jun</td>
<td>2625</td>
<td>278</td>
<td>-110</td>
</tr>
<tr>
<td>Jul</td>
<td>2735</td>
<td>110</td>
<td>62</td>
</tr>
<tr>
<td>Aug</td>
<td>2673</td>
<td>-62</td>
<td>311</td>
</tr>
<tr>
<td>Sep</td>
<td>2362</td>
<td>-311</td>
<td>268</td>
</tr>
<tr>
<td>Oct</td>
<td>2094</td>
<td>-268</td>
<td>28</td>
</tr>
<tr>
<td>Nov</td>
<td>2066</td>
<td>-28</td>
<td>-222</td>
</tr>
<tr>
<td>Dec</td>
<td>2288</td>
<td>222</td>
<td>2288</td>
</tr>
</tbody>
</table>

See Also

NextMember
Lag

Rank

Returns the numeric position of a tuple in a set.
Syntax

Rank ( member_or_tuple, set [, numeric_value_expr [, ORDINALRANK | DENSERANK | PERCENTRANK ]] )

Parameters

member_or_tuple
The member or tuple to rank.

set
The set containing the tuple to rank. Should not have duplicate members.

numeric_value_expr
Optional. Numeric sorting criteria.

ORDINALRANK
Optional. Rank duplicates separately.

DENSERANK
Optional. Rank with no gaps in ordinals.

PERCENTRANK
Optional. Rank on a scale from 0 to 1.

Notes

This function is applicable only to aggregate storage databases.

If no numeric value expression is given, this function returns the 1-based position of the tuple in the set.

If a numeric value expression is given, this function sorts the set based on the numeric value and returns the 1-based position of the tuple in the sorted set.

If an optional rank flag is given, this function sorts the set based on the numeric value and returns the 1-based position of the tuple in the sorted set according to the instructions in the flag. The meanings of the flags are:

- [no flag]: Default behavior. Ties are given the same rank, and the next member is the count of members. Example: (1,1,1,4,5)
- ORDINALRANK: Ties are decided by Essbase. Duplicates are considered different entities. Example: (1,2,3,4,5).
- DENSERANK: Ties are given the same rank, but there are no gaps in ordinals. Example: (1,1,1,2,3).
- PERCENTRANK: Rank values are scaled by the cumulative sum up to this member. Example: (.1, .15, .34, .78, 1.0). Values range from 0.0 to 1.0.

In the cases where this function sorts the set, it sorts tuples in descending order, and assigns ranks based on that order (highest value has a rank of 1).
Example

Example 1

WITH MEMBER [Measures].[Units_Rank] AS
  'Rank(Products.CurrentMember, Products.CurrentMember.Siblings)'
SELECT
  {Units, [Price Paid], [Units_Rank]}
ON COLUMNS,
  { Products.Members } ON ROWS
FROM ASOSamp.Sample;

Example 2

WITH MEMBER [Measures].[Units_Rank] AS
  'Rank(Products.CurrentMember, Products.CurrentMember.Siblings)'
SELECT {Units, [Measures].[Units_Rank]}
ON COLUMNS,
  Union(Children([Televisions]),
        Children([Radios]))
ON ROWS
FROM ASOSamp.Sample;

RealValue

Returns a value for the specified member or tuple without the inherited attribute dimension context.

Syntax

tuple[.RealValue]

member[.RealValue]

Parameters

tuple
A tuple for which to return a real value

member
A member for which to return a real value

Example

The following query sorts level-0 members of the Product dimension by the real value of Sales without the attribute dimension (Ounces_12) context, in descending order, and returns their sales for Ounces_12.

SELECT
  {[Sales]}

FROM ASOSamp.Sample;
ON COLUMNS,
Order([Product].Levels(0).Members,
    [Sales].REALVALUE, BDESC)
ON ROWS
FROM Sample.Basic
WHERE ([OUNCES_12]) ;

RelMemberRange

Returns a set that is based on the relative position of the specified member in the
database outline.

Note:

When multiple hierarchies are enabled, this function returns NULL when the
range begins in one hierarchy and terminates in another hierarchy.

Syntax

RelMemberRange ( member, prevcount, nextcount, [,layertype] [, hierarchy ])

Parameters

member
An input member in the set you want to return.

prevcount
The number of members in the same layer specified by layertype prior to member to
include in the return set.

nextcount
The number of members in the same layer specified by layertype following member to
include in the return set.

layertype
GENERATION or LEVEL. If omitted, the default is GENERATION. Defines whether
the set to be returned is based the same generation or on the same level as member.

hierarchy
Optional. A specific hierarchy within the time dimension.

Example

The following examples are based on ASOSamp.Sample.

Example 1

SELECT
RelMemberRange ([PORTLAND - OR],1,2)
ON COLUMNS
FROM asosamp.sample
This query returns the set:

```
{[PHOENIX - OR],[PORTLAND - OR],[POWERS - OR],[PRAIRIE CITY - OR]}
```

**Example 2**

RelMemberRange(Apr, 5, 0)

returns the set {Jan, Feb, Mar, Apr}. Note that the output set has only four members.

RelMemberRange(Apr, 5, 10)

returns the set {Jan, Feb, Mar, Apr, May ... Dec}. Note that the output set has only four previous members and seven next members of Apr.

**See Also**

LastPeriods

**Remainder**

Returns the fractional part of the numeric value expression.

**Syntax**

```
Remainder ( numeric_value_expression )
```

**Parameters**

**numeric_value_expression**

A numeric value expression (see MDX Grammar Rules).

**Example**

```
Remainder([Margin %])
```

extracts the fractional part of the [Margin %] value.

The following query shows [Margin %] and the fractional part of it for all members of the Product dimension.

```
WITH
  MEMBER [Measures].[Margin % Rem] AS 'Remainder([Margin %])'
SELECT
  {[Margin %], [Margin % Rem]} ON COLUMNS,
  [Product].Members ON ROWS
FROM Sample.Basic
```

This query returns the following grid:
Table 6-124  Output Grid from MDX Example

<table>
<thead>
<tr>
<th>(axis)</th>
<th>Margin %</th>
<th>Margin % Rem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product</td>
<td>55.262</td>
<td>0.262</td>
</tr>
<tr>
<td>100</td>
<td>57.273</td>
<td>0.273</td>
</tr>
<tr>
<td>100-10</td>
<td>61.483</td>
<td>0.483</td>
</tr>
<tr>
<td>100-20</td>
<td>51.479</td>
<td>0.479</td>
</tr>
<tr>
<td>100-30</td>
<td>50.424</td>
<td>0.424</td>
</tr>
<tr>
<td>200</td>
<td>55.540</td>
<td>0.540</td>
</tr>
<tr>
<td>200-10</td>
<td>54.270</td>
<td>0.270</td>
</tr>
<tr>
<td>200-20</td>
<td>56.436</td>
<td>0.436</td>
</tr>
<tr>
<td>200-30</td>
<td>56.450</td>
<td>0.450</td>
</tr>
<tr>
<td>200-40</td>
<td>55.753</td>
<td>0.753</td>
</tr>
<tr>
<td>300</td>
<td>54.238</td>
<td>0.238</td>
</tr>
<tr>
<td>300-10</td>
<td>55.816</td>
<td>0.816</td>
</tr>
<tr>
<td>300-20</td>
<td>42.992</td>
<td>0.992</td>
</tr>
<tr>
<td>300-30</td>
<td>57.551</td>
<td>0.551</td>
</tr>
<tr>
<td>400</td>
<td>53.600</td>
<td>0.600</td>
</tr>
<tr>
<td>400-10</td>
<td>57.354</td>
<td>0.354</td>
</tr>
<tr>
<td>400-20</td>
<td>56.299</td>
<td>0.299</td>
</tr>
<tr>
<td>400-30</td>
<td>39.477</td>
<td>0.477</td>
</tr>
<tr>
<td>Diet</td>
<td>55.397</td>
<td>0.397</td>
</tr>
<tr>
<td>100-20</td>
<td>51.479</td>
<td>0.479</td>
</tr>
<tr>
<td>200-20</td>
<td>56.436</td>
<td>0.436</td>
</tr>
<tr>
<td>300-30</td>
<td>57.551</td>
<td>0.551</td>
</tr>
</tbody>
</table>

Right

Returns a specified number (length) of characters from the right side of the string.

Syntax

Right ( string , length )

Parameters

string
Input string.

length
The number of characters to return from the right side of the input string.

Example

Right ("Northwind", 4)

returns wind.
Round

Rounds a numeric value expression to the specified number of digits.

Syntax

Round ( numeric_value_expression, index )

Parameters

numeric_value_expression
A numeric value expression (see MDX Grammar Rules).

index
Expression yielding an integer value. numeric_value_expression is rounded to the number of digits specified by this value. The fractional part of index is ignored.

Example

Round(234.5678, 2) returns 234.57.

RTrim

Trims all whitespace on the right side of the string.

Syntax

RTrim ( string )

Parameters

string
Input string.

Example

RTrim("STRING   ")

returns "STRING"

siblings

Returns the siblings of the input member, optionally based on selection options.

Syntax

siblings ( member[, selection [,include_or_exclude]] )

member.Siblings
Parameters

**member**
The member for which siblings are returned.

**selection**
Optional. This option can be one of the following:
- LEFT—Selects the siblings to the left of the input member
- RIGHT—Selects the siblings to the right of the input member
- ALL—Selects all the siblings of the input member
If no selection is made, the default is ALL.

**include_or_exclude**
Optional. This option can be one of the following:
- INCLUDEMEMBER—Includes the input member in the siblings list
- EXCLUDEMEMBER—Excludes the input member from the siblings list
If neither is specified, the default is to include the input member.

Notes
- If the input member is the top level of the dimension, this function returns a set containing the input member.
- In aggregate storage databases, in multiple-hierarchy-enabled dimensions, if the input member is a top-level member of a hierarchy, the output is members across hierarchies that are top-level members of hierarchies.
- This function is the same as `Children(member.parent)`.
- The `member. Siblings` syntax returns the same set as `Siblings(member), Siblings(member, ALL), or Siblings(member, ALL, INCLUDEMEMBER)`.

Example

**Example 1**

`Siblings(Year)` returns `{Year}`.

The following query

```mdx
SELECT
CrossJoin ( 
    Union ( 
        Siblings ([Old Fashioned]),
        {{[Root Beer]}, {[Cream Soda]}}
    ),
    {{[Budget]}, {[Variance]}}
) 
ON COLUMNS
from Sample.Basic
```

returns the grid:
Table 6-125  Output Grid from MDX Example

<table>
<thead>
<tr>
<th></th>
<th>Old Fashioned</th>
<th>Diet Root Beer</th>
<th>Sarsaparilla</th>
<th>Birch Beer</th>
<th>Root Beer</th>
<th>Cream Soda</th>
</tr>
</thead>
<tbody>
<tr>
<td>Budget Variance</td>
<td>11640</td>
<td>-4439</td>
<td>14730</td>
<td>5050</td>
<td>4530</td>
<td>35950</td>
</tr>
<tr>
<td></td>
<td>Budget Variance</td>
<td></td>
<td></td>
<td>Budget Variance</td>
<td></td>
<td>Budget Variance</td>
</tr>
<tr>
<td></td>
<td>-439</td>
<td>-2705</td>
<td>-414</td>
<td>-438</td>
<td>-7996</td>
<td>-3561</td>
</tr>
</tbody>
</table>

Example 2

The following examples are based on a Years – Quarters – Months Time hierarchy.

\[ \text{Siblings([Feb 2000], LEFT, INCLUDEMEMBER)} \]

\[ \text{Returns ([Jan 2000], [Feb 2000])}. \]

\[ \text{Siblings([Feb 2000], RIGHT, EXCLUDEMEMBER)} \]

\[ \text{Returns ([Mar 2000])}. \]

\[ \text{Siblings([Mar 2000], LEFT)} \]

\[ \text{Returns ([Jan 2000], [Feb 2000], [Mar 2000])}. \]

\[ \text{Siblings([May 2000], RIGHT)} \]

\[ \text{Returns ([May 2000], [Jun 2000])}. \]

\[ \text{Siblings([Mar 2000])} \]

\[ \text{OR} \]

\[ \text{[Mar 2000].Siblings} \]

\[ \text{Returns ([Jan 2000], [Feb 2000], [Mar 2000])}. \]

Stddev

Calculates the standard deviation of the specified set. The calculation is based upon a sample of a population. Standard deviation is a measure of how widely values are dispersed from their mean (average).
Syntax

Stddev ( set [,numeric_value_expression [,IncludeEmpty] ])

Parameters

set
A valid MDX set specification.

numeric_value_expression
A numeric value or an expression that returns a numeric value (see MDX Grammar Rules).

IncludeEmpty
Use this keyword if you want to include in the calculation any tuples with #MISSING values. Otherwise, they are omitted by default.

Example

The following example, based on Sample Basic, calculates the standard deviation (based on a sample of a population) of the January sales values for all products sold in New York.

```
WITH MEMBER [Measures].[Std Deviation]
AS
  'Stddev(
    Crossjoin(
      {[Product].Children}, {[Measures].[Sales]}
    )
  ),
SELECT
  {[Scenario].[Actual],[Scenario].[Budget]}
ON COLUMNS,
  {Crossjoin(
    {[Measures].[Sales]},{[Product].Children}
  ),
  Crossjoin(
    {[Measures].[Sales], [Measures].[Std Deviation]},
    {[Product]}
  )}
ON ROWS
FROM
  Sample.Basic
WHERE
  ([Year].[Jan], [Market].[New York])
```

This query returns the following grid:
Table 6-126   Output Grid from MDX Example

<table>
<thead>
<tr>
<th>(axis)</th>
<th>Actual</th>
<th>Budget</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Sales, 100)</td>
<td>678</td>
<td>640</td>
</tr>
<tr>
<td>(Sales, 200)</td>
<td>551</td>
<td>530</td>
</tr>
<tr>
<td>(Sales, 300)</td>
<td>663</td>
<td>510</td>
</tr>
<tr>
<td>(Sales, 400)</td>
<td>587</td>
<td>620</td>
</tr>
<tr>
<td>(Sales, Diet)</td>
<td>#Missing</td>
<td>#Missing</td>
</tr>
<tr>
<td>(Sales, Product)</td>
<td>2479</td>
<td>2300</td>
</tr>
<tr>
<td>(Std Deviation, Product)</td>
<td>60.723</td>
<td>64.55</td>
</tr>
</tbody>
</table>

See Also
Stddevp

Stddevp

Calculates the standard deviation of the specified set. This function assumes that the set represents the entire population. If you want to calculate based a sample of a population, use Stddev.

Standard deviation is a measure of how widely values are dispersed from their mean (average).

Syntax

\[
\text{Stddevp} \left( \text{set} [, \text{numeric\_value\_expression} [,\text{IncludeEmpty}]] \right)
\]

Parameters

- **set**
  A valid MDX set specification.

- **numeric\_value\_expression**
  A numeric value or an expression that returns a numeric value (see MDX Grammar Rules).

**IncludeEmpty**
Use this keyword if you want to include in the calculation any tuples with #MISSING values. Otherwise, they are omitted by default.

Example

The following example, based on Sample Basic, calculates the standard deviation (based on the entire population) of the January sales values for all products sold in New York.

```
WITH MEMBER [Measures].[Std Deviation] AS
'StddevP{
    Crossjoin{
        ([Product].Children), ([Measures].[Sales])
    }
}
```
SELECT
    {[Scenario].[Actual],[Scenario].[Budget]}
ON COLUMNS,
    Crossjoin(
        {[Measures].[Sales]},{[Product].Children}
    ),
    Crossjoin(
        {[Measures].[Sales], [Measures].[Std Deviation]},{[Product]}
    )
ON ROWS
FROM
    Sample.Basic
WHERE
    ([Year].[Jan], [Market].[New York])

This query returns the following grid:

**Table 6-127  Output Grid from MDX Example**

<table>
<thead>
<tr>
<th>(axis)</th>
<th>Actual</th>
<th>Budget</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Sales, 100)</td>
<td>678</td>
<td>640</td>
</tr>
<tr>
<td>(Sales, 200)</td>
<td>551</td>
<td>530</td>
</tr>
<tr>
<td>(Sales, 300)</td>
<td>663</td>
<td>510</td>
</tr>
<tr>
<td>(Sales, 400)</td>
<td>587</td>
<td>620</td>
</tr>
<tr>
<td>(Sales, Diet)</td>
<td>#Missing</td>
<td>#Missing</td>
</tr>
<tr>
<td>(Sales, Product)</td>
<td>2479</td>
<td>2300</td>
</tr>
<tr>
<td>(Std Deviation, Product)</td>
<td>52.59</td>
<td>55.9</td>
</tr>
</tbody>
</table>

See Also

**Stddev**

**StrToMbr**

Converts a string to a member name.

**Syntax**

StrToMbr ( string [, dimension ] [, MEMBER_NAMEONLY | alias_table_name ] )

**Parameters**

**string**
Input string.
dimension

Optional dimension specification. If used, only member names found in this dimension will be returned.

MEMBER_NAMEONLY

Optional. Create member name only out of member names found (not including aliases). The default is to search for member names and all aliases.

alias_table_name

Optional. Create member name only out of alias name strings found. The default is to search for member names and all aliases.

Notes

You can also use member properties as string input. These properties include MEMBER_NAME, MEMBER_UNIQUE_NAME, MEMBER_ALIAS, ANCESTOR_NAMES, and COMMENTS. For example:

SELECT {StrToMbr(Sales.MEMBER_NAME)} ON COLUMNS
FROM Sample.Basic

Example

SELECT
{ StrToMbr("CA", [Geography], "Default") } 
ON COLUMNS,
Children([High End Merchandise])
ON ROWS
FROM Asosamp.Sample

returns CA.

SELECT
{ StrToMbr("Quarter1", [Year], MEMBER_NAMEONLY) }
DIMENSION PROPERTIES [YEAR].[MEMBER_ALIAS]
ON COLUMNS,
Children([100])
ON ROWS
FROM Sample.Basic

returns nothing, because "Quarter1" is an alias.

SELECT
{ StrToMbr("Qtr1", [Year], MEMBER_NAMEONLY) }
DIMENSION PROPERTIES [YEAR].[MEMBER_ALIAS]
ON COLUMNS,
Children([100])
ON ROWS
FROM Sample.Basic
returns Qtr1.

SELECT
  { StrToMbr("Quarter1", [Year], "Long Names") }
DIMENSION PROPERTIES [YEAR].[MEMBER_ALIAS]
  ON COLUMNS,
  Children([100])
  ON ROWS
FROM Sample.Basic

returns Qtr1 because "Quarter1" is in the "Long Names" alias table.

StrToNum

Converts a string to a number.

Syntax

StrToNum (string)

Parameters

string
Input string.

Notes

This function returns a numeric value after converting the string to a number. For example, string "0.9" becomes the number 0.9. StrToMbr returns zero if the string cannot be converted.

Example

StrToNum("0.9")

returns 0.9 as a numeric value expression.

Subset

Returns a subset from a set, in which the subset is a numerically specified range of tuples.

Syntax

Subset { set, index1 [,index2 ] }

Parameters

set
The set from which to take tuples.
index1
The location of the tuple with which to begin the subset. Example: if index1 is 0, the subset begins with the first tuple of set. If a negative value, the return is an empty set.

index2
Optional. The count of tuples to include in the subset. If omitted, all tuples to the end of set are returned. If a negative value, the return is an empty set. If the count goes beyond the range of the input set, all tuples to the end of the set are returned.

Notes
The first tuple of the subset is represented by index1. If index1 is 0, then the first tuple of the returned subset will be the same as the first tuple of the input set.

Example
Example 1
The following expression

\[
\text{Subset}\ {\{\text{Product.Members}\}, 0}
\]

returns the set:

\[
\{ \text{Product, [100-10], [100-20], [100-30], [100], [200-10], [200-20], [200-30], [200-40], [200], [300-10], [300-20], [300-30], [300], [400-10], [400-20], [400-30], [400], [100-20], [200-20], [300-30], \text{Diet} } \}
\]

All tuples of the set \{Product.Members\} are returned, because the subset is told to begin with the first tuple, and no count of tuples given for index2.

Example 2
The following expression

\[
\text{Subset}\ {\{\text{Product.Members}\}, 0, 4}
\]

returns the set:

\[
\{ \text{Product, [100], [100-10], [100-20] } \}
\]

Therefore, the following query

Select
\[
\text{Subset}\ {\{\text{Product.Members}\}, 0, 4}
\]
on columns
from sample.basic

returns the grid:
### Table 6-128  Output Grid from MDX Example

<table>
<thead>
<tr>
<th>Product</th>
<th>100</th>
<th>100-10</th>
<th>100-20</th>
</tr>
</thead>
<tbody>
<tr>
<td>105522</td>
<td>30468</td>
<td>22777</td>
<td>5708</td>
</tr>
</tbody>
</table>

**Substring**

Returns the substring between a starting and ending position. Both the positional arguments are 1-based.

**Syntax**

```plaintext
Substring ( string, index1 [, index2 ] )
```

**Parameters**

- **string**
  String to subdivide (or field containing that string).
- **index1**
  A number \( n \) representing a starting position within a string.
- **index2**
  Optional. A number \( n \) representing an ending position within a string. If omitted, the endpoint is assumed to be the end of the original string.

**Sum**

Returns the sum of values of tuples in a set.

**Syntax**

```plaintext
Sum ( set [, numeric_value_expression ] )
```

**Parameters**

- **set**
  The set containing the tuples to aggregate. If empty, the return value is `#MISSING`.
- **numeric_value_expression**
  Optional. An expression that returns a value. Commonly used to restrict the aggregation to a slice from a Measures dimension (see MDX Grammar Rules). In the example below, `[Measures].[Total Expenses]` is the numeric value expression provided to the Sum function.

**Notes**

For optimized performance of this function on aggregate storage databases, include in your query the following kinds of sets:

- Any of the following functions, used within the named set and/or as an argument to this function: Intersect, CurrentMember, Distinct, CrossJoin, PeriodsToDate.
• The Filter function, with the search condition defined as:
  \texttt{dimensionName.CurrentMember IS memberName}.

• The IIF function, with the \texttt{true_part} and \texttt{false_part} being sets that meet the above criteria.

• The use of any other functions (such as Members) disables the optimization.

• The second parameter, \texttt{numeric_value_expression}, must be included for optimal performance.

Optimal query performance may require a larger formula cache size. If you get an error message similar to the following, adjust the \texttt{MAXFORMULACACHESIZE} configuration setting accordingly:

\texttt{Not enough memory for formula execution. Set MAXFORMULACACHESIZE configuration parameter to [1072]KB and try again.}

For each tuple in set, the numeric value expression is evaluated in the context of that tuple and the resulting values are summed up.

The return value of \texttt{Sum} is \#MISSING if either of the following is true:

• The input set is empty.

• All tuple evaluations result in \#MISSING values.

Example

\begin{verbatim}
WITH MEMBER [Market].[Sum Expense for Main States] AS 'Sum
  \{[Market].[California], [Market].[Colorado],
  [Market].[Texas], [Market].[Illinois],
  [Market].[Ohio], [Market].[New York],
  [Market].[Massachusetts], [Market].[Florida],
  [Measures].[Total Expenses]
  
  SELECT
  \{[Measures].[Total Expenses]\}
ON COLUMNS,
  \{UDA([Market], "Major Market"),
  [Market].[Sum Expense for Main States]\}
ON ROWS
FROM
  Sample.Basic
WHERE ([Scenario].[Actual])
\end{verbatim}

returns the grid:

\textbf{Table 6-129 Output Grid from MDX Example}

\begin{center}
\begin{tabular}{l|c}
\hline
\textbf{(axis)} & \textbf{Total Expenses} \\
\hline
New York & 8914 \\
Massachusetts & 3412 \\
\hline
\end{tabular}
\end{center}
Table 6-129  (Cont.) Output Grid from MDX Example

<table>
<thead>
<tr>
<th>(axis)</th>
<th>Total Expenses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Florida</td>
<td>5564</td>
</tr>
<tr>
<td>East</td>
<td>25310</td>
</tr>
<tr>
<td>California</td>
<td>11737</td>
</tr>
<tr>
<td>Texas</td>
<td>4041</td>
</tr>
<tr>
<td>Illinois</td>
<td>6900</td>
</tr>
<tr>
<td>Ohio</td>
<td>5175</td>
</tr>
<tr>
<td>Colorado</td>
<td>6131</td>
</tr>
<tr>
<td>Central</td>
<td>34864</td>
</tr>
<tr>
<td>Sum Expense for Main States</td>
<td>51874</td>
</tr>
</tbody>
</table>

See Also

Aggregate

Tail

Returns the last $n$ members or tuples present in a set.

Syntax

Tail ( set [, index ] )

Parameters

set
The set from which to take items.

index
The number of items to take from the end of the set. If omitted, the default is 1. If less than 1, an empty set is returned. If the value exceeds the number of tuples in the input set, the original set is returned.

Example

Example 1

This example uses the following part of the Sample Basic outline:
The following expression

\[ \text{[Product].children} \]

returns the set:

\[ \{ [100], [200], [300], [400], [\text{[Diet]}] \} \]

Therefore, the following expression

\[ \text{Tail ( [Product].children, 2) } \]

returns the last two members of the previous result set:

\[ \{ [400], [\text{[Diet]}] \} \]

**Example 2**

This example uses the following parts of the Sample Basic outline:

-100 (+)
  -100-10
  -100-20
  -100-30

-South (+)
  -Texas
  -Oklahoma
  -Louisiana
  -New Mexico

-Year
  -Qtr1
  -Qtr2
  -Qtr3
  -Qtr4

The following expression

\[ \text{Crossjoin ( [100].children, [South].children )} \]

returns the set:

\[ \{ ([100-10], \text{Texas}), ([100-10], \text{Oklahoma}), ([100-10], \text{Louisiana}), ([100-10], \text{New Mexico}), ([100-20], \text{Texas}), ([100-20], \text{Oklahoma}), ([100-20], \text{Louisiana}), ([100-20], \text{New Mexico}) \} \]
(100–30, Texas), (100–30, Oklahoma), (100–30, Louisiana),
(100–30, [New Mexico])

And the following expression:

Tail ( Crossjoin ([100].children, [South].children), 8 )

returns the last 8 tuples of the previous result set:

{ (100–20, Texas), (100–20, Oklahoma), (100–20, Louisiana),
(100–20, [New Mexico]),
(100–30, Texas), (100–30, Oklahoma), (100–30, Louisiana),
(100–30, [New Mexico])

Additionally, the following expression

([Year].generations(2).members)

returns the set of members comprising the second generation of the Year dimension:

{ [Qtr1], [Qtr2], [Qtr3], [Qtr4] }

Therefore, the following query

```
SELECT
{([Year].generations(2).members)}
ON COLUMNS,
Tail ( Crossjoin ([100].children, [South].children), 8 )
ON ROWS
FROM Sample.Basic
```

returns the grid:

<table>
<thead>
<tr>
<th>(axis)</th>
<th>(axis)</th>
<th>Qtr1</th>
<th>Qtr2</th>
<th>Qtr3</th>
<th>Qtr4</th>
</tr>
</thead>
<tbody>
<tr>
<td>100–20</td>
<td>Texas</td>
<td>206</td>
<td>199</td>
<td>152</td>
<td>82</td>
</tr>
<tr>
<td></td>
<td>Oklahoma</td>
<td>84</td>
<td>66</td>
<td>55</td>
<td>79</td>
</tr>
<tr>
<td></td>
<td>Louisiana</td>
<td>119</td>
<td>158</td>
<td>171</td>
<td>104</td>
</tr>
<tr>
<td></td>
<td>New Mexico</td>
<td>−103</td>
<td>−60</td>
<td>−97</td>
<td>−18</td>
</tr>
<tr>
<td>100–30</td>
<td>Texas</td>
<td>#Missing</td>
<td>#Missing</td>
<td>#Missing</td>
<td>#Missing</td>
</tr>
<tr>
<td></td>
<td>Oklahoma</td>
<td>#Missing</td>
<td>#Missing</td>
<td>#Missing</td>
<td>#Missing</td>
</tr>
<tr>
<td></td>
<td>Louisiana</td>
<td>#Missing</td>
<td>#Missing</td>
<td>#Missing</td>
<td>#Missing</td>
</tr>
<tr>
<td></td>
<td>New Mexico</td>
<td>#Missing</td>
<td>#Missing</td>
<td>#Missing</td>
<td>#Missing</td>
</tr>
</tbody>
</table>
To suppress the missing rows, use NON EMPTY at the beginning of the row axis specification:

```
SELECT
    {([Year].generations(2).members)}
ON COLUMNS,
    NON EMPTY
    Tail (
        Crossjoin ([100].children, [South].children),
        8)
ON ROWS
FROM Sample.Basic
```

This modified query returns as many of the 8 requested tuples as it can, without returning any that have entirely #Missing data:

**Table 6-131  Output Grid from MDX Example**

<table>
<thead>
<tr>
<th>(axis)</th>
<th>Qtr1</th>
<th>Qtr2</th>
<th>Qtr3</th>
<th>Qtr4</th>
</tr>
</thead>
<tbody>
<tr>
<td>100-20</td>
<td>206</td>
<td>199</td>
<td>152</td>
<td>82</td>
</tr>
<tr>
<td>100-20</td>
<td>84</td>
<td>66</td>
<td>55</td>
<td>79</td>
</tr>
<tr>
<td>100-20</td>
<td>119</td>
<td>158</td>
<td>171</td>
<td>104</td>
</tr>
<tr>
<td>100-20</td>
<td>-103</td>
<td>-60</td>
<td>-97</td>
<td>-18</td>
</tr>
</tbody>
</table>

**See Also**

Head

**Todate**

Converts date strings to numbers that can be used in calculations.

**Syntax**

```
Todate ( string_value_expression_1 , string_value_expression_2 )
```

**Parameters**

- **string_value_expression_1**
  The format of the date string, either "mm-dd-yyyy" or "dd-mm-yyyy" (must be in lower case).

- **string_value_expression_2**
  The date string.

**Notes**

- If you specify a date that is earlier than 01-01-1970, this function returns an error.
- The latest date supported by this function is 12-31-2037.
Example

For products introduced before 06.01.1996, the following query calculates a Revised Budget that is 110% of Budget.

```mdx
WITH MEMBER 
    [Scenario].[Revised Budget] 
AS 
    'IIF ( 
        [Product].CurrentMember.[Intro Date] 
        > TODATE("mm-dd-yyyy","06-01-1996"), 
        Budget * 1.1, Budget 
    )' 
SELECT 
    {[Scenario].Budget, [Scenario].[Revised Budget]} 
ON COLUMNS, 
    [Product].[200].Children 
DIMENSION PROPERTIES [Intro Date] 
ON ROWS 
FROM Sample.Basic 
WHERE ([Measures].[Sales], [Year].[Qtr3])
```

This query returns the grid:

### Table 6-132  Output Grid from MDX Example

<table>
<thead>
<tr>
<th>Axis-1</th>
<th>Axis-1.properties</th>
<th>Budget</th>
<th>Revised Budget</th>
</tr>
</thead>
<tbody>
<tr>
<td>200-10</td>
<td>(Intro Date = 09-27-1995, type: TIME,)</td>
<td>11060</td>
<td>11060</td>
</tr>
<tr>
<td>200-20</td>
<td>(Intro Date = 07-26-1996, type: TIME,)</td>
<td>9680</td>
<td>10648</td>
</tr>
<tr>
<td>200-30</td>
<td>(Intro Date = 12-10-1996, type: TIME,)</td>
<td>3880</td>
<td>4268</td>
</tr>
<tr>
<td>200-40</td>
<td>(Intro Date = 12-10-1996, type: TIME,)</td>
<td>2660</td>
<td>2926</td>
</tr>
</tbody>
</table>

**TodateEx**

Returns the numeric date value from input date-string according to the date-format specified. The date returned is the number of seconds elapsed since midnight, January 1, 1970.

If the date or the date format strings are invalid, an error is returned.

**Syntax**

```
TodateEx ( internal-date-format, date-string )
```
Parameters

**internal-date-format**
One of the following literal strings (excluding ordered-list numbers and parenthetical examples) indicating a supported date format.

1. "mon dd yyyy" *(Example: mon = Aug)*
2. "Month dd yyyy" *(Example: Month = August)*
3. "mm/dd/yy"
4. "mm/dd/yyyy"
5. "yy.mm.dd"
6. "dd/mm/yy"
7. "dd.mm.yy"
8. "dd-mm-yy"
9. "dd Month yy"
10. "dd mon yy"
11. "Month dd, yy"
12. "mon dd, yy"
13. "mm-dd-yyyy"
14. "yy/mm/dd"
15. "ymmd"
16. "dd Month yyyy"
17. "dd mon yyyy"
18. "yyyy-mm-dd"
19. "yyyy/mm/dd"
20. Long format *(Example: WeekDay, Mon dd, yyyy)*
21. Short format *(Example: m/d/yy)*

**date-string**
A date string following the rules of *internal-date-format*. The following examples correspond to the above listed internal date formats.

1. Jan 15 2006
2. January 15 2006
3. 01/15/06
4. 01/15/2006
5. 06.01.06
6. 15/01/06
7. 15.01.06
Notes

- This function is an extension of `Todate`.
- This function is case-sensitive. For example, using `apr` instead of `Apr` returns an error.
- Using extra whitespace not included in the internal format strings returns an error.
- Trailing characters after the date format has been satisfied are ignored. If you erroneously use a date string of 06/20/2006 with date format `mm/dd/yy`, the trailing 06 is ignored and the date is interpreted as June 20, 2020.
- Long Format (Weekday, Mon dd, yyyy) is not verified for a day-of-week match to the given date.
  
  For example: For date string `Sunday, March 13, 2007` with date format Long Format, the input date string is parsed correctly for March 13, 2007, although March 13, 2007 does not fall on Sunday.
- If you specify a date that is earlier than 01-01-1970, this function returns an error.
- The latest date supported by this function is 12-31-2037.
- When the `yy` format is used, this function interprets years in the range 1970 to 2029.

Example

The following query returns the actual sales on May 31, 2005 for the product Cola in the market California.

`TodateEx()` returns the date May 31, 2005, corresponding to date string `05.31.2005`. `StrToMbr` returns the corresponding day level member, capturing May 31, 2005.

```sql
SELECT
  ([Sales])
ON COLUMNS,
```
Today

Returns a number representing the current date on the Essbase computer. The number is the number of seconds elapsed since midnight, January 1, 1970.

Syntax

Today

Notes

The date returned can be used as input to other functions listed in the See Also section.

Example

This query returns today's actual sales for the product Cola in the market California. Today() returns today's date. StrToMbr retrieves the day member represented by the date returned by Today.

```
SELECT
    {[Sales]}
ON COLUMNS,
    {
        StrToMbr( 
            FormatDate( Today(), "Mon dd yyyy")
        )
    }
ON ROWS
FROM Mysamp.basic;
```

See Also

DateToMember
DateRoll
DatePart
FormatDate
TopCount

Returns a set of \( n \) elements ordered from largest to smallest, optionally based on an evaluation.

This function ignores missing values.

Syntax

\[
\text{TopCount} \left( \text{set}, \text{index} \left[, \text{numeric\_value\_expression} \right] \right)
\]

Parameters

\textbf{set}

The set from which the top \( n \) elements are selected.

\textbf{index}

The number of elements to include in the set (\( n \)).

\textbf{numeric\_value\_expression}

Optional. An expression further defining the selection criteria (see MDX Grammar Rules).

Example

The following query selects the five top-selling markets in terms of yearly Diet products sales, and displays the quarterly sales for each Diet product.

\[
\begin{align*}
\text{SELECT} & \\
& \text{CrossJoin}(
\quad [\text{Product}].[\text{Diet}].\text{Children},
\quad [\text{Year}].\text{Children})
\quad \text{ON COLUMNS},
\quad \text{TopCount}(
\quad [\text{Market}].\text{Levels}(0).\text{Members},
\quad 5,
\quad [\text{Product}].[\text{Diet}])
\quad \text{ON ROWS}
\quad \text{FROM Sample.Basic}
\quad \text{WHERE ([Scenario].[Actual], [Measures].[Sales])}
\end{align*}
\]

This query returns the grid:

Table 6-133  Output Grid from MDX Example

<table>
<thead>
<tr>
<th>(axis)</th>
<th>100-20</th>
<th>100-20</th>
<th>100-20</th>
<th>100-20</th>
<th>200-20</th>
<th>200-20</th>
<th>200-20</th>
<th>200-20</th>
<th>300-30</th>
<th>300-30</th>
<th>300-30</th>
<th>300-30</th>
</tr>
</thead>
<tbody>
<tr>
<td>Illinois</td>
<td>755</td>
<td>958</td>
<td>1050</td>
<td>888</td>
<td>1391</td>
<td>1520</td>
<td>1562</td>
<td>1402</td>
<td>1391</td>
<td>1520</td>
<td>1562</td>
<td>1402</td>
</tr>
<tr>
<td>California</td>
<td>367</td>
<td>491</td>
<td>506</td>
<td>468</td>
<td>1658</td>
<td>1833</td>
<td>1954</td>
<td>1706</td>
<td>700</td>
<td>802</td>
<td>880</td>
<td>673</td>
</tr>
</tbody>
</table>
Table 6-133  (Cont.) Output Grid from MDX Example

<table>
<thead>
<tr>
<th>(axis)</th>
<th>100-20</th>
<th>100-20</th>
<th>100-20</th>
<th>100-20</th>
<th>200-20</th>
<th>200-20</th>
<th>200-20</th>
<th>200-20</th>
<th>300-30</th>
<th>300-30</th>
<th>300-30</th>
<th>300-30</th>
</tr>
</thead>
<tbody>
<tr>
<td>Colorado</td>
<td>700</td>
<td>802</td>
<td>880</td>
<td>673</td>
<td>549</td>
<td>465</td>
<td>412</td>
<td>539</td>
<td>1006</td>
<td>921</td>
<td>892</td>
<td>991</td>
</tr>
<tr>
<td>Washing ton</td>
<td>637</td>
<td>712</td>
<td>837</td>
<td>704</td>
<td>459</td>
<td>498</td>
<td>597</td>
<td>514</td>
<td>944</td>
<td>799</td>
<td>708</td>
<td>927</td>
</tr>
<tr>
<td>Iowa</td>
<td>162</td>
<td>153</td>
<td>121</td>
<td>70</td>
<td>129</td>
<td>129</td>
<td>129</td>
<td>129</td>
<td>1658</td>
<td>1833</td>
<td>1954</td>
<td>1706</td>
</tr>
</tbody>
</table>

See Also

BottomCount

TopPercent

Returns the smallest possible subset of a set for which the total results of a numeric evaluation are at least a given percentage. Elements in the result set are listed from largest to smallest.

Syntax

TopPercent ( set, percentage, numeric_value_expression )

Parameters

set
The set from which the top-percentile elements are selected.

percentage
The percentile. This argument must be a value between 0 and 100.

numeric_value_expression
The expression that defines the selection criteria (see MDX Grammar Rules).

Notes
This function ignores negative and missing values.

Example
The following query selects the top-selling markets that contribute 25% of the total yearly Diet products sales, and displays the quarterly sales for each Diet product.

SELECT
CrossJoin({
    [Product].[Diet].Children,
    [Year].Children
})
ON COLUMNS,
TopPercent(
    [Market].Levels(0).Members,
    25,
    [Product].[Diet]
This query returns the grid:

Table 6-134    Output Grid from MDX Example

<table>
<thead>
<tr>
<th>(axis)</th>
<th>100-20</th>
<th>100-20</th>
<th>100-20</th>
<th>100-20</th>
<th>200-20</th>
<th>200-20</th>
<th>300-30</th>
<th>300-30</th>
<th>300-30</th>
</tr>
</thead>
<tbody>
<tr>
<td>Illinois</td>
<td>755</td>
<td>958</td>
<td>1050</td>
<td>888</td>
<td>1391</td>
<td>1520</td>
<td>1562</td>
<td>1402</td>
<td>675</td>
</tr>
<tr>
<td>Califor</td>
<td>367</td>
<td>491</td>
<td>506</td>
<td>468</td>
<td>1658</td>
<td>1833</td>
<td>1954</td>
<td>1706</td>
<td>700</td>
</tr>
<tr>
<td>Colorado</td>
<td>700</td>
<td>802</td>
<td>880</td>
<td>673</td>
<td>549</td>
<td>465</td>
<td>412</td>
<td>539</td>
<td>1006</td>
</tr>
<tr>
<td></td>
<td>821</td>
<td>892</td>
<td>991</td>
<td>673</td>
<td>1006</td>
<td>921</td>
<td>892</td>
<td>991</td>
<td></td>
</tr>
</tbody>
</table>

TopSum

Returns the smallest possible subset of a set for which the total results of a numeric evaluation are at least a given sum. Elements of the result set are listed from largest to smallest.

Syntax

TopSum ( set, numeric_value_expression1, numeric_value_expression2 )

Parameters

set
The set from which the highest-summing elements are selected.

numeric_value_expression1
The given sum (see MDX Grammar Rules).

numeric_value_expression2
The numeric evaluation (see MDX Grammar Rules).

Notes

• If the total results of the numeric evaluation do not add up to the given sum, an empty set is returned.
• This function ignores negative and missing values.
Example

The following query selects the top-selling markets that collectively contribute 60,000 to the total yearly Diet products sales, and displays the quarterly sales for each Diet product.

```
SELECT
   CrossJoin(
      [Product].[Diet].Children,
      [Year].Children
   )
ON COLUMNS,
   TopSum(
      [Market].Levels(0).Members, 60000,
      [Product].[Diet]
   )
ON ROWS
FROM Sample.Basic
WHERE ([Scenario].[Actual], [Measures].[Sales])
```

This query returns the grid:

<table>
<thead>
<tr>
<th>(axis)</th>
<th>Qtr1</th>
<th>Qtr2</th>
<th>Qtr3</th>
<th>Qtr4</th>
<th>Qtr1</th>
<th>Qtr2</th>
<th>Qtr3</th>
<th>Qtr4</th>
<th>Qtr1</th>
<th>Qtr2</th>
<th>Qtr3</th>
<th>Qtr4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Illinois</td>
<td>755</td>
<td>958</td>
<td>1050</td>
<td>888</td>
<td>1391</td>
<td>1520</td>
<td>1562</td>
<td>1402</td>
<td>675</td>
<td>755</td>
<td>859</td>
<td>894</td>
</tr>
<tr>
<td>Califonia</td>
<td>367</td>
<td>491</td>
<td>506</td>
<td>468</td>
<td>1658</td>
<td>1833</td>
<td>1954</td>
<td>1706</td>
<td>700</td>
<td>802</td>
<td>880</td>
<td>673</td>
</tr>
<tr>
<td>Colorado</td>
<td>700</td>
<td>802</td>
<td>880</td>
<td>673</td>
<td>549</td>
<td>465</td>
<td>412</td>
<td>539</td>
<td>1006</td>
<td>921</td>
<td>892</td>
<td>991</td>
</tr>
<tr>
<td>Washington</td>
<td>637</td>
<td>712</td>
<td>837</td>
<td>704</td>
<td>459</td>
<td>498</td>
<td>597</td>
<td>514</td>
<td>944</td>
<td>799</td>
<td>708</td>
<td>927</td>
</tr>
<tr>
<td>Iowa</td>
<td>162</td>
<td>153</td>
<td>121</td>
<td>70</td>
<td>129</td>
<td>129</td>
<td>129</td>
<td>129</td>
<td>1658</td>
<td>1833</td>
<td>1954</td>
<td>1706</td>
</tr>
<tr>
<td>Florida</td>
<td>620</td>
<td>822</td>
<td>843</td>
<td>783</td>
<td>548</td>
<td>611</td>
<td>657</td>
<td>577</td>
<td>332</td>
<td>323</td>
<td>260</td>
<td>159</td>
</tr>
<tr>
<td>Oregon</td>
<td>389</td>
<td>303</td>
<td>277</td>
<td>322</td>
<td>1006</td>
<td>921</td>
<td>892</td>
<td>991</td>
<td>263</td>
<td>231</td>
<td>197</td>
<td>184</td>
</tr>
</tbody>
</table>

Truncate

Returns the integral part of a number. The return value has the same sign as its argument.

Syntax

```
Truncate ( numeric_value_expression )
```
Parameters

**numeric_value_expression**
Numeric value expression (see MDX Grammar Rules).

Example

`Truncate(2.65)` returns 2.

`Truncate(-8.12)` returns -8.

**TupleRange**

Returns the range of tuples between (and inclusive of) two tuples at the same level. The range is created by identifying the level of the arguments and pruning the result set to include only the argument tuples and the tuples that are, in terms of outline order, between them.

**Syntax**

`TupleRange (tuple1, tuple2)`

**Parameters**

**tuple1**
The first input tuple, marking the beginning of the range.

**tuple2**
The second input tuple, marking the end of the range.

**Notes**

- TupleRange serves the same purpose as the @XRANGE function in the Essbase calculator language.
- The two input tuples must be of the same dimensionality. See the example, wherein both input tuples are of the format ([Year],[Month]).

**Example**

TupleRange can be useful if you have two Time dimensions. For example, the following expression averages a value for the range of months from Mar 2005 to Feb 2006, inclusive.

```
AVG ( 
    TUPLERANGE ( 
        ([2005], [Mar]), ([2006], [Feb]) 
    ) 
)
```

The values are averaged for the following range:

```
{([2005], [Mar]),
 ([2005], [Apr]),
```
Uda

Selects all members to which a specified user-defined attribute is associated in the entire dimension or in a subtree rooted at the input member.

Syntax

Uda ( dimension | member, string_value_expression )

Parameters

dimension
The dimension in which matching UDAs are searched.

member
A member to search (descendants included) for matching UDAs.

string_value_expression
The name of the UDA to be selected. Can be an expression that evaluates to the UDA string, or an exact character string (not case-sensitive) enclosed in double quotation marks.

Notes

A user-defined attribute is a term associated with members of an outline to describe a characteristic. This function selects all members that have the specified UDA.

Example

Dimension Example

In the following query, the Uda function searches a dimension (top member included) for descendant members having a UDA of Major Market:

SELECT
  {{[Measures].[Sales], [Measures].[Profit]} ON COLUMNS,
   {UDA([Market], "Major Market")} ON ROWS
FROM Sample.Basic
WHERE {{[Year].[Jul], [Product].[Cola]}}
Table 6-136  Output Grid from MDX Example

<table>
<thead>
<tr>
<th>(axis)</th>
<th>Sales</th>
<th>Profit</th>
</tr>
</thead>
<tbody>
<tr>
<td>East</td>
<td>2248</td>
<td>1156</td>
</tr>
<tr>
<td>New York</td>
<td>912</td>
<td>370</td>
</tr>
<tr>
<td>Massachusetts</td>
<td>665</td>
<td>564</td>
</tr>
<tr>
<td>Florida</td>
<td>286</td>
<td>104</td>
</tr>
<tr>
<td>California</td>
<td>912</td>
<td>370</td>
</tr>
<tr>
<td>Texas</td>
<td>567</td>
<td>206</td>
</tr>
<tr>
<td>Central</td>
<td>1392</td>
<td>369</td>
</tr>
<tr>
<td>Illinois</td>
<td>567</td>
<td>208</td>
</tr>
<tr>
<td>Ohio</td>
<td>85</td>
<td>18</td>
</tr>
<tr>
<td>Colorado</td>
<td>199</td>
<td>70</td>
</tr>
</tbody>
</table>

returning the grid:

**Member Example**

In the following query, the Uda function searches a member (itself included) for descendant members having a UDA of Major Market:

```
SELECT
    {{[Measures].[Sales], [Measures].[Profit]} ON COLUMNS,
    {UDA([East], "Major Market")} ON ROWS
FROM Sample.Basic
WHERE ([Year].[Jul], [Product].[Cola])
```

returning the grid:

Table 6-137  Output Grid from MDX Example

<table>
<thead>
<tr>
<th>(axis)</th>
<th>Sales</th>
<th>Profit</th>
</tr>
</thead>
<tbody>
<tr>
<td>East</td>
<td>2248</td>
<td>1156</td>
</tr>
<tr>
<td>New York</td>
<td>912</td>
<td>370</td>
</tr>
<tr>
<td>Massachusetts</td>
<td>665</td>
<td>564</td>
</tr>
<tr>
<td>Florida</td>
<td>286</td>
<td>104</td>
</tr>
</tbody>
</table>

**Union**

Returns the union of two input sets, optionally retaining duplicates.

**Syntax**

```
Union ( set1, set2 [,ALL] )
```
Parameters

**set1**
A set to join with set2.

**set2**
A set to join with set1.

**ALL**
If the optional ALL keyword is used, duplicates are retained.

Notes

Duplicates are eliminated by default from the tail of the set. The optional ALL keyword retains duplicates. The two input sets must have identical dimension signatures. For example, if set1 consists of dimensions Product and Market, in that order, then set2 should also consist of Product followed by Market.

Example

**Example 1**
The expression

```
Union( Siblings([Old Fashioned]), {[Sarsaparilla], [Birch Beer]})
```

returns the set

```
{ [Old Fashioned], [Diet Root Beer], [Sarsaparilla], [Birch Beer] }
```

**Example 2**
The expression

```
Union( Siblings([Old Fashioned]), {[Sarsaparilla], [Birch Beer]}, ALL)
```

returns the set

```
{ [Old Fashioned], [Diet Root Beer], [Sarsaparilla], [Birch Beer],
  [Sarsaparilla], [Birch Beer] }
```

**Example 3**
The following query

```
SELECT
  CrossJoin (   
    Union (       
      Siblings ( [Old Fashioned] ) ,
      { [Root Beer], [Cream Soda] } ,
      { [Budget], [Variance] } )  
  )
```
ON COLUMNS
from Sample.Basic

returns the grid

Table 6-138  Output Grid from MDX Example

<table>
<thead>
<tr>
<th>Old Fashioned</th>
<th>Diet Root Beer</th>
<th>Sarsaparilla</th>
<th>Birch Beer</th>
<th>Root Beer</th>
<th>Cream Soda</th>
</tr>
</thead>
<tbody>
<tr>
<td>Budget</td>
<td>Varian ce</td>
<td>Budget</td>
<td>Varian ce</td>
<td>Budget</td>
<td>Varian ce</td>
</tr>
<tr>
<td>11640</td>
<td>-4439</td>
<td>14730</td>
<td>-2705</td>
<td>5050</td>
<td>-414</td>
</tr>
<tr>
<td>14730</td>
<td>-2705</td>
<td>5050</td>
<td>-414</td>
<td>4530</td>
<td>-438</td>
</tr>
<tr>
<td>14730</td>
<td>-2705</td>
<td>5050</td>
<td>-414</td>
<td>4530</td>
<td>-438</td>
</tr>
<tr>
<td>14730</td>
<td>-2705</td>
<td>5050</td>
<td>-414</td>
<td>4530</td>
<td>-438</td>
</tr>
<tr>
<td>14730</td>
<td>-2705</td>
<td>5050</td>
<td>-414</td>
<td>4530</td>
<td>-438</td>
</tr>
<tr>
<td>14730</td>
<td>-2705</td>
<td>5050</td>
<td>-414</td>
<td>4530</td>
<td>-438</td>
</tr>
</tbody>
</table>

UnixDate

To the given Julian date, get its UNIX date.

Syntax

UnixDate ( juliandate )

Parameters

juliandate
A number representing the Julian date. This number is a continuous count of days and fractions elapsed since noon Universal Time on January 1, 4713 BC in the proleptic Julian calendar.

Note:

For Excel workbooks using 1900 date system, (JulianDate – 2415018.50) gets the sequential serial number as per 1900 date system.

Notes

- This function is useful in converting the Julian date to UNIX date.
- In the 1900 date system, the first day that is supported is January 1, 1900. When you enter a date, the date is converted into a serial number that represents the number of elapsed days since January 1, 1900. For example, if you enter July 5, 1998, Microsoft Excel converts the date to the serial number 35981. By default, Microsoft Excel for Windows uses the 1900 date system.

Return Value

This function returns date a number representing the input date between January 1, 1970 and Dec 31, 2037. The number is the number of seconds elapsed since midnight, January 1, 1970. To retrieve this number, use any of the following functions: Today(), TodateEx(), GetFirstDate(), GetLastDate(), DateRoll().

Date-Time type attribute properties of a member can also be used to retrieve this number. For example: Product.currentmember.[Intro Date] returns the Introduction
or release date for the current product in context. [Cola].[Intro Date] returns the introduction or release date for the “Cola” product.

See Also
JulianDate

Upper

Converts lower-case string to upper case.

Syntax

Upper ( string )

Parameters

string
Input string.

Example

Upper(string)

returns STRING

See Also
Lower

Value

Returns a value for the specified member or tuple.

Syntax

tuple[.Value]

member[.Value]

Parameters

tuple
A tuple for which to return a value.

member
A member for which to return a value.

Notes

The VALUE keyword is optional. In Example 2, the value of Sales can be represented either as [Sales].VALUE or [Sales]. Any value expression (for example, the value
expressions supplied to functions such as Filter, Order, or Sum) has an implicit Value function in it. The expression \([Qtr1] \leq 0.00\) is a shortcut for \([Qtr1].VALUE \leq 0.00\).

**Example**

**Example 1**

\([Sales].Value\)

<table>
<thead>
<tr>
<th>Returns the value of the Sales measure.</th>
</tr>
</thead>
</table>

\(([Product].CurrentMember, [Sales]).Value\)

<table>
<thead>
<tr>
<th>Returns the value of the Sales measure for the current member of the Product dimension.</th>
</tr>
</thead>
</table>

**Note:**

The Value keyword is optional. The above expressions can also be entered as:

\([Sales]\)

<table>
<thead>
<tr>
<th>Which is equivalent to [Sales].Value</th>
</tr>
</thead>
</table>

\(([Product].CurrentMember, [Sales])\)

<table>
<thead>
<tr>
<th>Which is equivalent to ([Product].CurrentMember, [Sales]).VALUE</th>
</tr>
</thead>
</table>

**Example 2**

The following query sorts level-0 members of the Product dimension by the value of Sales, in descending order.

```
SELECT
([Sales])
ON COLUMNS,
    Order(([Product].Levels(0).Members,
        [Sales].VALUE, BDESC)
ON ROWS
FROM Sample.Basic
```

This query returns the grid:
Table 6-139  Output Grid from MDX Example

<table>
<thead>
<tr>
<th>(axis)</th>
<th>Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>100-10</td>
<td>62824</td>
</tr>
<tr>
<td>300-10</td>
<td>46956</td>
</tr>
<tr>
<td>200-10</td>
<td>41537</td>
</tr>
<tr>
<td>200-20</td>
<td>38240</td>
</tr>
<tr>
<td>200-20</td>
<td>38240</td>
</tr>
<tr>
<td>300-30</td>
<td>36969</td>
</tr>
<tr>
<td>300-30</td>
<td>36969</td>
</tr>
<tr>
<td>400-10</td>
<td>35799</td>
</tr>
<tr>
<td>400-20</td>
<td>32670</td>
</tr>
<tr>
<td>100-20</td>
<td>30469</td>
</tr>
<tr>
<td>100-20</td>
<td>30469</td>
</tr>
<tr>
<td>200-30</td>
<td>17559</td>
</tr>
<tr>
<td>300-20</td>
<td>17480</td>
</tr>
<tr>
<td>400-30</td>
<td>15761</td>
</tr>
<tr>
<td>100-30</td>
<td>12841</td>
</tr>
<tr>
<td>200-40</td>
<td>11750</td>
</tr>
</tbody>
</table>

WithAttr

Returns all base members that are associated with an attribute member of the specified type.

Syntax

WithAttr ( member, character_string_literal, value_expression )

Parameters

member
The top member of an attribute dimension.

character_string_literal
An operator. Must be enclosed in double quotation marks.
The following operators are supported:

- > Greater than
- >= Greater than or equal to
- < Less than
- <= Less than or equal to
- = = Equal to
- <> or != Not equal to
- IN In
**value_expression**
An attribute value described by a value expression. The expression must evaluate to a numeric value for numeric/date attributes and must evaluate to a string for text valued attributes. Can also be an exact character string (not case-sensitive) enclosed in double quotation marks.

**Example**
The following query

```
SELECT Withattr([Pkg Type], ",==", "Can")
ON columns
FROM Sample.Basic
```

returns products that are packaged in a can:

<table>
<thead>
<tr>
<th>Cola</th>
<th>Diet Cola</th>
<th>Diet Cream</th>
</tr>
</thead>
<tbody>
<tr>
<td>22777</td>
<td>5708</td>
<td>11093</td>
</tr>
</tbody>
</table>

**See Also**

*Attribute*

**WithAttrEx**

Returns the set of base members that are associated with a specified varying attribute member or dimension, given the perspective setting and the predicate.

**Syntax**

```
WithAttrEx ( member, options, character_string_literal, value_expression, ANY, tuple|member[,tuple|member] )
```

**Parameters**

- **member**
The top member of an attribute dimension.

- **character_string_literal**
An operator. Must be enclosed in double quotation marks. The following operators are supported:
  - > Greater than
  - >= Greater than or equal to
  - < Less than
  - <= Less than or equal to
  - == Equal to
• <> or != Not equal to
• IN

value_expression
An attribute value described by a value expression. The expression must evaluate to a numeric value for numeric/date attributes and must evaluate to a string for text valued attributes. Can also be an exact character string (not case-sensitive) enclosed in double quotation marks.

ANY
The keyword ANY.

tuple | member
Level 0 start tuple (or member) of the independent dimension set. The tuple must contain all the discrete dimensions followed by the continuous dimension members, in the same order that the continuous range has been defined.

tuple | member
Optional level 0 end tuple (or member) of the independent dimension set. The tuple must contain all the discrete dimensions followed by the continuous dimension members, in the same order that the continuous range has been defined.

Example
Consider the following scenario: Products are packaged under different ounces over time and the market state, according to the marketing strategy of the company. Ounces is defined as a varying attribute for the Product dimension, to capture the varying attribute association over the continuous Year dimension and the discrete Market dimension.

Year and Market are the independent dimensions, and level-0 tuple months (for example, Jan) combined with a market state (for example, California) is a perspective for which the varying attribute association is defined.

The following query analyzes sales performance of products packaged in units of 20 ounces or greater any time from Jan to Dec in New York, over all quarters. This is the perspective view, which restates the sales according to the packaging strategy in July.

WITH PERSPECTIVE (Jul) FOR Ounces
SELECT
{Qtr1, Qtr2, Qtr3, Qtr4}
ON COLUMNS,
{WithattrEx(Ounces, ">=", 20, ANY, ([New York], Jan), ([New York], Dec))}
ON ROWS
FROM app.db
WHERE
(Sales, Ounces, [New York])
;

See Also
AttributeEx
xTD

Returns period-to-date values.

Syntax

\[ xTD \left( \text{[member]} \right) \]

Parameters

\text{xTD}

Values:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTD</td>
<td>History-To-Date (H-T-D)</td>
</tr>
<tr>
<td>YTD</td>
<td>Year-To-Date</td>
</tr>
<tr>
<td>STD</td>
<td>Season-To-Date</td>
</tr>
<tr>
<td>PTD</td>
<td>Period-To-Date</td>
</tr>
<tr>
<td>QTD</td>
<td>Quarter-To-Date</td>
</tr>
<tr>
<td>MTD</td>
<td>Month-To-Date</td>
</tr>
<tr>
<td>WTD</td>
<td>Week-To-Date</td>
</tr>
<tr>
<td>DTD</td>
<td>Day-To-Date</td>
</tr>
</tbody>
</table>

\text{member}

Member specification. Should be a member from the time dimension.

Notes

- \( xTD \left( \text{[member]} \right) \) is equivalent to \( \text{PeriodsToDate} \left( \text{layer}, \text{[member]} \right) \) where \text{layer} is assumed to be the value set in the corresponding Dynamic Time Series member in the database outline.

  For example, in Sample Basic, \( \text{QTD} \left( \text{[member]} \right) \) is equivalent to \( \text{PeriodsToDate} \left( \text{Year.Generations(2)}, \text{[member]} \right) \), because Q-T-D is Generation 2 in the Year dimension.

- The xTD functions YTD, QTD, MTD, etc. are not relevant for use in aggregate storage databases, because the xTD functions assume that Dynamic Time Series members are defined in the outline. Dynamic Time Series members are not supported for aggregate storage database outlines.

  You can use the \text{PeriodsToDate} function with aggregate storage databases in place of the xTD functions.

  For example,

  \[
  \text{YTD(May)} \text{ is equivalent to } \text{PeriodsToDate(Year.Generations(1), May)}
  \]

  \[
  \text{QTD(May)} \text{ is equivalent to } \text{PeriodsToDate(Year.Generations(2), May)}
  \]
Example

QTD([Feb])

returns the set \{[Jan], [Feb]\}.

\textbf{QTD([Feb]) is equivalent to} \textbf{PeriodsToDate(Year.Generations(2), [Feb])}, because the dynamic-time-series member Q-T-D is defined as Generation 2 of the Year dimension.

HTD([May])

returns the set \{[Jan], [Feb], [Mar], [Apr], [May]\}.

\textbf{HTD([May]) is equivalent to} \textbf{PeriodsToDate(Year.Generations(1), [May])}, because the dynamic-time-series member H-T-D is defined as Generation 1 of the Year dimension.

\begin{center}
\textit{Note:}
\end{center}

If a dynamic-time-series member is not defined, an empty set is returned.

PTD([Feb])

returns an empty set, because the dynamic-time-series member P-T-D is not enabled in the outline.
Query Logging Overview

Query logging provides a way for Essbase administrators to track query patterns of an Essbase database. The query log file tracks queries performed against the database from Smart View, Report Writer, or Grid-API clients. Query logging can track generation or level numbers of members belonging to specific generations or levels. Query logging also offers the flexibility to exclude logging of certain dimensions and members belonging to certain generations or levels. Because the query log file output is an XML document, you can import the log file to any XML-enabled tool to view the log.

Note:
You can import the .XML file to Microsoft Access or Microsoft Excel. However, you must first shut down the database.

For details about the query log file structure, refer to querylog.dtd in the ARBORPATH/bin directory.

Query logging is available for both block storage and aggregate storage databases. To enable query logging, create a query log file and add to the file the settings that control how query logging is performed. You must create a query log file for each database that requires query logging. If the query log file is missing or the QUERYLOG setting is off, query logging is disabled.

Query Logging Settings Procedure

The following steps explain how to create a query log settings file. To see a sample query log file, see Query Logging Sample File.

To enable query logging:

1. In the ARBORPATH/App\appname\dbname directory of Essbase, create a query log settings file.
The settings file must be named `dbname.cfg`, where `dbname` matches the name of the database. For example, the query log settings file for Sample Basic is `basic.cfg`. For databases in Unicode-mode applications, the query log file must be encoded in UTF-8 and include the UTF-8 signature.

2. In the settings file, specify required and optional elements, using the syntax from the section Query Logging Syntax:
   - The dimension for which you want to log queries (QUERYLOG [dimension_name]).
   - **Optional**: The setting to log generation or level numbers for members of specified generations or levels in a dimension (QUERYLOG GENERATION generation-range or QUERYLOG LEVEL level-range).
   - **Optional**: The setting to exclude logging of members from specified generations or levels in a dimension (QUERYLOG NONE GENERATION generation-range or QUERYLOG NONE LEVEL level-range).
   - **Optional**: The location where the query log file is created (QUERYLOG LOGPATH path-expression).
   - **Optional**: The format of the log file output (QUERYLOG LOGFORMAT CLUSTER | TUPLE).
   - **Optional**: The size of the log file (QUERYLOG LOGFILESIZE n)
   - **Optional**: The size of all log files (QUERYLOG TOTALLOGFILESIZE n).
   - A setting to enable or disable query logging the next time the application starts (QUERYLOG ON | OFF).

3. Restart the database to accept the settings.

   ✍️ **Note:**
   Restart after creating a file or changing any entries in a file.

4. After query logging is enabled, review the log entries in the query log file, `dbname.qlg`.
   For example, you can view the output of the log file to analyze how many times a certain member has been queried. You can use a UTF-8-enabled editor to view query log files for databases in Unicode-mode applications.

**Query Log Settings File Syntax**

The query log settings filename must be of the form `dbname.cfg`, where `dbname` represents the name of a database. The `dbname.cfg` file must be located in the ARBORPATH\App\appname\dbname directory of Essbase. The `dbname.cfg` file consists of the following syntax:

```
QUERYLOG [dimension_name]
QUERYLOG NONE GENERATION generation-range
QUERYLOG NONE LEVEL level-range
QUERYLOG GENERATION generation-range
QUERYLOG LEVEL level-range
QUERYLOG LOGPATH path-expression
```
### QUERYLOG Parameters

<table>
<thead>
<tr>
<th>QUERYLOG Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[dimension_name]</td>
<td>Identifies the dimension name to be tracked. The brackets around the dimension name are required. QUERYLOG [dimension_name] logs all members of a dimension. For example, QUERYLOG [Product] tracks all members of the Product dimension. Each dimension must be specified in a separate QUERYLOG [dimension_name] setting.</td>
</tr>
<tr>
<td>NONE GENERATION generation-range</td>
<td>Prevents tracking of members from the specified generation range. For example, QUERYLOG NONE GENERATION 2 excludes tracking of all members from generation 2 of the named dimension.</td>
</tr>
<tr>
<td>NONE LEVEL level-range</td>
<td>Prevents tracking of members from the specified level range. For example, QUERYLOG NONE LEVEL 0-2 excludes tracking of all members of levels 0, 1, and 2 of the named dimension.</td>
</tr>
<tr>
<td>GENERATION generation-range</td>
<td>Tracks members of the specified generation range by generation number, rather than by member name. For example, QUERYLOG GENERATION 5-7 logs members of generations 5, 6, and 7 of the named dimension by their generation number in the log file.</td>
</tr>
<tr>
<td>LEVEL level-range</td>
<td>Tracks members of the specified level range by level number, rather than by member name. For example, QUERYLOG LEVEL -3 logs members of levels 0, 1, 2, and 3 of the named dimension by their level number in the log file.</td>
</tr>
<tr>
<td>LOGPATH path-expression</td>
<td>Specifies the location of the output log file. The log file name is dbname00001.qlg; for example, basic00001.qlg. Examples of the log path are QUERYLOG LOGPATH /usr/local/Essbase\logs/ and QUERYLOG LOGPATH d:\Essbase\logs\querylogs. You must include a backslash \ (for Windows directories) or forward slash / (for UNIX directories) at the end of the path expression; otherwise, the query log file is not created. By default, the location for the log output file is the ARBORPATH\App\appname\dbname\ directory. If the LOGPATH path-expression setting is missing, the default is used. Essbase writes log information to the query log file after an application stops running.</td>
</tr>
</tbody>
</table>
Table 7-1  (Cont.) QUERYLOG Parameters

<table>
<thead>
<tr>
<th>QUERYLOG Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOGFORMAT CLUSTER</td>
<td>TUPLE</td>
</tr>
<tr>
<td>LOGFILESIZE $n$</td>
<td>Specifies the maximum size of an individual query log file in megabytes (MB). The minimum value is 1 MB. The maximum value is 2048 MB (2 GB). If the LOGFILESIZE setting is missing, then, by default, the query log file size is 1 MB. If an initial query log file size exceeds the specification, log information is added to a new query log file. Each time a new file is created, the filename is incremented by one.</td>
</tr>
<tr>
<td>TOTALLOGFILESIZE $n$</td>
<td>Specifies the maximum size of all query log files combined in megabytes (MB). The minimum value is 512 MB (1/2 GB). The maximum value is 4095 MB. If the TOTALLOGFILESIZE setting is missing, then, by default, the total query log file size is 1024 MB (1 GB). Query log files are created until the file size total exceeds the specified maximum. When the maximum is exceeded, a message is displayed and query logging automatically turns off.</td>
</tr>
<tr>
<td>ON</td>
<td>OFF</td>
</tr>
</tbody>
</table>

Generation-range and level-range values are represented in one of the following ways:

Table 7-2  Generation and Level Range Specifications

<table>
<thead>
<tr>
<th>Generation-Range or Level-Range Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>A specific generation or level number. For example, QUERYLOG NONE GENERATION 2 excludes generation 2 from query logging.</td>
</tr>
</tbody>
</table>
Table 7-2  (Cont.) Generation and Level Range Specifications

<table>
<thead>
<tr>
<th>Generation-Range or Level-Range Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x-y</td>
<td>All generations or levels inclusive of number x through number y. For example, QUERYLOG GENERATION 1-3 or QUERYLOG LEVEL 1-3 includes generation or level numbers 1, 2, and 3.</td>
</tr>
<tr>
<td>-x</td>
<td>For generation-range, all generations within the range 1 through x. For level-range, all levels within the range 0 through x. For example, QUERYLOG GENERATION -2 includes generations 1 and 2. QUERYLOG LEVEL -3 includes levels 0, 1, 2, and 3.</td>
</tr>
<tr>
<td>x-</td>
<td>For generation-range, all generations within the range from number x through the highest generation. For level-range, all levels within the range from number x through the highest level. For example, QUERYLOG Level 1- includes levels 1, 2, 3 and so on up to the highest level.</td>
</tr>
</tbody>
</table>

Notes

- When query logging is enabled, queries to the database may be slower. Performance depends on how many members are being tracked and the size of the query.
- If the settings file name does not match the name of the database or the settings file is located in a place other than the `ARBORPATH\App\appname\dbname` directory, Essbase ignores query logging.
- If, in the settings, QUERYLOG ON is missing or if QUERYLOG OFF is set, query logging is disabled.
- If generation and level settings cause contradictions in the settings file, the following precedence rules apply:
  - generation numbers (highest priority)
  - level numbers
  - member names (lowest priority)

For example, if a member belongs to both level 1 and generation 2 and the settings QUERYLOG GENERATION 2 and QUERYLOG NONE LEVEL 1 are in the settings file, the generation setting takes precedence, and members of generation 2 are logged by generation number.

Tips

- To view query log output easily, change the file extension `.QLG` to `.XML`, and then using the Internet Explorer or Netscape browser view the `.XML` file.
Note:

You can import the .XML file to Microsoft Access or Microsoft Excel. However, you must first shut down the database.

- If Essbase is not producing a query log file as expected, view the `dbname.log` file in the `ARBORPATH\App\AppName` directory to search for query log messages.

Query Logging Sample File

```
# Log the Product dimension
QUERYLOG [Product]
# Log the Market dimension
QUERYLOG [Market]
# Log members of generation 2 of Market by generation number
QUERYLOG GENERATION 2
# Display log output in cluster format
QUERYLOG LOGFORMAT CLUSTER
# Create log file in C:\QUERYLOG\ 
QUERYLOG LOGPATH C:\QUERYLOG\ 
# Start a new log file after an individual log file size reaches 2 MB
QUERYLOG LOGFILESIZE 2
# Turn off query logging after the total size of all log files reaches 1024 MB (1 GB)
QUERYLOG TOTALLOGFILESIZE 1024
# Enable query logging
QUERYLOG ON
```

Query Logging Sample Output

The following sample Query Log Output shows an example of how log settings look in a log file. In the example, the log settings show that all members of Product are logged and that members of generation 2 of Market are logged by generation number. The log format is cluster and the log path is `C:\QUERYLOG\`.

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<root>
  <session>
  </session>
</root>
```
Description

A query is a unit of retrieval from the user perspective. The way a user may perceive a query is different than how the server analyzes and executes a query. Even if a user performs a single retrieval, in order for the server to efficiently execute the logical query, the server splits the query into a number of subqueries to execute. Therefore, a single retrieval from the user perspective may actually consist of several subqueries from the server perspective. These subqueries are reflected in the query log.

Sample Cluster Output

The following segment shows an example of how queries are logged in cluster format. The username is listed along with the query execution date and the start time of the query. Each cluster contains two dimension entries. The first cluster shows that members 100 and 200 of the Product dimension were queried. The second cluster shows that member 300 of Product and Generation 2 of Market were queried. The elapsed time to perform the query is also provided.

<query>
    <user>User1</user>
    <time>Tue Aug 13 12:29:49 2002</time>
    <subquery>
        <cluster size="2">
            <dim size="2">
                <member>100</member>
                <member>200</member>
            </dim>
            <dim size="1">
                <member>Market</member>
            </dim>
        </cluster>
    </subquery>
    <subquery>
        <cluster size="2">
            <dim size="1">
                <member>300</member>
            </dim>
            <dim size="2">
                <member>Market</member>
                <generation>2</generation>
            </dim>
        </cluster>
    </subquery>
    <elapsedtime>0.016 seconds</elapsedtime>
</query>
Sample Tuple Output

The following segment shows an example of how queries are logged in tuple format. The username is listed along with the query execution date and the start time of the query. Note that each member of Product is displayed with Market. Each possible member combination is displayed for a given query. The elapsed time to perform the query is also provided.

<query>
  <user>User1</user>
  <time>Tue Aug 13 12:28:14 2002</time>
  <subquery>
    <tuples>
      <tuple>
        <member>100</member>
        <member>Market</member>
      </tuple>
    </tuples>
  </subquery>
  <subquery>
    <tuples>
      <tuple>
        <member>200</member>
        <member>Market</member>
      </tuple>
    </tuples>
  </subquery>
  <elapsedtime>0.02 seconds</elapsedtime>
</query>