Oracle® Essbase

Calculation and Query Reference for Oracle Essbase

F17644-04
Sept 2020

Oracle Essbase Calculation and Query Reference for Oracle Essbase,

F17644-04

Copyright © 2019, 2020, Oracle and/or its affiliates.

Primary Author: Essbase Information Development Team

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

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

U.S. GOVERNMENT END USERS: Oracle programs (including any operating system, integrated software, any programs embedded, installed or activated on delivered hardware, and modifications of such programs) and Oracle computer documentation or other Oracle data delivered to or accessed by U.S. Government end users are "commercial computer software" or "commercial computer software documentation" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, reproduction, duplication, release, display, disclosure, modification, preparation of derivative works, and/or adaptation of i) Oracle programs (including any operating system, integrated software, any programs embedded, installed or activated on delivered hardware, and modifications of such programs), ii) Oracle computer documentation and/or iii) other Oracle data, is subject to the rights and limitations specified in the license contained in the applicable contract. The terms governing the U.S. Government’s use of Oracle cloud services are defined by the applicable contract for such services. No other rights are granted to the U.S. Government.

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

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Inside are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Epyc, and the AMD logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

Contents

3 Calculation Commands

Accessibility and Support

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.

1

Calculation and Query Reference Overview

You can use a wide variety of commands and functions to calculate and query Essbase cubes. This reference is intended for advanced users who need detailed information and examples about Essbase calculation functions, calculation commands, and MDX.

About the Calculation and Query Reference

The Calculation and Query Reference describes commands and functions you can use to calculate and query Oracle Essbase cubes. This reference is intended for advanced users who need detailed information and examples about calculation functions, calculation commands, and MDX.

This document provides examples based mostly on the Sample Basic cube, provided with Essbase as a template you can build into a cube. The Sample application, as well as more samples you can build, are available in the Applications > Demo Samples section of the gallery. The gallery is available in the Files section of Essbase. See Explore the Gallery Templates.

What You Should Know Before You Start

To use this document, you need the following:

- A working knowledge of the operating system.
- An understanding of Essbase concepts and features.
- An understanding of the typical database administration requirements and tasks, including calculation, querying, security, and maintenance.

Syntax Conventions

This document uses several formatting styles to indicate actions you should take or types of information you need.

Table 1-1 Syntax Conventions

Syntax	Purpose	Example
UPPERCASE	Calculation command or function names in syntax.	`DATAEXPORT`

Table 1-1 (Cont.) Syntax Conventions

Syntax	Purpose	Example
italic	Terms, such as parameters, that you replace with a value	`AGG (dimlist);`
" "	Double quotation marks enclose text parameters or single parameters that include a space	`@CHILDREN ("New York")`
( )	Parentheses are used in a couple of ways: • To enclose function parameters • To show the order of execution of the enclosed operations	`@POWER (14,3)` `(a + b) * c`
/* ... */	Comment markers in calculation scripts. The /* ... */ comment markers indicate the enclosed text should be ignored in processing.	`/*Get results*/`
;	Statement terminator	`AGG ("Product");`
[]	Brackets enclose optional parameters in syntax . Used with OR symbol | if there is more than one optional parameter. Do not type brackets or the OR symbol |.	`@RANGE (mbrName [, rangeList])`
[, *numeric*] [, "*text*"]	Indicates an optional numeric (no quotes) or character (quoted) parameter and the comma which must precede the optional parameter. Do not type the brackets.	[, *year*] [, "*columnName*"]
|	Syntax: OR. Separates alternatives from which you choose only one. Do not type the OR symbol.	`SET AGGMISSG ON | OFF`
@	Essbase calculation functions: Precedes many function names	`@ABS`
->	Essbase calculation functions: Cross-dimensional operator (a hyphen followed by a greater-than sign) points to data values of specific member combinations -> (cross-dimensional operator)	`Price -> West = AVGRANGE`

About Aggregate Storage Cubes

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

Consider using the aggregate storage model if the following is true:

- Your cube is sparse and has many dimensions, and/or the dimensions have many levels of members.
- The cube 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 cube 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 cubes:

- **MDX**—Used for querying on block storage and aggregate storage cubes. 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, because calculation scripts are not relevant to aggregate storage.
- **Calculation functions**—Not relevant to aggregate storage cubes. Instead, MDX formulas can be written using MDX numeric-value expressions. Only the Accounts dimension can have formulas in aggregate storage cubes.

2 Calculation Functions

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

Calculation and Member Hierarchy

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

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
```

See also: Relationship Functions

Function Parameters

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

> **Note:**
>
> 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.

Table 2-1 Function Parameters

Parameter	Description
attDimName	A single attribute dimension name specification. `@WITHATTR(Ounces,"<",16)`

attMbrName	A single attribute member name specification.
	
	`@ATTRIBUTE(Can)`
	`@ATTRIBUTEVAL(Ounces)`
	`@WITHATTR("Pkg Type","= =",Can)`

Table 2-1 (Cont.) Function Parameters

Parameter	Description

dimName	A single dimension name specification.
	
	`@CURLEV(Accounts)`
	`@CURGEN(Year)`
	`@PARENT(Measures,Sales)`

expList	A comma-delimited list of member names, variable names, functions, and numeric expressions, all of which return numeric values.
	
	`@MAX(Jan,Feb,100,Apr-May)`
	`@MIN(Oct:Dec)`
	`@COUNT(SKIPNONE,@RANGE(Sales,@CHIL DREN(Product)))`

expression	Any mathematical or numeric expression that is valid within Essbase and that, when calculated, returns a numeric value. This definition of *expression* also includes parameters such as *numDigits*, *generation*, and *level*, and other similar parameters for the financial group of functions, such as *rateMbrConst* and *lifeMbrConst.*
	
	`@ABS(Actual-Budget)`
	`@ROUND(Sales / 10.0 + 100)`

genLevName	Generation or level name specification.
	
	`@DESCENDANTS(Market,Regions)`
	`@RELATIVE(Qtr1,Month)`

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.
	
	`@ANCESTORS(Sales,-2)`
	`@SANCESTVAL(Product,2,Sales)`

mbrList	A comma-delimited list of members. `@ISMBR(New_York,Boston,Chicago)`

Table 2-1 (Cont.) Function Parameters

Parameter	Description

mbrName	Any valid single member name or member combination, or a function that returns a single member or member combination. This definition also includes similar parameters, such as *balanceMbrName*, *costMbr*, and *cashflowMbr*, for the financial group of functions.
	
	`@GEN(Actual)`
	`@CHILDREN(Product)`
	`@MAXRANGE(@ANCESTORS(Qtr4),Jan:Dec)`
	
	For functions that expect a single member name (for example, @DESCENDANTS and @CHILDREN), if a member combination is provided, Essbase uses the first member in the combination. For example, if *mbrName* is Utah->Sales, Essbase uses Utah.

n	A positive or negative integer value.
	
	`@NEXT(2,Jan:Dec)`
	`@SHIFT(3)`

propertyName	Dimension property name.
	
	`@PROPERTY(Market,Size)`
	`@ISPROPERTY([Market].[New York],Size,Medium)`

propertyValue	Optional. Member property value. The value must match the data type of the dimension property specified in *propertyName*.
	
	`@PROPERTY(Market,Size,Medium)`
	`@PROPERTYBVAL(“New York”,Color)`

Table 2-1 (Cont.) Function Parameters

Parameter	Description

rangeList	A valid member name, a comma-delimited list of member names, member set functions, and range functions from the same dimension. If *rangeList* 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 *rangeList* also includes *mbrList*.
	
	`@ACCUM(Q189:Q491)`
	`@MAXRANGE(Sales,@CHILDREN(Qtr1))`

tag	Any valid account tag defined in the current database including First, Last, Average, Expense, and Two-Pass. `@ISACCTYPE("EXPENSE")` To ensure that the tag is resolved as a string rather than a member name, enclose the tag in quotation marks.
XrangeList	Similar to *rangeList*, 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 @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. See also Range List Parameters.

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

Operator	Description
+	Adds

Table 2-2 (Cont.) Mathematical Operators

Operator	Description
-	Subtracts
*****	Multiplies
/	Divides
%	Evaluates percentage, for example: *Member1%Member2* evaluates *Member1* as a percentage of *Member2*.
( )	Controls the order of calculations and nests equations and formulas

Conditional and Logical Operators

Conditional operators build logical condition into calculations.

Table 2-3 Conditional and Logical Operators

Operator	Description
IF | ELSE | ELSEIF | ENDIF	Tests conditions and calculates a formula based on the success or failure of the test
>	Data value is greater than
>=	Data value is greater than or equal to
<	Data value is less than
<=	Data value is less than or equal to
= =	**If** data value is equal to
< > or **!=**	Data value is not equal to
AND	Logical AND linking operator for multiple value tests. Result is TRUE if both conditions are TRUE. Otherwise the result is FALSE.[1]
OR	Logical OR linking operator for multiple value tests. Result is TRUE if either condition is TRUE. Otherwise the result is FALSE.[2]
NOT	Logical NOT operator. Result is TRUE if condition is FALSE. Result is FALSE if condition is TRUE.[3]

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: ->

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

Calculation/Operation	Result
X + #MISSING	X
X – #MISSING	X
#MISSING – X	-X
X * #MISSING	#MISSING
X / #MISSING	#MISSING
#MISSING / X	#MISSING
X / 0	#MISSING
X % #MISSING	#MISSING
#MISSING % X	#MISSING
X % 0	#MISSING
X == #MISSING	False, unless X is #MISSING
X != #MISSING	True, unless X is #MISSING
X <> #MISSING	True, unless X is #MISSING
(X <= #MISSING)	(X <=0)
(X >= #MISSING)	(X >=0) or (X == #MISSING)
(X > #MISSING)	(X > 0)
(X < #MISSING)	(X < 0)
X AND #MISSING:	#MISSING
1 AND #MISSING (1 represents any nonzero value)	0
0 AND #MISSING	#MISSING
#MISSING AND #MISSING	
X OR #MISSING:	1
1 OR #MISSING (1 represents any nonzero value)	#MISSING
0 OR #MISSING	#MISSING
#MISSING OR #MISSING	
IF (#MISSING)	IF (0)
f (#MISSING)	#MISSING for any Essbase function of one variable
f (X)	#MISSING for any X not in the domain of f, and any Essbase function of more than one variable (except where specifically noted)

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

Function	Condition Tested
@ISACCTYPE	Whether the current member has a particular *accounts tag*.
@ISANCEST	Whether the current member is an ancestor of *member*.
@ISCHILD	Whether the current member is a child of *member*.

Table 2-5 (Cont.) Boolean Functions

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

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

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

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

Function	Return Value
@ABS	Absolute value of *expression*.
@AVG	Average of all values in *expList*.

Table 2-7 (Cont.) Mathematical Functions

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

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-dimension 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

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

Table 2-8 (Cont.) Member Set Functions

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

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

Function	Return Value
@ACCUM	The sum of values of a specified member across a range
@AVGRANGE	The average of values of a specified member across a range
@COMPOUND	The compound interest of values of a specified member across a range, calculated at a specified rate
@COMPOUNDGROWTH	A series of values that represent the compound growth of the specified member across a range of members, calculated at a specified rate
@CURRMBRRANGE	A range of members that is based on the relative position of the member combination Essbase is currently calculating
@DECLINE	Depreciation of a member over a specified period, calculated using the declining balance method

Table 2-9 (Cont.) Range and Financial Functions

Function	Return Value
@DISCOUNT	Discounted values of a specified member, calculated at a specified rate, across a range of values from the time dimension
@GROWTH	A series of values that represents the linear growth of the specified value
@INTEREST	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
@IRR	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).
@IRREX	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.
@MAXRANGE	The maximum value of a member across a range of members
@MAXSRANGE	The maximum value of a member across a range of members, with the ability to skip zero and #MISSING values
@MDSHIFT	The next or *n*th member in a range of members, retaining all other members identical to the current member across multiple dimensions
@MINRANGE	The minimum value of a member across a range of members
@MINSRANGE	The minimum value of a member across a range of members, with the ability to skip zero and #MISSING values
@NEXT	The next or *n*th member in a range of members
@NEXTS	The next or *n*th member in a range of members, with the option to skip #MISSING, zero, or both values
@NPV	The Net Present Value of an investment based on a series of payments and income values
@PTD	The period-to-date values of members in the time dimension
@PRIOR	A list of the previous or *n*th previous members in a range of members
@PRIORS	A list of the previous or *n*th previous members in a range of members, with the option to skip #MISSING, zero, or both values

Table 2-9 (Cont.) Range and Financial Functions

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

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= @MOVAVG(Sales, 3, @XRANGE("2001"->Mar, "2003"->Jan));
```

and returns the following members:

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

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	A generation range and a comma-delimited list
@SIBLINGS(Dept01), Dept65**:**Dept73, Total_Dept	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

Example	Description
Jan->Actual->Sales, Dec->Actual->Sales	A comma-delimited list of members from one or more dimensions.
Actual->Jan, @XRANGE(Actual->December, Budget->Mar);	A comma-delimited list and a range.
@XRANGE(Jan->Actual,Dec->Budget);	A @XRANGE function.
@CHILDREN("Colas"),@CHILDREN("West")	A member set function as part of a range list.

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

Function	Allocation Type
@ALLOCATE	Allocates values to lower-level members in one level.
@MDALLOCATE	Allocates values to lower-level members in multiple dimensions.

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

Function	Data Manipulation
@MOVAVG	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.
@MOVMAX	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.

Table 2-13 (Cont.) Forecasting Functions

Function	Data Manipulation
@MOVMED	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.
@MOVMIN	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.
@MOVSUM	Applies a moving sum to a data set. This function modifies the data set for smoothing purposes.
@MOVSUMX	Applies a moving sum to a data set, enabling specification of values for trailing members. This function modifies the data set for smoothing purposes.
@SPLINE	Applies a smoothing spline to a set of data points. A spline is a mathematical curve that is used to smooth or interpolate data.
@TREND	Calculates future values, basing the calculation on curve-fitting to historical values

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

Function	Return Value
@CORRELATION	The correlation coefficient between two parallel data sets
@COUNT	The number of data values in the specified data set
@MEDIAN	The median (middle value) of the specified data set
@MODE	The mode (the most frequently occurring value) in the specified data set
@RANK	The rank (position in the sorted data set) of the specified members or the specified value among the values in the specified data set.
@STDEV	The standard deviation of the specified data set
@STDEVP	The standard deviation of the specified data set, calculated over the entire population
@STDEVRANGE	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.

Table 2-14 (Cont.) Statistical Functions

Function	Return Value
@VARIANCE	The statistical variance of the specified data set (expList), based upon a sample of a population
@VARIANCEP	The statistical variance of the specified data set (expList), based upon the entire population

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.
- @CREATEBLOCK—This function populates 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

Alphabetical List of Calculation Functions		
@ABS	@ISATTRIBUTE	@NEXT
@ACCUM	@ISCHILD	@NEXTS
@ALLANCESTORS	@ISDESC	@NEXTSIBLING
@ALIAS	@ISGEN	@NONEMPTYTUPLE
@ALLOCATE	@ISIANCEST	@NOTEQUAL
@ANCEST	@ISIBLINGS	@NPV
@ANCESTORS	@ISICHILD	@PARENT
@ANCESTVAL	@ISIDESC	@PARENTVAL
@ATTRIBUTE	@ISIPARENT	@POWER
@ATTRIBUTEBVAL	@ISISIBLING	@PREVSIBLING
@ATTRIBUTESVAL	@ISLEV	@PRIOR
@ATTRIBUTEVAL	@ISMBR	@PRIORS
@AVG	@ISMBRUDA	@PTD
@AVGRANGE	@ISMBRWITHATTR	@RANGE
@BETWEEN	@ISPARENT	@RANGEFIRSTVAL

Table 2-15 (Cont.) Calculation Function List

Alphabetical List of Calculation Functions		
@CALCMODE	@ISRANGENONEMPTY	@RANGELASTVAL
@CHILDREN	@ISSAMEGEN	@RANK
@COMPOUND	@ISSAMELEV	@RDESCENDANTS
@COMPOUNDGROWTH	@ISSIBLING	@RELATIVE
@CONCATENATE	@ISUDA	@RELXRANGE
@CORRELATION	@LANCESTORS	@REMAINDER
@COUNT	@LDESCENDANTS	@REMOVE
@CREATEBLOCK	@LEV	@RETURN
@CURGEN	@LEVMBRS	@ROUND
@CURLEV	@LIKE	@RSIBLINGS
@CURRMBR	@LIST	@SANCESTVAL
@CURRMBRRANGE	@LN	@SHARE
@DATEDIFF	@LOG	@SHIFT
@DATEPART	@LOG10	@SHIFTMINUS
@DATEROLL	@LSIBLINGS	@SHIFTPLUS
@DECLINE	@MATCH	@SHIFTSIBLING
@DESCENDANTS	@MAX	@SIBLINGS
@DISCOUNT	@MAXRANGE	@SLN
@ENUMVALUE	@MAXS	@SPARENTVAL
@EQUAL	@MAXSRANGE	@SPLINE
@EXP	@MBRCOMPARE	@STDEV
@EXPAND	@MBRPARENT	@STDEVP
@FACTORIAL	@MDALLOCATE	@STDEVRANGE
@FORMATDATE	@MDANCESTVAL	@SUBSTRING
@GEN	@MDPARENTVAL	@SUM
@GENMBRS	@MDSHIFT	@SUMRANGE
@GROWTH	@MEDIAN	@SYD
@IALLANCESTORS	@MEMBER	@TODATE
@IANCESTORS	@MEMBERAT	@TODATEEX
@ICHILDREN	@MERGE	@TODAY
@IDESCENDANTS	@MIN	@TREND
@ILANCESTORS	@MINRANGE	@TRUNCATE
@ILDESCENDANTS	@MINS	@UDA
@ILSIBLINGS	@MINSRANGE	@VAR
@INT	@MOD	@VARPER
@INTEREST	@MODE	@VARIANCE
@INTERSECT	@MOVAVG	@VARIANCEP
@IRDESCENDANTS	@MOVMAX	@WEIGHTEDSUMX
@IRR	@MOVMED	@WITHATTR
@IRREX	@MOVMIN	@XRANGE
@IRSIBLINGS	@MOVSUM	@XREF
@ISACCTYPE	@MOVSUMX	@XWRITE
@ISANCEST	@NAME	

@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:

```
Sales                  VCR      San_Francisco
                       Jan      Feb      Mar
                       ===      ===
===
Actual               1,323    1,290    1,234
Budget               1,200    1,100    1,100
Variance               123      190      134
```

@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:

```
             FY1997   FY1998   FY1999   FY2000   FY2001   FY2002
            =======  =======  =======  =======  =======  =======
Asset         9,000        0    1,000        0   2 ,500    1,500
Residual        750        0        0        0  #MISSING #MISSING
Life              5        0        3        0  #MISSING #MISSING
Accum Asset #MISSING #MISSING   1,000    1,000    3,500    5,000
```

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.

```
                              Jan
                     Actual       Budget
                     Sales        Sales
                     =====        =====
Market  100-10       4860         5200
        100-20       2372         2610
        100-30       1082         1170
         100         8314         9429  *
        100-20       2372         2610
        200-20       3122         3090
        300-30       2960         2560
```

```
Diet        8454        8673  *
 Product   31538       30954  *
```

See Also

- @IALLANCESTORS
- @ILANCESTORS
- @LANCESTORS

@ALIAS

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 *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:

amount * (@CURRMBR()->*basisMbr*/@SUM(*allocationRange*-> *basisMbr*)

- *spread*:

  Spreads *amount* across *allocationRange*:

  amount * (1/@COUNT(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*:

  amount * (@CURRMBR()->*basisMbr* * .01)

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

  amount + @CURRMBR()->*basisMbr*

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

  amount - @CURRMBR()->*basisMbr*

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

  amount * @CURRMBR()->*basisMbr*

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

  amount/@CURRMBR()->*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.

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:

```
                        Product   Market
                     PY Actual         Budget
                     Jan     Feb       Jan     Feb
                     ===     ===       ===
===
Marketing          5223     5289     3908.60  4493.63
Payroll            4056     4056     3035.28  3446.05
Misc                 75       71       56.13    60.32
  Total Expenses   9354     9416     7000     8000
```

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:

```
Function                        Generated List

@ANCEST(Product,2,Sales)        Colas->Sales, if the current member of
                                Product being calculated is Diet Cola.

@ANCEST(Measures,3,East)        Total Expenses->East, if the current
member of
                                Measures being calculated is Payroll.
```

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:

```
              Sales   SKU Share
              =====   =========
SKU101          510        26.0
SKU102          520        26.5
  Group01      1030        52.5
```

```
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}
      100-30 {Caffeinated:False}
   200
      200-10 {Caffeinated:True}
      200-20 {Caffeinated:True}
      200-30 {Caffeinated:False}
      200-40 {Caffeinated:False}
Caffeinated Attribute {Type: Boolean}
   True
   False
```

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

Current Member	Return Value
100-10	True
100-20	True
100-30	False
100	#MISSING
200-10	True
200-20	True
200-30	False
200-40	False
200	#MISSING
Product	#MISSING

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 2-17 Values Returned for @ATTRIBUTESVAL("Pkg Type") Function

Current Member	Return Value
100-10	Can
100-20	Can
100-30	Bottle
100	(empty string)
200-10	Bottle
200-20	Bottle
200-30	Bottle

Table 2-17 (Cont.) Values Returned for @ATTRIBUTESVAL("Pkg Type") Function

Current Member	Return Value
200-40	Bottle
200	(empty string)
Product	(empty string)

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:

```
              Actual      Year      West
              Profit        Profit Per Ounce
             ========      ================
Cola          4593            382.75
```

Example 2

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

```
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:

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

This example produces the following report:

```
                    Sales      Jan       Actual
                Cola    Diet Cola    Caffeine Free Cola
                ====    =========
==================
California       678       118             145
Oregon           160       140             150
Washington       130       190             #MI
Utah             130       190             170
Nevada            76        62             #MI
  West           234.8     140             155
```

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 @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 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.

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

This example produces the following report:

```
               Sales   Colas   Actual
                 Jan     Feb     Mar
                 ===     ===     ===
California       941     899     927
Oregon           450     412     395
Washington       320     362     377
Utah             490     488     476
Nevada           138     137     138
  West         467.8   459.6   462.6
```

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

```
@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.

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:

```
@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.

```
@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

(axis)	Jan	Feb	Mar
Budget Sales	10	20	30

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

(axis)	Jan	Feb	Mar
Budget Sales	110	100	110

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.

```
@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

(axis)	Jan	Feb	Mar
Budget Sales	110	100	120

Assume that we load the values 10, 20, and 30 into the Budget Sales data block for Jan, Feb, and Mar, as follows:

Table 2-21 Values Loaded in Budget Sales Data Block

(axis)	Jan	Feb	Mar
Budget Sales	10	20	30

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

(axis)	Jan	Feb	Mar
Budget Sales	30	100	40

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.

```
@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

(axis)	Jan	Feb	Mar
Opening Inventory	#MI	#MI	#MI
Ending Inventory	#MI	#MI	#MI

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

(axis)	Jan	Feb	Mar
Opening Inventory	10	10	10
Ending Inventory	10	10	10

The following formula on the Opening Inventory member causes Essbase to use cell calculation mode (the default for formulas containing @PRIOR):

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

"Ending Inventory"="Opening Inventory";
```

The results are as follows:

Table 2-25 Cell Calculation Mode Inventory Results

(axis)	Jan	Feb	Mar
Opening Inventory	10	20	30
Ending Inventory	10	20	30

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:

Table 2-26 Data for Actual and Budget Members

(axis)	Cola	New York Sales
(axis)	**Actual**	**Budget**
Jan	#MISSING	50
Feb	200	#MISSING
Mar	400	450

The following formula is calculated bottom-up.

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

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.

Table 2-27 Bottom-up Calculation Results for Actual and Budget

(axis)	Cola	New York Sales	(Comment)
(axis)	**Actual**	**Budget**	**-**
Jan	#MISSING	#MISSING	(#MISSING*1.10)

Table 2-27 (Cont.) Bottom-up Calculation Results for Actual and Budget

(axis)	Cola	New York Sales	(Comment)
(axis)	**Actual**	**Budget**	**-**
Feb	200	#MISSING	(No calculation is performed)
Mar	400	440	(400*1.10)

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

(axis)	Cola	New York Sales	(Comment)
(axis)	**Actual**	**Budget**	**-**
Jan	#MISSING	#MISSING	(#MISSING*1.10)
Feb	200	220	(200*1.10)
Mar	400	440	(400*1.10)

@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
- @ICHILDREN
- @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:

```
                FY1998   FY1999   FY2000   FY2001   FY2002
                ======   ======   ======   ======   ======
Credit Rate     0.0525   0.0525   0.0525   0.0525   0.0525
Compound Interest    0      105 110.5125 273.8144 288.1897
Deposit              0    2,000        0    3,000        0
```

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:

```
                FY1998 FY1999 FY2000 FY2001  FY2002  FY2003
                ====== ====== ====== ======  ======
```

```
======
Principal Amount  2,000  2,000  2,000  3,000  2,500    -500
Growth Rate      0.0525      0      0      0      0       0
Compound Growth   2,105  2,105  2,105  2,105  2,105   2,105
```

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

```
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 @MEMBER with @CONCATENATE; for example,

```
@MEMBER(@CONCATENATE("2000_", QTR1));
```

Example

The following examples are based on the Sample Basic database:

Example 1 (@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:

```
@CONCATENATE("Item",@NAME(@CURRMBR(Product)))
```

Example 2 (@CONCATENATE)

To concatenate more than two strings, you can nest multiple instances of the @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:

```
@CONCATENATE(@NAME(@CURRMBR(Year)),@CONCATENATE("_",@NAME(@CURRMBR(Measu
res))))
```

See Also

- @MEMBER
- @NAME
- @SUBSTRING

@CORRELATION

Returns the correlation coefficient between two parallel data sets (*XrangeList1* and *XrangeList2*). The correlation coefficient determines the relationship between two data sets.

Syntax

```
@CORRELATION (SKIPNONE | SKIPMISSING | SKIPZERO | SKIPBOTH,
XrangeList1, XrangeList2)
```

Parameters

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:

$$\rho_{X,Y} = \frac{Cov(X,Y)}{\sigma_X * \sigma_Y}$$

so that

$$-1 \leq \rho_{X,Y} \leq 1$$

and

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

σ_x stands for the standard deviation of $X = \{x_i\}_{i=1}^{n}$

σ_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:

```
             Colas     Actual    New York
               Sales       Sales Correl
               =====       ============
Jan              678            #MI
Feb              645            #MI
Mar              675            #MI
Apr              712            #MI
May              756            #MI
Jun              890       0.200368468
```

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.

```
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

```
@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.

```
                            Jan     New York
                          Actual      Budget
                          ======      ======
Sales        Colas          678         640
             Root Beer      551         530
             Cream Soda     663         510
             Fruit Soda     587         620
             Diet Drinks    #MI         #MI
              Product      2479        2300

Prod Count   Product         4           4
```

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").

```
/* 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:

```
Formula                      Current Member   Value
Position = @CURGEN(Year);    Year             1
Position = @CURGEN(Year);    Qtr2             2
Position = @CURGEN(Year);    Oct              3
```

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:

```
Formula                     Current Member   Value
Position = @CURLEV(Year);   Year             2
Position = @CURLEV(Year);   Qtr3             1
Position = @CURLEV(Year);   Aug              0
```

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:

```
          Jan        New York       Actual
              Sales        Average Sales
              =====        =============
100-10          5              5
100-20         10             10
100-30         15             15
  100          30             10
200-10         20             20
200-20         25             25
200-30         30             30
200-40         35             35
  200         110             27.5
  300         #MI            #MI
  400         #MI            #MI
  Diet         35             11.67
Product       140             35
```

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:

```
                    Jan    Feb    Mar    Apr        Nov    Dec
Opening Inventory   100    110    120    130 . . . 200    210
Average Inventory   105    110    115    120 . . ..155    155
```

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.

```
Turnover = @SUMRANGE(Sales,@CURRMBRRANGE(Year, LEV, 0, , 0))/"Average
Inventory"
```

which produces the following result:

```
                    Jan     Feb     Mar     Apr
Average Inventory   110     116.7   122.5   126
Sales                40      44      48      52
Turnover               0.36    0.72    1.08    1.46
```

Example 3

Consider the following formula:

```
@CURRMBRRANGE(Year,LEV,@CURLEV("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 configuration 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 54).
- 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*, seeRange 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:

```
             FY2000    FY2001    FY2002    FY2003
             ======    ======    ======    ======
Asset         9,000         0         0         0
Residual        750         0         0         0
Life              5         0         0         0
Decline Dep   3,600     2,160     1,296       778
```

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)
```

returns New York, Massachusetts, Florida, Connecticut, and New Hampshire (in that order).

```
@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:

```
                    FY1999  FY2000  FY2001  FY2002  FY2003
                    ======  ======  ======  ======
======
Cash                  0.00    0.00 1000.00 1000.00    0.00
Credit Rate           0.00    0.00    0.05    0.05    0.00
Discount Amount        #MI     #MI  863.84  822.70    #MI
```

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
```

```
2012->Feb
2012->Mar
```

@ENUMVALUE

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

Syntax

```
@ENUMVALUE ([mbrName,]enum_string)
```

Parameters

mbrName
Optional. Any valid single member name, or a function that returns a single member. If given as the first argument, @ENUMVALUE checks the text list associated with that member, and returns the numeric value of the character string provided in the second argument.

enum_string
If *mbrName* is given as the first argument, this is a *char_string_literal* of one of the text strings represented in the text list.
If no *mbrName* is given as first argument, this is a 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

Example 1, No Member Name

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");
```

The above example returns 1.

Example 2, with Member Name

The following calculation example performs a conditional test and assigns a value to Profit depending on the test results. A calculation script is used to set the member named SmartText with the text string "RED":

```
SET CREATENONMISSINGBLK ON;
FIX ("100-10", California, Actual, Oct)
SmartText = "RED";
ENDFIX;
```

A member formula on Profit causes the value of Profit to depend on the following conditional test: if the internal numeric value of member SmartText is associated with the text string "RED", the value of Profit will be set to 99.

```
Profit (IF(@EnumValue(SmartText, "RED") == SmartText )
Profit=99;
```

The conditional test returns true, and the value of Profit is set to 99.

@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:

```
                    East      West     South   Central
Variance %          10.7      10.9       3.6       3.6
Index            1.11293   1.11516   1.03666   1.03666
```

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
- @ICHILDREN
- @DESCENDANTS
- @IDESCENDANTS
- @EQUAL
- @MBRPARENT
- @SIBLINGS
- @ISIBLINGS

mbrList
A comma-delimited list of members grouped together using @LISTor 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
      300-20
   Diet
      100-10
         100-10-11
      200-10
      300-10
   Bottle
      200-10
      300-20


@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

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

See Also

@POWER

@FORMATDATE

Returns a formatted date-string.

Syntax

```
@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:

```
                  FY1998 FY1999 FY2000 FY2001 FY2002 FY2003
                  ====== ====== ====== ====== ======
======
Principal Amount   1,000      0  2,000      0      0      0
Growth Amount      1,050  1,120  1,200  1,280  1,380  1,480
Growth Rate         0.05   0.07   0.08   0.08    0.1    0.1
```

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

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

```
FIX(Budget,@IALLANCESTORS("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.

```
                          Jan
                    Actual      Budget
                    Sales       Sales
                    =====       =====
Market  100-10      4860        5200
        100-20      2372        2740.5 *
        100-30      1082        1170
         100        8314        9429   *
        100-20      2372        2610
        200-20      3122        3090
        300-30      2960        2560
         Diet       8454        8673   *
          Product   31538      30954   *
```

See Also

- @ALLANCESTORS
- @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

```
@IANCESTORS (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.

```
@IANCESTORS("New York")
```

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

```
@IANCESTORS(Qtr4)
```

Returns Qtr4, Year (in that order).

```
@IANCESTORS(Sales,-2)
```

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

```
@IANCESTORS("100-10",1)
```

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

See Also

- @ANCESTORS
- @IALLANCESTORS
- @ILANCESTORS
- @LANCESTORS

@ICHILDREN

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

```
@ICHILDREN (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)
```

Returns East, New York, Massachusetts, Florida, Connecticut, and New Hampshire (in that order).

```
@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(Ounce
  s_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
- @RDESCENDANTS
- @SIBLINGS
- @SHIFTSIBLING

@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:

```
                  Profit %
                Cola      Actual
               Jan    Feb    Mar
               ===    ===    ===
California    38.64  37.98  38.37
Oregon        17.50  16.13  16.11
Washington    29.23  30.90  32.00
Utah          23.08  23.08  20.97
Nevada        -3.95  -6.76  -5.33
  West       104     101   102
```

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:

```
                FY1998  FY1999   FY2000  FY2001   FY2002   FY2003
                ======  ======   ======  ======   ======
======
Balance        2000.00 3000.00 -1000.00 3000.00  9000.00 -6000.00
Credit Rate      0.065   0.065    0.065   0.065    0.065    0.065
Borrow Rate     0.1125  0.1125   0.1125  0.1125   0.1125   0.1125
Interest Amount 130.00  195.00  -112.50  195.00   585.00  -675.00
```

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

```
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

```
@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.

`@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:

```
Product
   100
     100-10
     100-20
     100-30
   200
     200-10
     200-20
     200-30
     200-40
   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).
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 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:

```
         FY1998  FY1999  FY2000  FY2001  FY2002    FY2003
         ======  ======  ======  ======  ======
======
Cash    (1,000)    500    600    500   #MISSING   #MISSING
Rate          0      0      0      0   #MISSING   #MISSING
Return        0      0      0      0          0          0
```

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

```
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

```
@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)
```

Returns Florida, Connecticut, and New Hampshire (in that order). Connecticut and New Hampshire appear below Florida in the Sample Basic outline.

```
@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

@ISIANCEST

@ISATTRIBUTE

Returns TRUE if the current member under calculation matches the attribute or varying attribute name specified in *attMbrName*.

Syntax

```
@ISATTRIBUTE (attMbrName)
```

Parameters

attMbrName
Single attribute member name or member combination.

Notes

- This function provides the same functionality as @ISMBR (@ATTRIBUTE(*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;
  ENDIF
);
```

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)
```

Returns TRUE for Year, Qtr1.

```
@ISIPARENT(Margin)
```

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

```
@ISMBRWITHATTR (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 `@ISMBR(@WITHATTR())`, but is faster.
- This function is a superset of the @ISATTRIBUTE function. The following two formulas return the same member set:

  ```
  @ISATTRIBUTE(Bottle)
  @ISMBRWITHATTR("Pkg Type","==",Bottle)
  ```

  However, the following formula can be performed only with @ISMBRWITHATTR (not with @ISATTRIBUTE) because you specify a condition:

  ```
  @ISMBRWITHATTR(Ounces,">","16")
  ```

- If you specify a date attribute with the @ISMBRWITHATTR function, you must use @TODATE in the *string* parameter to convert the date string to a number.
- The following operators are supported:

Table 2-29 Supported Operators

Operator	Description
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
= =	Equal to
<> or !=	Not equal to
IN	In

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

```
@ISRANGENONEMPTY(ZEROASDATA, Sales->Cola)
```

Because the intersection of Cola and Sales contains non-#MISSING values, the condition returns TRUE.

Example 2

```
//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

```
@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
- @ISSAMEGEN
- @LEV

@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 (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

@LANCESTORS

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

```
@LANCESTORS ((memberSetFunction) [,genLevNum])
```

Parameters

memberSetFunction
A member set function that returns a list of members.
How the @LANCESTORS 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
- @IANCESTORS
- @ILANCESTORS

@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),"2
  00-40"))
  ...
  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

Function	Level Returned
@LEV(Margin)	1
@LEV("New York")	0

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.
- 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.

See Also

- @BETWEEN
- @EQUAL
- @EXPAND
- @MBRCOMPARE

- @MBRPARENT
- @NOTEQUAL

@LIST

Creates and distinguishes lists that are processed by functions that require list arguments. Can be used to create *expLists*, member lists, or *rangeLists*. This function treats a collection of parameters as one entity.

Syntax

```
@LIST (argument1, argument2, ..., argumentN)
```

Parameters

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.

Notes

@LIST does not check for or eliminate duplicates.

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 @CORRELATION function to determine the sales relationship between a product's two peak periods (Jan-Mar and Apr-May):

```
FIX(Sales)
"Sales Correl" = @CORRELATION(SKIPNONE,
  @LIST(Jan,Feb,Mar),@LIST(Apr,May,Jun));
ENDFIX
```

This example produces the following report:

```
        Colas     Actual    New York
                  Sales
                  =====
Jan                678
Feb                645
Mar                675
Apr                712
May                756
Jun                890

Sales Correl  0.200368468
```

@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:

```
                                Cola East
                Jan       Feb       Mar           Nov      Dec
Sales           100       110       120 . . .       0      210
LN_Sales    4.65052   4.70048   4.78749 . . . #MISSING  5.34710
```

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:

```
LOG2_Sales = @LOG(Sales,2);
```

This example produces the following result:

```
                                  Cola East
                Jan         Feb       Mar             Nov       Dec
Sales           100    #MISSING       120 . . .         0       210
LOG2_Sales  6.64386    #MISSING   6.90689 . . .  #MISSING  7.71425
```

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:

```
LOG10_Sales = @LOG10(Sales);
```

This example produces the following result:

```
                                Product  Actual
                     East        West     South   Central
Sales                87398      132931     50846    129680
LOG10_Sales        4.94150    5.12363   4.70626   5.11287
```

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:

```
           Colas   New York   Actual
          Jan     Feb     Mar     Qtr1
          ===     ===     ===     ====
Sales     678     645     675     678
```

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:

```
           Colas    New York    Actual
          Jan      Feb      Mar      Qtr1
          ===      ===      ===      ====
Sales     678      645      675      678
```

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:

```
                 Jan       Feb       Mar  Qtr1_Max
            ========  ========  ========  ========
COGS        #MISSING      1500      2300      2300
OtherInc_Exp    -500      -350         0      -350
```

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:

```
                Jan       Feb       Mar  Qtr1_Max
           ========  ========  ========
========
COGS       #MISSING      1500      2300      2300
OtherInc_Exp    -500      -350         0         0
```

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:

```
                 Jan       Feb       Mar  Qtr1_Max
            ========  ========  ========  ========
OtherInc_Exp    -500  #MISSING      -250      -250
COGS               0      1500      2300      -250
```

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:

```
                 Jan       Feb       Mar  Qtr1_Max
            ========  ========  ========  ========
OtherInc_Exp    -500  #MISSING      -250  #MISSING
COGS               0      1500      2300  #MISSING
```

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
- @MBRCOMPARE
- @NOTEQUAL

@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:

  ```
  amount * (@CURRMBR()-> basisMbr / @SUM(allocationRange ->
  basisMbr))
  ```

- *spread*: Spreads *amount* across *allocationRange*:

  ```
  amount * (1/@COUNT(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*:

  ```
  amount * (@CURRMBR()->basisMbr * .01).
  ```

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

  ```
  amount + @CURRMBR()-> basisMbr
  ```

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

  ```
  amount - @CURRMBR()->basisMbr
  ```

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

  ```
  amount * @CURRMBR()->basisMbr
  ```

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

  ```
  amount/@CURRMBR()->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.
- 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:

```
FIX("Total Expenses")
Budget = @MDALLOCATE(Budget->"Total Expenses"->East,2,
   @CHILDREN(East),@CHILDREN("Total Expenses"),Actual,,share);
ENDFIX
```

This example produces the following report:

```
                                  Jan       Colas
                      Marketing   Payroll   Misc   Total Expenses
                      =========   =======   ====   ==============
Actual   New York          94          51      0              145
         Massachusetts     23          31      1               55
         Florida           53          54      0              107
         Connecticut       40          31      0               71
         New Hampshire     27          53      2               82
           East           237         220      3              460
Budget   New York        102.174      55.435    0              #MI
         Massachusetts     25        33.696    1.087          #MI
         Florida         57.609      58.696    0              #MI
         Connecticut     43.478      33.696    0              #MI
         New Hampshire   29.348      57.609    2.173          #MI
           East           #MI          #MI      #MI            500
```

See Also

@ALLOCATE

@MDANCESTVAL

Returns ancestor-level data from multiple dimensions based on the current member being calculated.

Syntax

```
@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:

```
                      Sales   Marketing
                      =====
=========
New York     100-10     300     N/A
             100-20     200     N/A
                100     500     N/A
Boston       100-10     100     N/A
             100-20     400     N/A
                100     500     N/A
East         100-10     400     N/A
             100-20     600     N/A
                100    1000     200
```

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:

```
                     Sales   Marketing
                     =====   =========
New York     100-10    300     60
             100-20    200     40
                100    500     100
Boston       100-10    100     20
             100-20    400     80
                100    500     100
East         100-10    400     80
             100-20    600     120
                100   1000     200
```

The Marketing expenses can then be reconsolidated across Products and Markets.

See Also

- @ANCESTVAL
- @MDPARENTVAL
- @SANCESTVAL

@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:

```
                      Sales   Marketing
                      =====
=========
New York   100-10      300      N/A
           100-20      200      N/A
              100      500      N/A
Boston     100-10      100      N/A
           100-20      400      N/A
           100         500      N/A
East       100-10      400      N/A
           100-20      600      N/A
              100     1000      200
```

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 / @MDPARENTVAL(2, Market, Product, Sales)) *
@MDPARENTVAL(2, Market, Product, Marketing);
```

which produces the following result:

```
                    Sales   Marketing
                    =====
=========
New York   100-10     300       60
           100-20     200       40
              100     500       N/A
Boston     100-10     100       20
           100-20     400       80
```

```
          100        500       N/A
East      100-10     400       N/A
          100-20     600       N/A
             100    1000       N/A
```

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:

```
                                  Jan   Feb   Mar
                                  ===   ===   ===
Prior Year   Opening Inventory    110   120   130  . .
Budget       Ending Inventory     N/A   N/A   N/A . .
```

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:

```
                                  Jan      Feb     Mar
                                  ===      ===
===
Prior Year  Opening Inventory    110      120     130 . .
Budget      Ending Inventory     120      130     140 . .
```

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:

```
                        Jan     New York
                     Actual       Budget
                     ======       ======
Sales    Colas         678          640
         Root Beer     551          530
         Cream Soda    663          510
         Fruit Soda    587          620
         Diet Drinks   #MI          #MI
          Product     2479         2300

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.

```
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

```
@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.

```
QTD=@MEMBER(@CONCATENATE("2000_", QTR1));
```

See Also

- @CONCATENATE
- @SUBSTRING

@MEMBERAT

Returns the specified member in a list of members.

Syntax

```
@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.

`@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:

```
                  Product     Year     Budget
                            Marketing
                            =========
New York                      6039
Massachusetts                 1276
Florida                       2530

California                    7260
Oregon                        2090
Washington                    2772
Utah                          1837
Nevada                        4521
```

The values prior to running the calculation script were:

```
New York        5490
Massachusetts   1160
Florida         2300

California      6600
Oregon          1900
Washington      2520
Utah            1670
Nevada          4110
```

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:

```
            Colas   New York   Actual
          Jan     Feb     Mar     Qtr1
          ===     ===     ===     ====
Sales     678     645     675     645
```

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:

```
            Colas   New York   Actual
          Jan     Feb     Mar     Qtr1
          ===     ===     ===     ====
Sales     678     645     675     645
```

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:

```
                Jan       Feb       Mar  Qtr1_Min
           ========  ========  ========  ========
COGS       #MISSING      1500      2300      1500
OtherInc_Exp    -500      -350         0      -500
```

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:

```
                Jan       Feb       Mar  Qtr1_Min
           ========  ========  ========  ========
COGS       #MISSING      1500      2300  #MISSING
OtherInc_Exp    -500      -350         0      -500
```

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 = @MINSRANGE(SKIPBOTH, OtherInc_Exp, Jan:Mar);
```

This example ignores the 0 value for Mar and produces the following results:

```
                 Jan       Feb       Mar  Qtr1_Min
            ========  ========  ========  ========
COGS        #MISSING      1500      2300       350
OtherInc_Exp     500       350         0       350
```

Example 2

```
Qtr1_Min = @MINS(SKIPNONE, OtherInc_Exp, Jan:Mar);
```

This example does not ignore the 0 value in the calculation. This example produces the following results:

```
                 Jan       Feb       Mar  Qtr1_Min
            ========  ========  ========  ========
COGS        #MISSING      1500      2300         0
OtherInc_Exp     500       350         0         0
```

See Also

- @MINS
- @MINRANGE
- @MAXSRANGE

@MOD

Calculates the modulus of a division operation.

Syntax

```
@MOD (mbrName1, mbrName2)
```

Parameters

mbrName1 and mbrName2
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:

```
Factor = @MOD("Margin %", "Profit %");
```

This example produces the following report:

```
             Market      Product      Scenario
            Margin %     Profit %      Factor
            ========     ========      ======
Jan          55.10        25.44         4.22
Feb          55.39        26.03         3.34
Mar          55.27        25.87         3.53
```

@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:

```
FIX (Central)
"Mode" = @MODE(SKIPMISSING,
   @RANGE(Sales,@CHILDREN(Central)));
ENDFIX
```

This example produces the following report:

```
                    Colas   Actual    Jan
                          Units Sold
                          ==========
```

```
Units Sold   Illinois            3
             Ohio                2
             Wisconsin           3
             Missouri          #MI
             Iowa                0
             Colorado            6
               Central          14

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.

```
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:

```
Trailing Average       Centered Average
   1   2   3              1   2   3
           2                  2
```

- 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:

```
          Colas   New York   Actual
              Sales     Mov Avg
              =====     =======
Jan            678        678
```

```
Feb             645         645
Mar             675         666
Apr             712         677.3
May             756         714.3
Jun             890         786
```

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
- @MOVMED
- @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:

```
Trailing Maximum      Centered Maximum
  1   2   3             1   2   3
          3                 3
```

- 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).

This example produces the following report:

```
            Root Beer    New York    Actual
                 Sales        Mov Max
                 =====        =======
Jan                551           551
Feb                641           641
Mar                586           641
Apr                630           641
May                612           630
Jun                747           747
```

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

- @MOVAVG
- @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:

```
Trailing Median        Centered Median
  1   2   3              1   2   3
          2                  2
```

- 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.

- When you use @MOVMED in a calculation script, use it within a FIX statement. Although FIX is not required, using it may improve calculation performance.
- When you use @MOVMED 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 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:

```
             Colas      New York     Actual
                 Sales        Mov Med
                 =====        =======
Jan               678           678
Feb               645           645
Mar               675           675
Apr               712           675
May               756           712
Jun               890           756
```

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
- @MOVSUMX
- @TREND

@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:

```
Trailing Minimum       Centered Minimum
    1   2   3              1   2   3
            1                  1
```

- 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:

```
            Colas     New York    Actual
                Sales       Mov Min
                =====       =======
Jan              678          678
Feb              645          645
Mar              675          645
Apr              712          645
May              756          675
Jun              890          712
```

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
- @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:

```
Trailing Sum          Centered Sum
 1   2   3             1   2   3
         6                 6
```

- 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);
```

In this example, @MOVSUM smoothes sales data for the first six months of the year (Jan through Jun). The results of @MOVSUM can be used with @TRENDto forecast average sales data for a holiday season (for example, October through December).

This example produces the following report:

```
          Colas   New York   Actual
              Sales     Mov Sum
              =====     =======
Jan            678        678
Feb            645        645
Mar            675       1998
Apr            712       2032
May            756       2143
Jun            890       2358
```

See Also

- @MOVAVG
- @MOVMED
- @MOVMAX
- @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:

```
Trailing Sum          Centered Sum
1   2   3             1   2   3
        6                 6
```

- 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
- @MOVMIN
- @MOVSUM
- @TREND

@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.

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

MonthName	ProductId	Price
Jan	100-10	1.90
Feb	100-10	1.95
Mar	100-10	1.98
Jan	100-20	1.95
Feb	100-20	2.00
Mar	100-20	2.05

```
Price = @J_GetPrice(@NAME(@CURRMBR(Product)),@NAME(@CURRMBR(Year)));
```

The following report illustrates the above example:

```
                Price   Actual  Market
                 Jan     Feb     Mar
                 ===     ===     ===
```

```
      100-10   1.90    1.95    1.98
      100-20   1.95    2.00    2.05
```

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:

```
             Actual      Year      West
             Profit        Profit Per Ounce
            ========      ================
Cola         4593           382.75
```

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:

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 *n*th cell value from *mbrName*, in the sequence *XrangeList* , retaining all other members identical to the current member. @NEXT cannot operate outside the given range.

Syntax

```
@NEXT (mbrName [, n, XrangeList])
```

Parameters

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

n
Optional signed integer. If you do not specify *n*, then the default is set to 1, which provides the next member in the range. Using a negative value for *n* has the same effect as using the matching positive value in @PRIOR.

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 *n* is not specified, the default is 1, which provides the next member in the range. Since *XrangeList* is not specified, the level 0 members from the dimension tagged as Time are used (Jan,Feb,Mar, ...).

```
"Next Cash" = @NEXT(Cash);
```

This example produces the following report:

```
            Jan   Feb   Mar   Apr   May   Jun
            ===   ===   ===   ===   ===   ===
Cash        100    90   120   110   150   100
Next Cash    90   120   110   150   100   #MI
```

The following example assumes a Year dimension is added to Sample Basic.

```
"Next Sales" = @NEXT(Sales, 1, @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
2012->Jan
```

```
2012->Feb
2012->Mar
```

See Also

- @PRIOR
- @SHIFT
- @SHIFTMINUS
- @SHIFTMINUS

@NEXTS

Returns the *n*th cell value from *mbrName*, in the sequence *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.

Syntax

```
@NEXTS (SKIPNONE | SKIPMISSING | SKIPZERO | 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 0 values in the sequence.

SKIPBOTH
Ignores all `#MISSING` and 0 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 *n* has the same effect as using the matching positive value in @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:

```
             Jan     Feb    Mar    Apr    May    Jun
             ===     ===    ===    ===    ===    ===
Cash         1100    #MI    1000   1300     0    1400
Next Cash    1000    1000   1300   1400   1400    #MI
```

The following example assumes a Year dimension is added to Sample Basic.

```
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 wrap the @NEXTSIBLING function call within a @MEMBER function call.

You must also wrap this function within the @MEMBER function if you are calling it inside a member combination specified using the cross-dimensional operator (->). For example, this is correct usage: `@MEMBER(@NEXTSIBLING("FY19"))->"A1"`.

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

@NONEMPTYTUPLE

Use before a formula to force queries to use only data dependent cells when executing the formula. This optimizes query times by identifying the required intersections for calculation, making the query time proportional to input data size.

This function can be used as a formula directive. Using it before the formula specification is recommended when the formula is long and contains many cross-dimensional operators. Such formulas often cause the formula cache to grow large while also being sparse (having a relatively small input data set). Using this directive

causes query execution to occur in bottom-up mode, to resolve dependency analysis quickly in cases where the formula cache is sparse.

Syntax

```
@NONEMPTYTUPLE [(nonempty_member_list)]
```

Parameters

nonempty_member_list
Optional. A list of members from the current dimension (the dimension in which this formula applies).
The formula will execute in bottom-up mode if any of the members specified in *nonempty_member_list* are empty. If *nonempty_member_list* is not specified, the formula will execute in bottom-up mode if any dependent members of the current formula are empty. For most use cases, you do not need to specify *nonempty_member_list*; simply place @NONEMPTYTUPLE before the formula syntax to cause bottom-up formula execution.

Example

The following outline formula example is based on the Compensation Analytics sample cube, for which the application workbook is available in the HR Analysis directory of the Applications gallery.

```
@NONEMPTYTUPLE("Headcount under Target")
IF("Headcount under Target"!=#missing)
"Market Movement"*"Size %"->"Actual"->"Sep17"->"No JG"->"unassigned_OU"-
>"No Job Code"->"No EE"->"No_MktComp"*"BASE"*"Competitive Incr %"-
>"Actual"->"Sep17"->"No Region"->"No JG"->"unassigned_OU"->"No Job
Code"->"No EE"->"No_MktComp";
ENDIF;
```

The following example is for a calculation script use case:

```
"Headcount under Target"(
@NONEMPTYTUPLE("COMPARATIO")
IF("COMPARATIO"!=#missing AND "COMPARATIO"<="Competitive Target"->"No
JG"->"No Region"->"unassigned_OU"->"No EE"->"No Job Code"->"No_MktComp")
1;
ENDIF;)
```

See Also

NONEMPTYTUPLE in MDX Optimization Properties

IGNORECONSTANTS application configuration setting

@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:

```
Value = @NPV(Cash, Rate, 0, FY1990:FY1994, FY1995:FY2000);
```

This example produces the following report:

```
       FY1990  FY1991  FY1992  FY1993  FY1994  FY1995
       ======  ======  ======  ======  ======  ======
Cash   (1,000)   500     600     500  #MISSING  #MISSING
Rate       0     0       0       0   #MISSING  #MISSING
Value    296   296     296     296      296     296
```

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

```
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.

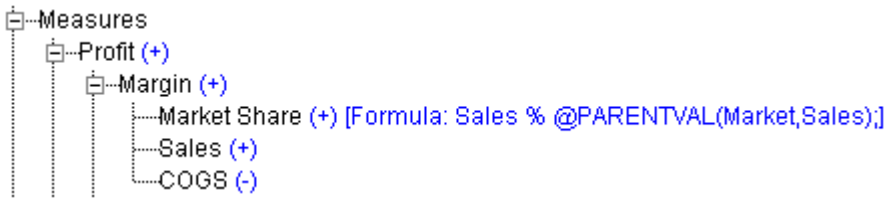
```
"Market Share" = Sales % @PARENTVAL(Market,Sales);
```

This example produces the following report:

```
                Cola     Actual    Jan
                Sales     Market Share
                =====     ============
New York         678         37.42
Massachusetts    494         27.26
Florida          210         11.59
Connecticut      310         17.11
New Hampshire    120          6.62
  East          1812         37.29

Market          4860        100
```

Adding the "Market Share" member and formula to the outline would produce the same result as above.



See Also

- @ANCESTVAL
- @MDPARENTVAL
- @SPARENTVAL

@POWER

Returns the value of the specified member or expression raised to *power*.

Syntax

```
@POWER (expression, power)
```

Parameters

expression
Single member specification, variable name, function, or other numeric expression.

power
Single member specification, variable name, function, or other numeric expression.

Notes

- If *expression* is negative, and if *power* is not an integer, Essbase returns `#MISSING`.
- If the value calculated by @POWER is an infinite number, Essbase returns `#MISSING`.

Example

Table 2-32 @POWER Results

Usage	Return Value
`@POWER(14,3)`	`2744`
`@POWER(2,8)`	`256`

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 wrap the @PREVSIBLING function call within a @MEMBER function call.

You must also wrap this function within the @MEMBER function if you are calling it inside a member combination specified using the cross-dimensional operator (->). For example, this is correct usage: `@MEMBER(@PREVSIBLING("FY19"))->"A1"`.

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 *n*th 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:

```
                Jan     Feb     Mar     Apr    May    Jun
                ===     ===     ===     ===    ===
===
Inventory      1100    1200    1000    1300   1300   1400
Prev Inventory  #MI    1100    1200    1000   1300   1300
```

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
```

```
2012->Jan
2012->Feb
2012->Mar
```

See Also

- @NEXT
- @SHIFT
- @SHIFTMINUS
- @SHIFTPLUS

@PRIORS

Returns the *n*th previous cell member from *mbrName*, in the sequence *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(SKIPNONE | SKIPMISSING | SKIPZERO | SKIPBOTH mbrName[,n,
rangeList])
```

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 *n* has the same effect as using the matching positive value in the @NEXTS function. 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, 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:

```
                  Jan      Feb      Mar      Apr      May      Jun
                  ===      ===      ===      ===      ===      ===
 Inventory       1100      #MI     1000     1300        0     1400
 Prev Inventory   #MI     1100     1100     1000     1300
1300
```

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
Range 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:

```
                         Product   Market
Scenario
                   Sales    Opening Inventory     Ending Inventory
                   =====    =================     ================
Jan                31538         117405               116434
Feb                32069         116434               115558
Mar                32213         115558               119143
  Qtr1             95820         117405               119143
Apr                32917         119143               125883
May                33674         125883               136145
Jun                35088         136145               143458
  Qtr2            101679         119143               143458
QTD                66591         245026               262028
YTD               162411         117405               136145
```

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:

```
                    Colas
               Sales           COGS
               Actual          Actual

            Qtr3    Qtr4     Qtr3    Qtr4
            ====    ====     ====    ====
California  3401    2767     2070    1701
Oregon       932    1051      382     434
Washington  1426    1203      590     498
Utah        1168    1294      520     575
Nevada       496     440      222     197
  West      1484.6  1351     3784    3405
```

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:

```
             Jan     New York     Actual
               Sales      Prod Count
               =====      ==========
Colas           678          #MI
Root Beer       551          #MI
Cream Soda      663          #MI
Fruit Soda      587          #MI
Diet Drinks     #MI          #MI
  Product      2479           4
```

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 (rankOrderType, SKIPNONE | SKIPMISSING | SKIPZERO | SKIPBOTH,
value, XrangeList)
```

Parameters

rankOrderType
The type of order in which to sort the data set. Options:

- ASCEND Rank values listed in *XrangeList* in ascending order.
- DESCEND Rank values listed in *XrangeList* in descending order. This is the default.

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) or ascending order (1989, 4255, 6556, 9650, 15341). The rank of a value identifies its position in the sorted data set in descending order (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 in descending order [5,4,2,2] and then ranks it: [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 (in descending order) 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]
3. Ranks [3] in the list: in this case, [3] has a rank of 4.

- 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

Example 1

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:

```
                New York   Actual    Jan
                  Sales       Sales Rank
                  =====       ==========
Colas              678            1
Root Beer          551            4
Cream Soda         663            2
Fruit Soda         587            3
Diet Drinks        #MI          #MI
```

Example 2

The following example is based on the Sample Basic database. Assume that the Measures dimension contains an additional member, Expense Rank. Essbase ranks the total expenses values for a set of products in ascending order (the minimum expense is assigned rank 1).

In this example, ASCEND is used to rank the values in ascending order.

```
"Expense Rank" = @RANK(ASCEND,SKIPBOTH,"Total Expenses",@RANGE("Total
Expenses",@LEVMBRS(Product,1)));
```

This example produces the following report.

```
                New York   Actual    Jan
                  Total Expense       Expense Rank
                  =============       ==========
Colas              145                   2
Root Beer          215                   4
Cream Soda         213                   3
Fruit Soda         100                   1
Diet Drinks        #MI                 #MI
```

Example 3

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
     100 (Shared Member)
     200 (Shared Member)
```

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
Apr->Actual
Apr->Budget
May->Actual
```

@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

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).

```
FIX(Sales)
"Non-West"=@SUMRANGE(Sales,@REMOVE(@LEVMBRS(Market,0),
          @LIST(@CHILDREN(West),Texas,"New Mexico")));
ENDFIX
```

This example produces the following report:

```
                    Jan      Colas
Actual
                             Sales
                             =====
```

```
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:

```
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;

    ENDIF
  )
ENDFIX
```

@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:

```
                    Market Product
          Profit_%                  Profit
    Jan     Feb     Mar      Jan      Feb      Mar
```

```
            ===     ===     ===     ===     ===     ===
Scenario   21.37   19.09   18.46    21.4    19.1    18.5
```

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
```

```
                Sales   Marketing
                =====   =========
100-10           300       0
100-20           200       0
100              500       0
200-10           100       0
200-30           400       0
200              900       0
100-10           300       0
200-10           100       0
Diet             400      50
100-20           200       0
200-30           400       0
Caffeine Free    600      40
```

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:

```
                Sales    Marketing
                =====    =========
100-10          300       37.5
100-20          200       13.3
100             500       #MI
200-10          100       12.5
200-30          400       26.7
200             900       #MI
100-10          300       37.5
200-10          100       12.5
Diet            400       50
100-20          200       13.3
200-30          400       26.7
Caffeine Free   600       40
```

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(@DESCENDENT((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^{th} cell value from *mbrName*, in the sequence *XrangeList*, retaining all other members identical to the current member.

The direction of @SHIFT is wholly based on *n*, with positive *n* values producing an effect equivalent to @NEXT and negative values of *n* producing an equivalent effect to @PRIOR.

Syntax

```
@SHIFT (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 a positive value in the @PRIOR function. *n* must be a numeric value, not a reference, such as a member name.

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.

Notes

@SHIFT is provided as a more appropriate, self-documenting name than @NEXT or @PRIOR when the value for *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 *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 *n* parameter. Since the range sequence is not specified for either formula, the level 0 members from the dimension tagged as Time are used.

```
"Prev Asset" = @SHIFT(Asset,-1);
"Next Avl Asset" = @SHIFT(Asset,2);
```

This example produces the following report:

```
                Jan    Feb    Mar    Apr    May    Jun
                ===    ===    ===    ===    ===
===
Asset           100    110    105    120    115    125
Prev Asset      #MI    100    110    105    120    115
Next Avl Asset  105    120    115    125    #MI    #MI
```

The following example assumes a Year dimension is added to Sample Basic.

```
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 - @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 @SHIFTMINUS, which runs the formula in block mode to improve performance.

Note: If you use this function in combination with a function that runs in cell mode, it may necessitate execution in cell mode to resolve dependencies. To determine whether a formula executed in cell mode, check the log for the following informational message: `Formula for member [mbrName] will be executed in [CELL] mode`. To learn which functions use cell mode, see the @CALCMODE topic.

Syntax

```
@SHIFTMINUS (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 @SHIFTMINUS to replace the @NEXT function, use `1` as the value for *n*. If you are using @SHIFTMINUS to replace the @PRIOR function, use `-1` as the value for *n*. Default value is +1.

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().

```
Sales = Loss - @SHIFT(Sales, 1);
```

Here is the formula using @SHIFTMINUS() to improve performance:

```
@SHIFTMINUS (Loss, Sales, 1)
```

The following example assumes a Year dimension is added to Sample Basic.

```
FIX("South", "East")
Sales = @SHIFTMINUS (COGS, Sales, 1, @XRANGE("2018"->"Sep", "2019"-
>"Mar"));
ENDFIX;
```

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

```
2018->Sep
2018->Oct
2018->Nov
2018->Dec
2019->Jan
2019->Feb
2019->Mar
```

See Also

@SHIFTPLUS

@CALCMODE (for an explanation of block calculation and cell calculation modes)

@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.

Note: If you use this function in combination with a function that runs in cell mode, it may necessitate execution in cell mode to resolve dependencies. To determine whether a formula executed in cell mode, check the log for the following informational message: `Formula for member [mbrName] will be executed in [CELL] mode`. To learn which functions use cell mode, see the @CALCMODE topic.

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.

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().

```
Sales = Loss + @SHIFT(Sales, 1);
```

Here is the formula using @SHIFTPLUS() to improve performance:

```
@SHIFTPLUS (Loss, Sales, 1);
```

The following example assumes a Year dimension is added to Sample Basic.

```
FIX("North")
Sales = @SHIFTPLUS (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

@SHIFTMINUS

@CALCMODE (for an explanation of block calculation and cell calculation modes)

@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 wrap the @SHIFTSIBLING function call within a @MEMBER function call.

You must also wrap this function within the @MEMBER function if you are calling it inside a member combination specified using the cross-dimensional operator (->). For example, this is correct usage: `@MEMBER(@SHIFTSIBLING("FY19"))->"A1"`.

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.

See Also

- @MEMBER
- @NEXTSIBLING
- @PREVSIBLING

@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:

```
cost - salvage value / 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 @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).

```
"SLN Dep" = @SLN(Asset,Residual,Life,FY1991:FY1995);
```

This example produces the following report:

```
            FY1991  FY1992  FY1993  FY1994  FY1995  FY1996
            ======  ======  ======  ======  ======  ======
Asset        9,000       0   1,000       0       0    0
Residual    750.00    0.00    0.00    0.00       0    0
Life          5.00     #MI    5.00    0.00    0.00    0
SLN Dep       1650    1650    1850    1850    1850    0
```

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:

```
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
```

```
                Sales    Marketing
                =====    =========
100-10          300         0
100-20          200         0
100             500         0
200-10          100         0
200-30          400         0
200             900         0
100-10          300         0
200-10          100         0
Diet            400        50
100-20          200         0
200-30          400         0
Caffeine Free   600        40
```

The Marketing Expense value is allocated down to each Product code with the following formula:

```
Marketing = (Sales / @SPARENTVAL(Product, Sales)) *
@SPARENTVAL(Product, Marketing);
```

which produces the following result:

```
                Sales    Marketing
                =====    =========
100-10          300         37.5
100-20          200         13.3
100             500         #Missing
200-10          100         12.5
200-30          400         26.7
200             900         #Missing
100-10          300         37.5
200-10          100         12.5
Diet            400         #Missing
100-20          200         13.3
200-30          400         26.7
Caffeine Free   600         #Missing
```

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

$(x_i, y_i), \quad i = 0,1,\ldots,N$

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

$$J(f) = s\int_{x_N}^{x_0} (f''(x))^2 dx + \sum_{i=0}^{N}(f(x_i) - y_i)^2,$$

where y_i are given numbers and $s \geq 0$, where s is the smoothness parameter, and

4) satisfies the boundary condition:

$$S''(x_0) = 0, \; S''(x_N) = 0$$

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:

$$S(x) = S_i(x) = z_i(1-t) + z_{i+1}t - \frac{h_i^2}{6}t(1-t)[(2-t)n_i + (1+t)n_{i+1}]. \quad (*)$$

where

$$h_i = x_{i+1} - x_i, \quad t = \frac{x - x_i}{h_i}, \quad (**)$$

and numbers z_i and n_i, $i = 0,1,\ldots,N$ are a solution of a linear algebraic system.

The numbers n_i are solutions to the system:

$$a_0 n_0 + b_0 n_1 + c_0 n_2 = g_0,$$

$$b_0 n_0 + a_1 n_1 + b_1 n_2 + c_1 n_3 = g_1,$$

$$c_{i-2} n_{i-2} + b_{i-1} n_{i-1} + a_i n_i + b_i n_{i+1} + c_i n_{i+2} = g_i, \quad i = 2,3,\ldots,N-2,$$

$$c_{N-3} n_{N-3} + b_{N-2} n_{N-2} + a_{N-1} n_{N-1} + b_{N-1} n_N = g_{N-1},$$

$$c_{N-2} n_{N-2} + b_{N-1} n_{N-1} + a_N n_N = g_N,$$

where

$$a_i = \frac{1}{3}(h_{i-1} + h_i) + \frac{1}{h_{i-1}^2} s + \left(\frac{1}{h_{i-1}} + \frac{1}{h_i}\right)^2 s + \frac{1}{h_i^2} s,$$

$$i = 1,2,\ldots,N-1,$$

$$b_i = \frac{1}{6} h_i - \frac{s}{h_i}\left[\left(\frac{1}{h_{i-1}} + \frac{1}{h_i}\right) + \left(\frac{1}{h_i} + \frac{1}{h_{i+1}}\right)\right],$$

$$i = 1,2,\ldots,N-2,$$

$$c_i = \frac{s}{h_i h_{i+1}}, \quad i = 1,2,\ldots,N-3$$

$$g_i = \frac{y_{i+1} - y_i}{h_i} - \frac{y_i - y_{i-1}}{h_{i-1}}, \quad i = 1,2,\ldots,N-1$$

The end conditions are:

$$a_0 = 1, \quad b_0 = 0, \quad c_0 = 0, \quad g_0 = 0,$$

$$a_N = 1, \quad b_{N-1} = 0, \quad c_{N-2} = 0, \quad g_N = 0.$$

When numbers n_i are found, the magnitudes z_i are easily determined by formulas

$$z_i = y_i - s D_i, \quad i = 0,1,2,\ldots,N,$$

where

$$D_0 = \frac{1}{h_0}(n_1 - n_0), \quad D_N = -\frac{1}{h_{N-1}}(n_N - n_{N-1}),$$

$$D_i = \frac{1}{h_i}(n_{i+1} - n_i) - \frac{1}{h_{i-1}}(n_i - n_{i-1}), \quad i = 1,2,\ldots,N-1.$$

And now given any x, use (*) and (**) from above to calculate $S(x)$.

Example

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:

```
             Colas     Actual    New York
               Sales       Sales Spline
               =====       ============
Jan              645         632.8941564
Feb              675         675.8247101
Mar              712         724.7394598
Apr              756         784.2860765
May              890         852.4398456
Jun              912         919.8157517
```

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:

$$\sqrt{\frac{n\sum x^2 - \left(\sum x\right)^2}{n(n-1)}}$$

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:

```
                              Jan      New York
                           Actual        Budget
                           ======        ======
Sales          Colas          678           640
               Root Beer      551           530
               Cream Soda     663           510
               Fruit Soda     587           620
               Diet Drinks    #MI           #MI
                 Product     2479          2300
```

```
Std Deviation  Product          60.73       64.55
```

See Also

- @RANGE
- @STDEVP
- @STDEVRANGE

@STDEVP

Calculates the standard deviation of the specified data set (*expList*).

This function assumes that *expList* represents the entire population. If you want *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 *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.

Notes

@STDEVP calculates the standard deviation of the specified data set (*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:

$$\sqrt{\frac{n\sum x^2 - \left(\sum x\right)^2}{n^2}}$$

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:

```
                              Jan    New York
                            Actual     Budget
                            ======
======
Sales           Colas         678        640
                Root Beer     551        530
                Cream Soda    663        510
                Fruit Soda    587        620
                Diet Drinks   #MI        #MI
                  Product    2479       2300

Std Deviation   Product        52.59      55.90
```

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.

```
FIX (Product)
"Std Deviation" = @STDEVRANGE(SKIPBOTH,Sales,@CHILDREN(Product));
ENDFIX
```

This example produces the following report:

```
                              Jan      New York
                           Actual        Budget
                           ======        ======
Sales          Colas          678           640
               Root Beer      551           530
               Cream Soda     663           510
               Fruit Soda     587           620
               Diet Drinks    #MI           #MI
                 Product     2479          2300

Std Deviation  Product        60.73         64.55
```

See Also

- @STDEV

- @STDEVP

@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

Function Statement	Result
@SUBSTRING ("100-10",1)	"00-10"
@SUBSTRING ("200-21",0,2)	"20"
@SUBSTRING (@Name(@Parent(Jan)),3) (The parent of Jan is Qtr1.)	"1"

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:

```
                 Product    Qtr1    Actual
                  Sales    Total Expenses
                  =====    ==============
New York           7705        2068
Massachusetts      3660         892
Florida            4132        1313
Connecticut        3472        1087
New Hampshire      1652         801
  East            20621        6161
California        11056        2742
Oregon             5058        1587
Washington         4835        1621
Utah               4209        1544
Nevada             6516        2193
  West            31674       15848
```

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:


           Actual   New York   Colas
                     Sales
                     =====
Jan                   678
Feb                   645
Mar                   675
Apr                   712
May                   756
Jun                   890
Partial Year         4356
```

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:

```
          FY1999    FY2000    FY2001    FY2002    FY2003
          ======    ======    ======    ======    ======
Asset     9,000          0       1,000       0          0
Residual 750.00       0.00        0.00    0.00          0
Life        5.00  #MISSING        3.00    0.00       0.00
SYD Dep     2750      2200        2150    1433        717
```

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

```
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

```
@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:

```
                                  Actual    Jan   Massachusetts
                                         Marketing
Intro Date_12-10-1996      200-30            9
                           200-40            9
Intro Date_10-01-1996      400-10            9
                           400-20            9
Intro Date_07-26-1996      200-20            9
Intro Date_06-26-1996      300-10            9
                           300-20            9
                           300-30            9
Intro Date_04-01-1996      100-20           10
                           100-30           10
Intro Date_03-25-1996      100-10           10
Intro Date_09-27-1995      200-10           10
```

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`

8. `15-01-06`
9. `15 January 06`
10. `15 Jan 06`
11. `January 15 06`
12. `Jan 15 06`
13. `01-15-06`
14. `06/01/15`
15. `060115`
16. `15 January 2006`
17. `15 Jan 2006`
18. `2006-01-15`
19. `2006/01/15`
20. `Sunday, January 15, 2006`
21. `1/8/06` (m/d/yy)

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:

- *LR*[,*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.
- *SES*[,*c*]: single exponential smoothing with parameter *c* (default *c*=0.2). This method uses its own weight system, using the single parameter *c*. Increasing this parameter gives more weight to early observations than to later ones.
- *DES*[[,*c1*],*c2*]: double exponential smoothing (Holt's method) with optional parameters *c1*, *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.

- *TES*[[[[,*T*],*c1*],*c2*],*c3*]: triple exponential smoothing (Holt-Winters method) with optional parameters *c1*, *c2*, *c3*, *T* (default *c1*=0.2, *c2*=0.05, *c3*=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*:

- *XLOG*[,*c*]: logarithmic change with shift *c* (x' = log(x+c)) (default c=1
- *XEXP*[,*c*]: exponential change with shift *c* (x' = exp(x+c)) (default c=0).
- *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*:

- *YLOG*[,*c*]: logarithmic change with shift *c* (y' = log(y+c)) (default c=1)
- *YEXP*[,*c*]: exponential change with shift *c* (y' = exp(y+c)) (default c=0).
- *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.
- 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

Ylist $y_1, y_2, \ldots, y_K$

Xlist $x_1, x_2, \ldots, x_K$

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, $w_i = 1$)

$$S = \sum_{i=1}^{K} (w_i)^2 \qquad S_x = \sum_{i=1}^{K} x_i (w_i)^2 \qquad S_y = \sum_{i=1}^{K} y_i (w_i)^2$$

$$S_{xx} = \sum_{i=1}^{K} (x_i)^2 (w_i)^2 \qquad S_{xy} = \sum_{i=1}^{K} (x_i y_i)(w_i)^2$$

$$\Delta = SS_{xx} - (S_x)^2$$

$$a = \frac{S_{xx} S_y - S_x S_{xy}}{\Delta}$$

$$b = \frac{SS_{xy} - S_x S_y}{\Delta}$$

the equation of the line is:

$$line = Y_{LR}(x) = a + bx$$

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 Δ. 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)

$\underbrace{y_1 \quad y_2 \quad y_3} \quad y_4 \quad y_5 \quad y_6$

$* \quad \underset{=\tilde{y}_2}{\frac{y_1+y_2+y_3}{3}} \quad \underset{=\tilde{y}_3}{\frac{y_2+y_3+y_4}{3}} \quad \underset{=\tilde{y}_4}{\frac{y_3+y_4+y_5}{3}} \quad \underset{=\tilde{y}_5}{\frac{y_4+y_5+y_6}{3}} \quad *$

centered moving average

Ylist $y_1, y_2, \ldots, y_K$

Xlist $x_1, x_2, \ldots, x_K$

weightList $w_1, w_2, \ldots, w_K$

@TREND(*Ylist*,,,,,, *LR*, *t*)

Linear regression with seasonal adjustment example:

There are 6 data points and a seasonal adjustment parameter, *t*=3

Input data:

$x_1 = 1 \quad x_2 = 2 \quad x_3 = 3 \quad x_4 = 4 \quad x_5 = 5 \quad x_6 = 6$

$y_1 \quad y_2 \quad y_3 \quad y_4 \quad y_5 \quad y_6$

In linear regressions with seasonal adjustments, the intervals between x values must be the same. Δ is equal to that interval. 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)

$\underbrace{y_1 \quad y_2 \quad y_3} \quad y_4 \quad y_5 \quad y_6$

$$* \quad \frac{y_1 + y_2 + y_3}{3} \quad \frac{y_2 + y_3 + y_4}{3} \quad \frac{y_3 + y_4 + y_5}{3} \quad \frac{y_4 + y_5 + y_6}{3} \quad *$$

$$= \tilde{y}_2 \quad = \tilde{y}_3 \quad = \tilde{y}_4 \quad = \tilde{y}_5 \leftarrow$$

centered moving average

Step 2, Subtract $\tilde{y}$'s from y's:

$$\begin{array}{llll} y_2 & y_3 & y_4 & y_5 \\ -\ \tilde{y}_2 & \tilde{y}_3 & \tilde{y}_4 & \tilde{y}_5 \\ \hline \hat{y}_2 & \hat{y}_3 & \hat{y}_4 & \hat{y}_5 \end{array} \leftarrow \text{difference}$$

Step 3, Arrange $\hat{y}$'s into $n(n = 3)$ columns to derive P's and average values along columns:

$$\begin{array}{lll} * & \hat{y}_2 & \hat{y}_3 \\ \hat{y}_4 & \hat{y}_5 & * \\ \hline \frac{\hat{y}_4}{1} & \frac{\hat{y}_2 + \hat{y}_5}{2} & \frac{\hat{y}_3}{1} \end{array}$$

$$= P_0 \quad = P_1 \quad = P_2 \leftarrow$$

adjustment list

Step 4, Subtract P's from original *Ylist*:

y_1	y_2	y_3	y_4	y_5	y_6
P_0	P_1	P_2	P_0	P_1	P_2
y'_1	y'_2	y'_3	y'_4	y'_5	y'_6

Step 5, Linear Regression (LR) with

$x_1 = 1$	$x_2 = 2$	$x_3 = 3$	$x_4 = 4$	$x_5 = 5$	$x_6 = 6$
y'_1	y'_2	y'_3	y'_4	y'_5	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:

$$x: \quad Y_{forecast} = b * x + a + P_i, \quad where \quad P_i: \quad i = \frac{(x - x_1) \bmod t}{\Delta}$$

$$= \frac{(x-1) \bmod 3}{1}$$

$$= (x-1) \bmod 3$$

Algorithm for Single Exponential Smoothing (SES)

Ylist $y_1, y_2, \ldots, y_K$

Xlist $x_1, x_2, \ldots, x_K$

$c = .2$ default, or else c is input into the trend

find $S_1 S_2, \ldots, S_K$:

$$S_1 = y_1$$

$$S_{i+1} = a_i * S_i + (1 - a_i) y_i \quad \text{for } i = 1, \ldots, K-1$$

$$\text{then } Y_{forecast}(x) = a * S_K + (1 - a) * y_K$$

where $a_i = (1-c)^{x_{i+1} - x_i}$

$a = (1-c)^{x - x_K}$

Note: When *Xlist* is missing, $x_{i+1} - x_i = 1$ and the correspondent coefficients

$a_i = (1 - c)$ *for* $i = 1, \ldots, K-1$

Algorithm for Double Exponential Smoothing (DES)

Ylist $y_1, y_2, \ldots, y_K$

Xlist $x_1, x_2, \ldots, x_K$

$c_1 = .2, \quad c_2 = .3$ default, or else they are input into the trend

find $S_1 S_2, \ldots, S_K$
$b_1 b_2, \ldots, b_K$

$$S_1 = y_1$$
$$b_1 = \frac{(y_2 - y_1)}{(x_2 - x_1)}$$

$$S_{i+1} = a_i * (S_i + b_i (x_{i+1} - x_i)) + (1 - a_i) * (y_{i+1})$$

$$b_{i+1} = d_i * b_i + (1 - d_i) * \left[\frac{(S_{i+1} - S_i)}{x_{i+1} - x_i} \right]$$

where $a_i = (1 - c_1)^{x_{i+1} - x_i}$
$d_i = (1 - c_2)^{x_{i+1} - x_i}$

then $Y_{forecast}(x) = S_K + (x - x_K)\, b_K$

Note: When *Xlist* is missing, $x_{i+1} - x_i = 1$ and the correspondent coefficients

$a_i = (1 - c_1) \quad for\ i = 1, \ldots, K - 1$

$b_i = (1 - c_2)$

Algorithm for Triple Exponential Smoothing (TES)

Ylist $y_1, y_2, \ldots, y_K$

Xlist $x_1, x_2, \ldots, x_K$

TES with period T (if T is not given, it is assumed to be $T = 1$)

$x_1, x_2, \ldots, x_K$, $y_1, y_2 \ldots, y_K$ are input to TES, x is forecast value.

$a_i = (1-c)^{x_{i+1}-x_i}$ $d_i = (1-d)^{x_{i+1}-x_i}$ $e_i = (1-e)^{x_{i+1}-x_i}$

Note: When *Xlist* is missing, $x_{i+1} - x_i = 1$ and the correspondent coefficients

$a_i = (1-c)$ *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 $i = 1, \ldots, T-1$

$$S_{i+1} = a_i * (S_i + b_i (x_{i+1} - x_i)) + (1 - a_i) * \frac{y_i}{I_i}$$

$$I_{i+1} = \frac{y_i}{S_i}$$

$$b_{i+1} = d_i b_i + (1 - d_i) \frac{S_{i+1} - S_i}{x_{i+1} - x_i}$$

Step 3, For $i = T, \ldots K$

$$S_{i+1} = a_i * (S_i + b_i (x_{i+1} - x_i)) + (1 - a_i) \frac{y_{i+1}}{I_{i+1-T}}$$

$$I_{i+1} = e_i I_{i+1-T} + (1 - e_i) \frac{y_{i+1}}{S_{i+1}}$$

$$b_{i+1} = d_i b_i + (1 - d_i) \frac{S_{i+1} - S_i}{x_{i+1} - x_i}$$

Forecast for x is $(S_K + b_K (x - x_K)) * (I_j)^m$

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:

```
           Actual    Sales    West
                     Colas
                     =====
Jan                   2339
Feb                   2298
Mar                   2313
Apr                   2332
May                   2319
Jun                   2318.4
Jul                   2317.8
Aug                   2317.2
Sep                   2316.6
Oct                   2316
Nov                   2315.4
Dec                   2314.8
  Year               27817.2
```

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:

```
                   Colas      New York     Actual
                      Jan        Feb        Mar
                      ===        ===        ===
Direct Sales         678.557    645.874    675.299
Other Sales          411.299    389.554    423.547
Total Sales         1089       1035       1098
```

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:

```
                Year    Product    Market
                   Sales      COGS
                   =====      ====
Actual           400855     179336
```

```
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:

```
             Year     Product     Market
                Sales        COGS
                =====        ====
Actual          400855      179336
Budget          373080      158940
Variance %         7.4       (12.8)
```

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:

```
                              Jan      New
York
                            Actual        Budget
                            ======
======
Sales       Colas            678           640
            Root Beer        551           530
            Cream Soda       663           510
            Fruit Soda       587           620
            Diet Drinks      #MI           #MI
             Product        2479          2300

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:

$$\frac{n\sum x^2 - \left(\sum x\right)^2}{n^2}$$

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:

```
                                  Jan      New York
                                Actual        Budget
                                ======        ======
        Sales       Colas         678           640
                  Root Beer       551           530
                  Cream Soda      663           510
                  Fruit Soda      587           620
                  Diet Drinks     #MI           #MI
                   Product       2479          2300

        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->Jan->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.
- 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

Operator	Description
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
= =	Equal to
<> or !=	Not equal to
IN	In

- 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

Operator	Example	Result
>	@WITHATTR(Population,">","18000000")	Returns New York, California, and Texas
>=	@WITHATTR(Population,">=",10000000) where 10,000,000 is not a numeric attribute member, but a constant	Returns New York, Florida, California, Texas, Illinois, and Ohio
<	@WITHATTR(Ounces,"<","16")	Returns Cola, Diet Cola, Old Fashioned, Sasparilla, and Diet Cream

Table 2-35 (Cont.) Operator Results

Operator	Example	Result
<=	@WITHATTR("Intro Date","<=",@TODATE("mm-dd-yyyy", "04-01-2002"))	Returns Cola, Diet Cola, Caffeine Free Cola, and Old Fashioned
= =	@WITHATTR("Pkg Type","= =",Can)	Returns Cola, Diet Cola, and Diet Cream
<> or !=	@WITHATTR(Caffeinated,"<>",True)	Returns Caffeine Free Cola, Sasparilla, Birch Beer, Grape, Orange Strawberry
IN	@WITHATTR("Population","IN",Medium)	Returns Massachusetts, Florida, Illinois, and Ohio

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 */


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 */


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:

```
          Colas   New York   Actual
                     Sales       SalesAvg
                     =====       ========
2000
        Mar            678        678
        Apr            645        645
        May            675        666
        Jun            712        677.3
        Jul            756        714.3
        Aug            890        786
        Sep            924        856.7
        Oct            914        909.3
        Nov            912        916.7
        Dec            723        849.7
```

```
2001
        Jan             647         760.7
```

See Also

- @AVGRANGE
- @MAXRANGE
- @MAXSRANGE
- @MINRANGE
- @MINSRANGE
- @MOVAVG
- @MOVMAX
- @MOVMED
- @MOVMIN
- @MOVSUM
- @SPLINE
- @STDEVRANGE
- @SUMRANGE

@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:

```
@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:

  ```
  West = @XREF(SourceDb, @LEVMBRS(Market,0));
  ```

- The member list cannot contain ranges. For example, the following formula is not valid:

  ```
  West = @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:

  ```
  RedThings = @XREF(SourceDb, Sales, Red);
  ```

- *mbrList* can contain attribute operators. For example, the following formula calculates RedThings as the average sales of all red products:

  ```
  RedThings = @XREF(SourceDb, Sales, Red, Average);
  ```

- @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 @XREF queries that depend on dynamic and attribute members.
- Over the course of an @XREF calculation, data in the source database may change. @XREF does not incorporate changes made after the beginning of the calculation.
- @XREF is a top-down formula. For more information on top-down formulas, see Bottom-Up and Top-Down Calculation.
- For a member that does not exist in either the data source or the data target, @XREF returns the value of the top dimension, not the value #M1.
- If you are using @PARENT within @XREF, it must be within @NAME. For example:

  ```
  COGS=@XREF(Sample, @NAME(@PARENT(Product)),Sales);
  ```

- When running a parallel calculation that includes @XREF, the application times out if the number of threads you specify to use is higher than the configured number of SERVERTHREADS.

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

@XWRITE

Understand XREF/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.
- @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.

```
FIX (East, Actual, Budget, Sales)
"100-30" (
@XWRITE("100-30", @loopback, "100-20");
)
ENDFIX
```

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.

```
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.

```
FIX (East, Actual, Budget, Sales)
"100" (
 @XWRITE(@PARENT("100-30"), @loopback, "100-20");
 )
ENDFIX
```

See Also

@XREF

Understand XREF/XWRITE

Functions Supported in Hybrid Mode

The Essbase configuration setting ASODYNAMICAGGINBSO controls whether block storage databases use hybrid mode. Hybrid mode is the default for block storage cubes in Essbase 19c and Oracle Analytics Cloud - Essbase. See Adopt Hybrid Mode for Fast Analytic Processing for more information.

If enabled, hybrid mode 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
- @ICHILDREN
- @IDESCENDANTS
- @ILANCESTORS
- @ILDESCENDANTS
- @ILSIBLINGS
- @INT
- @INTEREST
- @INTERSECT
- @IRDESCENDANTS
- @IRR
- @IRSIBLINGS

- @ISACCTYPE
- @ISANCEST
- @ISATTRIBUTE
- @ISCHILD
- @ISDESC
- @ISGEN
- @ISIANCEST
- @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
- @RANGELASTVAL
- @RANK
- @RDESCENDANTS
- @RELATIVE

- @RELXRANGE
- @REMAINDER
- @REMOVE
- @RETURN
- @ROUND
- @RSIBLINGS
- @SHARE
- @SHIFT
- @SHIFTMINUS
- @SHIFTPLUS
- @SHIFTSIBLING
- @SIBLINGS
- @SLN
- @SPARENTVAL
- @SPLINE
- @SUBSTRING
- @SUM
- @SUMRANGE
- @TODATE
- @TODATEEX
- @TODAY
- @TRUNCATE
- @UDA
- @VAR
- @VARPER
- @VARIANCE
- @VARIANCEP
- @WEIGHTEDSUMX
- @WITHATTR
- @XRANGE
- @XREF

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

3
Calculation Commands

Calculation scripts enable you to develop custom operations to supplement the built-in calculation of the database outline.

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

Mathematical operators perform common arithmetic operations.

Table 3-1 Mathematical Operators

Operator	Description
+	Adds
-	Subtracts
*****	Multiplies

Table 3-1 (Cont.) Mathematical Operators

Operator	Description
/	Divides
%	Evaluates percentage, for example: *Member1%Member2* evaluates *Member1* as a percentage of *Member2*.
( )	Controls the order of calculations and nests equations and formulas

Conditional and Logical Operators

Conditional operators build logical condition into calculations.

Table 3-2 Conditional and Logical Operators

Operator	Description
IF | ELSE | ELSEIF | ENDIF	Tests conditions and calculates a formula based on the success or failure of the test
>	Data value is greater than
>=	Data value is greater than or equal to
<	Data value is less than
<=	Data value is less than or equal to
= =	**If** data value is equal to
< > or **!=**	Data value is not equal to
AND	Logical AND linking operator for multiple value tests. Result is TRUE if both conditions are TRUE. Otherwise the result is FALSE.[1]
OR	Logical OR linking operator for multiple value tests. Result is TRUE if either condition is TRUE. Otherwise the result is FALSE.[2]
NOT	Logical NOT operator. Result is TRUE if condition is FALSE. Result is FALSE if condition is TRUE.[3]

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.

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
- SET FRMLRTDYNAMIC
- SET MSG
- SET NOTICE
- SET REMOTECALC
- SET RUNTIMESUBVARS
- SET UPDATECALC

- USE_MDX_INSERT(for aggregate storage custom calculations only)

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.

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.

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
- SET CREATEBLOCKONEQ
- SET DATAEXPORTOPTIONS
- SET DATAIMPORTIGNORETIMESTAMP
- SET EMPTYMEMBERSETS
- SET FRMLBOTTOMUP
- SET FRMLRTDYNAMIC
- SET HYBRIDBSOINCALCSCRIPT
- SET MSG

- SET NOTICE
- SET REMOTECALC
- SET RUNTIMESUBVARS
- SET SCAPERSPECTIVE
- SET TRACE
- SET UPDATECALC
- USE_MDX_INSERT (for aggregate storage custom calculations only)
- THREADVAR
- VAR

& (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.

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):

  ```
  FIX("East")
      CLEARBLOCK ALL;
  ENDFIX
  ```

  The following command block sets New York data values to #MISSING without removing the blocks:

  ```
  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:

```
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.

```
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.

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 DATAEXPORTCOND 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 DATAEXPORTOPTIONS 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 DATAEXPORTOPTIONS
  {
  DataExportLevel "LEVEL0";
  };
DATAEXPORTCOND ("Sales">=1000);
FIX ("100-10","New York","Actual","Sales");
DATAEXPORT "File" ",” "jan.txt" "#MI";
ENDFIX;
```

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 intial 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
"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
"Sep","Budget",#NoValue,#NoValue,150,120,#Mi,980,1300,390
"Oct","Budget",#NoValue,#NoValue,110,70,#Mi,390,1180,110
"Nov","Budget",#NoValue,#NoValue,150,120,#Mi,110,1460,60
"Dec","Budget",#NoValue,#NoValue,150,120,#Mi,60,1300,-260
```

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.

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

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.

```
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

```
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 sub-section 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.

- 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` is the same as `((A OR B) AND C)`. However, if you use `(A OR (B AND C))`, Essbase evaluates the sub-expression in parentheses `(B AND 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:

  ```
  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 (*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:

```
VAR varName=1;
FIX (@relative(@memberat(@List("Product1","Product2"),varName),0))
   COMMANDS;
ENDFIX
```

Example

```
FIX (Budget)
   CALC DIM (Year, Measures, Product, Market);
ENDFIX
FIX (Budget, Jan, Feb, Mar, @DESCENDANTS(Profit))
   CALC DIM (Product, Market);
ENDFIX
```

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"))
```

See Also

- Calculate Selected Tuples
- EXCLUDE…ENDEXCLUDE
- LOOP...ENDLOOP
- SET EMPTYMEMBERSETS

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.

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 ););
ENDFIXPARALLEL
```

Related Topics

- FIXPARALLEL...ENDFIXPARALLEL
- THREADVAR

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 for this calculation.

The default behavior is determined by a database-level setting (Aggregate missing values: yes|no).

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 the CALCCACHE TRUE | FALSE configuration setting. 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.

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].
 OK/INFO - 1012899 - 9.690, 1, (42/132), 125, [ID_052230].
 OK/INFO - 1012899 - 7.192, 1, (38/132), 105, [ID_052073].
 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.
```

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.
 OK/INFO - 1012899 - 15.131, 1, (30/132), 53, [ID_051341].
 OK/INFO - 1012899 - 10.759, 2, (124/129), 211, [ID_050092].
 OK/INFO - 1012899 - 9.690, 1, (42/132), 125, [ID_052230].
 OK/INFO - 1012899 - 7.192, 1, (38/132), 105, [ID_052073].
```

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

Output Column	Description
Diagnostic Message ID	The message ID. For example, `OK/INFO - 1012899`. This ID can be used to extract diagnostic information from the application log into a file.
Time (secs)	Task execution time in seconds. For example, `15.131`. The tasks are listed in decreasing order based on execution time.
Thread ID	Calculation thread ID. For example, 1. This calculation uses up to 2 threads, so the thread ID will always be 1 or 2.

Table 3-3 (Cont.) Calc Diagnostic Output Columns

Output Column	Description
Task Index/Task Count	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.
Task ID	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.
Member Combinations	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].

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

```
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 for a list of these effects and for detailed information about how Essbase performs parallel calculation.

Example

```
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.

Notes

- A number of features are affected by parallel calculation. See Relationship Between CALCPARALLEL Parallel Calculation and Other Essbase Features 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.

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

```
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

```
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

```
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 FIX...ENDFIXregions. 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 FIX/ENDFIX.
- You can use multiple SET CREATENONMISSINGBLK commands in a calc script, each affecting calculations that follow. However, consider that each time SET CREATENONMISSINGBLK is encountered within a set of FIX and ENDFIX statements, the calculator cycles through the database, potentially affecting calculation performance.

Syntax

```
SET CREATENONMISSINGBLK ON|OFF;
```

Parameters

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.

OFF
Calculations are performed only on existing blocks. This is the default setting.

Notes

- SET CREATENONMISSINGBLK affects only creation of new blocks. If existing blocks become #MISSING after formula execution, they are not deleted.
- The value set by SET CREATENONMISSINGBLK stays in effect until the next SET CREATENONMISSINGBLK is processed, or the calculation script terminates.
- When the calculation script includes both SET CREATENONMISSINGBLK ON and SET MSG DETAIL, any non-stored #MISSING block is indicated in the application log.
- If SET MSG is set to SUMMARY, when SET CREATENONMISSINGBLK is set to ON, Essbase writes an entry to the application log stating that Create Non #MISSING Blocks is enabled.
- If SET MSG is set to SUMMARY, and SET CREATENONMISSINGBLK is set to 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.

```
FIX (Budget)
  SET MSG DETAIL;
  SET CREATENONMISSINGBLK ON;
  "Jan Rolling YTD Est"= (Jan-
>Actual+Feb+Mar+Apr+May+Jun+Jul+Aug+Sep+Oct+Nov+Dec);
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 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

Assigned Value	Sparse Member Formula Example	New Block Created?
Constant	`West = 350`	Yes
Non-constant	`West = California + 120`	Yes, if the Create Blocks on Equation setting is ON. Otherwise, no.
Non-constant	`West = California * 1.05`	Yes, if the Create Blocks on Equation setting is ON. Otherwise, no.

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.

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”)`.

```
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:

```
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

```
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 DATAEXPORTOPTIONS command specifies a set of option values that are in place until the next SET DATAEXPORTOPTIONS 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 60123456.029 38123456.6430
"Feb" "Actual" 645123 258123 3871234 9012345 5112345 112345678 14212345
24512345 2067123456 61912345 20411234 601234 37123456.98
"Mar" "Actual" 675 270 405 94 51 1 146 259 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",6.78123e+008,2.71123e+008,4.07e+008,9.41235e+017,5.11235e
+007,0,9.41235e+017,-9.41235e+017,2.10112e+009,6.44123e+008,2.06712e+009
,60.0186,-1.388e+011,-7.84362e+016
"Feb","Actual",645123,258123,387000,9.01235e+006,5.11235e+006,1.12346e+0
08,1.2647e+008,-1.26083e+008,2.06712e+009,6.19123e+007,2.04112e+007,59.9
886,-19544.1,-1.05069e+007
"Mar","Actual",675,270,405,94,51,1,146,259,2041,742,2108,60,38.3704,21.5
833
```

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 407123456.0 941234567890123456.0
51123456.0 0 145123456.0 262123456.0 2101123456.0 644123456.0
2067123456.0 60123456.029 38123456.6430
```

```
"Feb" "Actual" 645123 258123 3871234 9012345 5112345 112345678 14212345
24512345 2067123456 61912345 20411234 601234 37123456.98
"Mar" "Actual" 675 270 405 94 51 1 146 259 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",678123456.0000,271123456.0000,407000000.0000,941234567890
123520.0000,51123456.0000,0.0000,94123456794124698O.0000,-94123456753424
6910.0000,2101123456.0000,644123456.0000,2067123456.0000,60.0186,-138799
883591.4395,-78436213961187248.0000
"Feb","Actual",645123.0000,258123.0000,387000.0000,9012345.0000,5112345.
0000,112345678.0000,126470368.0000,-126083368.0000,2067123456.0000,61912
345.0000,20411234.0000,59.9886,-19544.0820,-10506947.3333
"Mar","Actual",675.0000,270.0000,405.0000,94.0000,51.0000,1.0000,146.000
0,259.0000,2041.0000,742.0000,2108.0000,60.0000,38.3704,21.5833
```

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 DataExporttDimHeader 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","Additions","Ending Inventory"
```

```
"100-10","New York","Jan","Actual",678,271,94,51,0,2101,644,2067
"100-10","New York","Feb","Actual",645,258,90,51,1,2067,619,2041
"100-10","New York","Mar","Actual",675,270,94,51,1,2041,742,2108
"100-10","New York","Apr","Actual",712,284,99,53,0,2108,854,2250
"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>
        0N
    </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] secs.
        </INFO>
  </MESSAGE>
```

SET DATAIMPORTIGNORETIMESTAMP

Specifies whether to ignore the outline timestamp captured at the time the data was exported.

Syntax

```
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 outine 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

```
SET DATAIMPORTIGNORETIMESTAMP ON;
DATAIMPORTBIN e:january\basic.bin
```

Specifies to ignore comparing the outline timestamp with the timestamp on the import tile, and to import the binary export file to the database on which the calculation script is running.

Related Topics

- DATAEXPORT
- DATAIMPORTBIN
- SET Commands

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.

```
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.

```
SET EMPTYMEMBERSETS ON;
...
FIX(@CHILDREN("100-10"))
  FIX("New York")
    Calc Dim(Year);
  ENDFIX
Calc Dim (Product);
ENDFIX
...
```

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.
- 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 HYBRIDBSOINCALCSCRIPT

This calculation command controls whether the subsequent blocks in the current calculation script execute in hybrid mode. This command can be used to enable or disable hybrid mode for specific blocks in a calculation script.

This setting applies only to block storage databases.

Hybrid aggregation for block storage databases means that wherever possible, block storage data calculation executes with efficiency similar to that of aggregate storage databases.

Syntax

```
SET HYBRIDBSOINCALCSCRIPT NONE|FULL;
```

Parameters

NONE
The calculation script runs in block storage mode. This is the default.

FULL
The calculation script runs in hybrid mode.

Notes

- The calculation commands CALC DIM and AGG are not supported in hybrid mode.
- Only formulas that have dynamic dependencies are supported in hybrid mode.

Example

```
SET HYBRIDBSOINCALCSCRIPT FULL;
```

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:

```
[Thu Mar 30 16:27:26 1995] local/Sample/Basic/Qatest/Info(1012669)
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]

 [Thu Mar 30 16:27:26 1995] local/Sample/Basic/Qatest/Info(1012669)
Calculator Information Message: Executing Block - [100], [East]

[Thu Mar 30 16:27:26 1995] local/Sample/Basic/Qatest/Info(1012669)

Calculator Information Message: Executing Block - [Product], [East]

[Thu Mar 30 16:27:26 1995] local/Sample/Basic/Qatest/Info(1012669)

Calculator Information Message: Executing Block - [100], [Market]

[Thu Mar 30 16:27:26 1995] local/Sample/Basic/Qatest/Info(1012669)

Calculator Information Message: Executing Block - [Product], [Market]

[Thu Mar 30 16:27:26 1995] local/Sample/Basic/Qatest/Info(1012669)
Calculator Information Message:

Total Block Created: [0.0000e+00] Blocks

Sparse Calculations: [4.0000e+00] Writes and [2.2000e+01] Reads

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.

<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

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".

```
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.

```
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:

```
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>
   </RTSV_HINT>;
};
```

SET SCAPERSPECTIVE

Sets the perspective for varying attribute calculations.

Syntax

```
SET SCAPERSPECTIVE (mbrName1) [, (mbrName2)] ... [,(mbrNamen)]) 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.

```
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
- @ISMBRWITHATTR
- @WITHATTR

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");
                ENDIF

                "Ending Inventory" = "Opening Inventory" + "Additions"
- "Sales";
   )
ENDFIX
```

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";
)

...
```

For more examples, see Tracing Calculations.

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 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
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 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 (a 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 client or MaxL shell (essmsh). When using the MaxL Shell, terminate all statements with a semicolon. Results are returned in the form of a grid.

See Also

- Analyzing and Moving Data with MDX
- Writing MDX Queries

MDX Query Format

Every query using the SELECT statement has the following basic format. Items in [brackets] are optional.

```
[<with_section>]
[<insert_clause>]
[<export_clause>]
SELECT [<axis_specification>
       [, <axis_specification>...]]
  <subselect> | FROM <cube_specification>
[WHERE [<slicer_specification>]]
```

Table 4-1 Description of MDX Query Elements

Item	Description
<with_section>	An optional section, beginning with the keyword WITH, in which you can define referenceable sets or members.
<insert_clause>	An optional clause for inserting tuples of data from a source to a target.
<export_clause>	An optional clause to save query results to a file on Essbase. This is an alternative to viewing the query output on a client.
SELECT	A literal keyword that must precede axis specifications.
[<axis_specification> [,<axis_specification>...]]	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.
[<subselect>]	An optional sub selection to filter an axis specification. See MDX Sub Select.
FROM	A literal keyword that must precede the cube specification.
<cube_specification>	The name of the database from which to select.
WHERE	A literal keyword that must precede the slicer specification, if one is used.

Table 4-1 (Cont.) Description of MDX Query Elements

Item	Description
<slicer specification>	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])`

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 4-2 BNF Notation Elements

Symbol	Description	Example
<word> (A word in angle brackets.)	The word presented in angle brackets is not meant to be literally used in a statement; its rules are further defined elsewhere.	When reading the following syntax, `SELECT <axis-specification> ...` 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 `<axis-specification> ::=` to get the definition).
<word> ::= (A word in angle brackets, followed directly by the symbol `::=`)	A definition, or BNF "production." The symbol `::=` can be interpreted to mean "is defined as." The word referred to elsewhere as the placeholder `<word>` is defined here, directly following `<word> ::=`.	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. `<tuple> ::= '(' <member> [,<member>]... ')'`
| The pipe symbol or "OR" symbol.	Precedes alternatives. The symbol `|` can be interpreted to mean "or."	The following syntax: `ON COLUMNS|ROWS|PAGES| CHAPTERS|SECTIONS` can be used to build any of the following literal statement parts: • `ON COLUMNS` • `ON ROWS` • `ON PAGES` • `ON CHAPTERS` • `ON SECTIONS`
WORD (Text in all caps.)	A query-grammar keyword, to be typed literally.	When reading the following syntax, `SELECT <axis-specification> ...` you know that `SELECT` is a keyword, and therefore should be typed literally into its proper location in the statement.

Table 4-2 (Cont.) BNF Notation Elements

Symbol	Description	Example

[<word>] or **[word]** or **[WORD]**
(Square brackets enclosing some word or item.)
	An optional element.	In the following high-level query syntax,
		
		`[<with_section>]`
		`SELECT`
		`[<axis_specification>`
		`[,`
		`<axis_specification>...]]`
		`FROM [<cube_specification>]`
		`[WHERE`
		`[<slicer_specification>]]`
		
		everything, technically, is optional except for SELECT and FROM. Therefore, a query containing only the words
		
		`SELECT FROM`
		
		would in fact be valid; however, it would select one consolidated data value from its best estimate of a cube context, which might not be very useful.

[, <word>...]
(A comma, a word, and an ellipsis, all enclosed in square brackets.)
	You can optionally append a comma-separated list of one or more `<words>`.	The following syntax
		
		`SELECT`
		`[<axis_specification>`
		`[,`
		`<axis_specification<...]]`
		
		indicates that multiple, comma-separated axis specifications can optionally be supplied to the `SELECT` statement.

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>]]

<insert_specification> ::=
  INSERT
   <source_tuple> TO <target_tuple>
   ....
   <source_tuple> TO <target_tuple>
   [<offset> <debitmember> <creditmember>]
   [USING <load_buffer_method>]
   INTO <cube_specification>
   <subselect>

<export_specification> ::=
EXPORT INTO FILE <file_name> [<OVERWRITE> <USING COLUMNDELIMITER
<delimiter_character>>]

<subselect> ::=
FROM SELECT [<axis_specification>
       [, <axis_specification>...]]

<cube_specification> ::=
       '[' <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'`

```
<axis_specification> ::=
        [NON EMPTY] <set> [<dim_props>] ON
        COLUMNS | ROWS | PAGES | CHAPTERS |
        SECTIONS | AXIS (<unsigned_integer>)
```

```
<dim_props> ::=
        [DIMENSION] PROPERTIES <property> [, <property>...]

<slicer_specification> ::= <set> | <tuple> | <member>
```

> **Note:**
>
> The cardinality of the `<set>` in the slicer should be 1.

```
<member> ::=
        <member-name-specification>
      | <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.

   For attribute members, the long name (qualified to uniquely identify the member) should be used; for example, `[Ounces_12]` instead of just `[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> ::=
        Parent ( <member> [,<hierarchy>] )
            | <member>.Parent [(<hierarchy>)]
      | FirstChild ( <member> )
            | <member>.FirstChild
      | LastChild ( <member> )
```

```
            | <member>.LastChild
        | PrevMember ( <member> [,<layertype>] )
            | <member>.PrevMember [( <layertype> ) ]
        | NextMember ( <member> [,<layertype>] )
            | <member>.NextMember [( <layertype> ) ]
        | FirstSibling ( <member> [,<hierarchy>])
            | <member>.FirstSibling [(<hierarchy>)]
        | LastSibling ( <member> [,<hierarchy>])
            | <member>.LastSibling [(<hierarchy>)]
        | Ancestor ( <member> , <layer> | <index> [,<hierarchy>] )
        | Lead ( <member>, <index> [,<layertype>] [,<hierarchy>] )
            | <member>.Lead ( <index> [,<layertype>] [,<hierarchy>] )
        | Lag  ( <member>, <index> [,<layertype>] [,<hierarchy>] )
            | <member>.Lag  ( <index> [,<layertype>] [,<hierarchy>] )
        | CurrentAxisMember()
        | CurrentMember ( <dim_hier> )
            | <dim_hier>. CurrentMember
        | DefaultMember ( <dim_hier> )
            | <dim_hier>. DefaultMember
        | OpeningPeriod ( [<layer> [,<member>]] )
        | ClosingPeriod ( [<layer> [,<member>]] )
        | Cousin (<member>, <member>)
        | ParallelPeriod( [<layer>[, <index>[, <member> [,<hierarchy>]]]] )
        | Item ( <tuple>, <index> )
            | tuple[.Item] ( <index> )
        | LinkMember ( <member>, <hierarchy> )
            | member.LinkMember ( <hierarchy> )
         | DateToMember (<date>, <dim_hier> [ ,<genlev>] )
        | StrToMbr (<string_value_expr> [,<dimension>] [, MEMBER_NAMEONLY
| <alias_table_name>])

<dim_hier> ::= <dimension>

<dimension> :: =
        <dimension-name-specification>
      | Dimension ( <member> | <layer> )
            | <member>.DIMENSION
            | <layer>.DIMENSION

<dimension-name-specification> ::=
    Same as <member_name-specification> case 1.
    e.g. Product, [Product]

<hierarchy> ::=


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.

<layertype> ::=
        GENERATION | LEVEL
```

```
<layer> ::=
        <layer-name-specification>
      | Levels ( <dim_hier>, <index> )
           | <dim_hier>.Levels ( <index> )
      | Generations ( <dim_hier>, <index> )
           | <dim_hier>.Generations ( <index> )
      | <member>.Generation
      | <member>.Level

<layer-name-specification> ::=
```

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.

```
<tuple> ::=
        <member>
      | ( <member> [,<member>].. )
      | <tuple_value_expression>
```

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.

```
<tuple_value_expression> ::=
        CurrentTuple ( <set> )
           | <set>.Current
      | Item ( <set>, <index> )
           | <set>[.Item] (<index>)
```

A **set** is a collection of tuples where members in all tuples must be from the same dimensions and in the same order.

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.

```
<set> ::=
        MemberRange ( <member>, <member>
                    [,<layertype>] [,<hierarchy>] )
      | <member> : <member>
      | { <tuple>|<set> [, <tuple>|<set>].. }
      | ( <set> )
      | <set_value_expression>

<set_value_expression> ::=
      | Members ( <dim_hier> )
          | <dim_hier>.Members
      | Members ( <layer> )
          | <layer>.Members
      | Children ( <member> )
          | <member>.Children
      | CrossJoin ( <set> , <set> )
      | CrossJoinAttribute ( <set> , <set> )
      | Union ( <set> , <set> [,ALL] )
      | Intersect ( <set> , <set> [,ALL])
      | Except ( <set> , <set> [,ALL])
      | Extract ( <set> , <dim_hier> [, <dim_hier>].. )
      | Head ( <set> [, <index>])
      | Subset ( <set> , <index> [,index])
      | Tail ( <set> [,index])
      | Distinct ( <set> )
      | Siblings ( <member> [, <selection_flags>, [INCLUDEMEMBER|
EXCLUDEMEMBER]] )
          | <member>.Siblings
      | Descendants ( <member> , [{<layer>|<index>}[, <Desc_flags>]] )
      | PeriodsToDate ( [<layer>[, <member> [,<hierarchy>]]] )
      | LastPeriods ( <index>[, <member> [,<hierarchy>]] )
      | xTD ( [<member>] )
           where xTD could be {HTD|YTD|STD|PTD|QTD|MTD|WTD|DTD}
      | Hierarchize ( <set> [,POST] )
      | Filter ( <set> , <search_condition> )
      | Order ( <set>, <value_expression> [,BASC | BDESC] )
      | TopCount ( <set> , <index> [,<numeric_value_expression>] )
      | BottomCount ( <set> , <index> [,<numeric_value_expression>] )
      | TopSum ( <set> , <numeric_value_expression>
                       , <numeric_value_expression> )
      | BottomSum ( <set> , <numeric_value_expression>
                          , <numeric_value_expression> )
      | TopPercent ( <set> , <percentage> , <numeric_value_expression> )
      | BottomPercent ( <set> , <percentage> ,
<numeric_value_expression> )
      | Generate ( <set> , <set> [, [ALL]] )
      | DrilldownMember ( <set> , <set>[, RECURSIVE] )
```

```
      | DrillupMember ( <set> , <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> )
      | Stddev ( <set> [,<numeric_value_expression> [,IncludeEmpty] ] )
      | Stddevp ( <set> [,<numeric_value_expression> [,IncludeEmpty] ] )
      | 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"
       | "yymmdd"
       | "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(").

```
<conditional_expression> ::=
        <if_expression>
      | <case_expression>
      | CoalesceEmpty ( <numeric_value_expression>
                      , <numeric_value_expression>)

<case_expression> ::=
        <simple_case> | <searched_case>

<if_expression> ::=
        IIF ( <search_condition>, <true_part>, <false_part> )
<true_part> ::=
        <value_expression>  | <set>
<false_part> ::=
        <value_expression> | <set>

<simple_case> ::=
        Case <case_operand>
             <simple_when_clause>...
```

```
            [ <else_clause> ]
        END

<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.

```
<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>.
      | .<unsigned_integer>
```

```
<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.

> **Note:**
>
> These rules are also applicable if you need to use MDX to explicitly reference shared member names in a unique member name outline (an outline with duplicate member names NOT enabled). See the "Shared Member Names Example" in this topic.

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.

  ```
  [DimensionMember].[Ancestors...].[DuplicateMember]
  ```

  For example:

  ```
  [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 4-3 Construction of Shortcut Qualified Member Names

Scenario	Qualified Name Syntax	Example
Duplicate member names exist at generation 2	`[DimensionMember].[DuplicateMember]`	[Year].[Jan] or [Product].[Jan]
Duplicate member names exist in an outline, but are unique within a dimension	`[DimensionMember]@[DuplicateMember]`	[Year]@[Jan]
Duplicate member names have a unique parent	`[ParentMember].[DuplicateMember]`	[East].[New York]
Duplicate member names exist at generation 3	`[DimensionMember].[ParentMember].[DuplicateMember]`	[Products].[Personal Electronics].[Televisions]
Duplicate member names exist at a named generation or level, and the member is unique at its generation or level	`[DimensionMember]@[GenLevelName]|[DuplicateMember]`	[2006]@[Gen1]|[Jan]

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:

```
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.

```
<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 4-4 Axis Keywords and Corresponding Ordinal Notation

Axis Keyword	Axis Ordinal
COLUMNS	AXIS(0) (default if nothing specified)
ROWS	AXIS(1)

Table 4-4 (Cont.) Axis Keywords and Corresponding Ordinal Notation

Axis Keyword	Axis Ordinal
PAGES	AXIS(2)
CHAPTERS	AXIS(3)
SECTIONS	AXIS(4)

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 4-5 Hypothetical Subset of Data

(axis)	Member names in *set1*
Member names in *set2*	Data at intersections of *set1* and *set2* members

The examples above are hypothetical because they will not return a cube until values are provided for the *set*s. 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 4-6 Output Grid from MDX Example

(axis)	100-10	100-20
Qtr1	5096	1359
Qtr2	5892	1534
Qtr3	6583	1528
Qtr4	5206	1287

Specifying the Set

You can represent the sets in each axis in many ways.

```
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.

```
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:

```
<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 4-7 Output Grid from MDX Example

	Qtr1				
Actual	#Missing	#Missing	#Missing	#Missing	#Missing

For another example, see the Tail function.

> **Note:**
>
> `NON EMPTY` syntax is not supported in an MDX sub select axis specification.
>
> To provide the best data export performance in MDX, any `NON EMPTY` specification on an axis is ignored for MDX Export.

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:

```
<axis_specification> ::=
        [NON EMPTY] <set> [<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

```
[WHERE [<slicer_specification> [<dim_props>]]

      <slicer_specification> ::= <set> | <tuple> | <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> ::=
```

```
        '<ident>'
        |<ident>
```

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 4-8 Ways To Specify an MDX Set

Item	Description
MemberRange (<member>, <member>)	A set can be a range of members, specified using the MemberRange function.
<member> : <member>	Alternate syntax that has the same effect as the MemberRange function.
{[<tuple> | <set>] [, <tuple> | <set>].. }	Unless it is returned by a function, a set must be enclosed in curly braces { }. A set can be one or more tuples, or it can be made up of other sets. All tuples in a set must have the same dimensionality.
<set_value_expression>	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.

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

```
WITH
      SET set_name AS ' set '
      | MEMBER calculated_member_name AS ' <numeric_value_expr> '
      [, <solve_order_specification> ]
      | <perspective_specification>
```

Table 4-9 MDX WITH Section Elements

Item	Description

set_name	The name of the set that will be defined after the AS 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 `Best5Books`, which names a set of the five top-selling paperback titles in December:
	
	`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.

Table 4-9 (Cont.) MDX WITH Section Elements

Item	Description

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 4-9 (Cont.) MDX WITH Section Elements

Item	Description

<solve_order_specification>	Optional. By adding `,SOLVE_ORDER = n` 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:
	
	`WITH`
	`MEMBER [Product].[mbr1] AS`
	`'calculation', SOLVE_ORDER = 2`
	
	`MEMBER [Product].[mbr2] AS`
	`'calculation', SOLVE_ORDER = 1`
	
	`SELECT`
	`{[Year].children}`
	`on columns,`
	`{`
	`[Product].[mbr1],`
	`[Product].[mbr2]`
	`}`
	`on rows`
	
	See Usage Examples for Solve Order.

<perspective_specification>	`PERSPECTIVE REALITY | tuple FOR dimension` When a database uses varying attributes, base members associated with the varying attributes are aggregated according to the specified perspective. You can set the perspective to reality (using the REALITY keyword) or to explicit (using an input tuple consisting of level 0 members). Reality-based evaluation and reporting is the default, in which independent members are determined by the current context. When using explicit evaluation and reporting, you specify a tuple of level 0 members from the independent dimension to be used as the context. 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.

Usage Examples for Solve Order

```
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:

```
([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 4-10 Ways to Specify a Dimension in MDX

Syntax	Description
<dimension-name-specification>	A dimension name. See Description, item 1.
<member>.DIMENSION	**Dimension** function with a member specification as input.
<layer>.DIMENSION	**Dimension** function with a layer specification as input.
DIMENSION (<member> | <layer>)	Alternate syntax. Dimension (<member>) has the same effect as <member>.Dimension. Dimension (<layer>) has the same effect as <layer>.Dimension.

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 `{([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

```
<layer> ::=
        <layer-name-specification>
      | Levels ( <dim_hier>, <index> )
           | <dim_hier>.Levels ( <index> )
      | Generations ( <dim_hier>, <index> )
           | <dim_hier>.Generations ( <index> )
      | <member>.Generation
      | <member>.Level
```

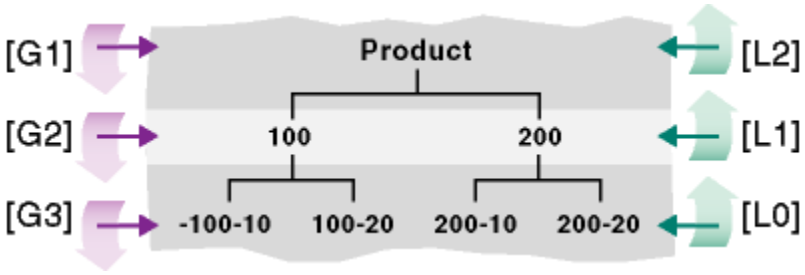
Table 4-11 Ways to Specify a Layer in MDX

Syntax	Description
<layer-name-specification>	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.
<dimension>.Levels (<index>)	Levels function with the dimension specification and a level number as input. For example, `[Year].Levels(0)`.
Levels (<dimension>, <index>)	Alternate syntax for Levels function with the dimension specification and a level number as input. For example, `Levels ( [Year], 0 )`.
<dimension>.Generations (<index>)	Generations function with the dimension specification and a generation number as input. For example, `[Year].Generations (3)`.
Generations (<dimension>, <index>)	Alternate syntax for Generations function with the dimension specification and a generation number as input. For example, `Generations ( [Year], 3)`.
<member>.Generation	Generation function with a member specification as input. For example, `[Year].Generation`. Returns the generation of the specified member.
<member>.Level	Level function with a member specification as input. For example, `[Year].Level`. Returns the level of the specified member.

Description

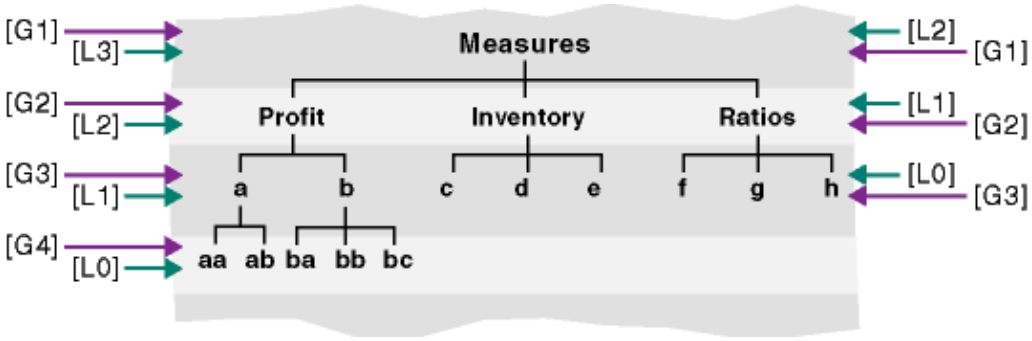
Generation numbers begin counting with 1 at the dimension name; higher generation numbers are those that are closest to leaf members in a hierarchy.

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

```
<member> ::=
           <member-name-specification>
         | <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 4-12 Ways to Specify a Tuple in MDX

Syntax	Description
<member>	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]`
(<member> [, <member>]..)	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] )`
<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])
   )
  }
```

```
'
 WHERE ([Market].[East], [Scenario].[Actual])
```

Notes

- Only 16 session-based named sets maybe stored simultaneously.
- Named set definitions may not contain references to other named sets.

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:

```
[<with_section>]
SELECT <axis_specification>
       [, <axis_specification>...]
  <subselect>
 [WHERE [<slicer_specification>]]
```

Where `<subselect>` is:

```
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 4-13 MDX INSERT Clause Elements

Item	Description
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 4-1 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 4-2 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 4-3 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 4-4 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
      ({[Jan]},
       Crossjoin([Total Expenses].Children, {[100],[200]})
     ), {[New York]})
   ON ROWS
   FROM
```

```
   [Sample1].[Basic]
 );
```

The above MDX example has similar functionality to a block storage allocation as shown in the following calc script example:

```
FIX("Total Expenses", {[Jan]}, [[New York]])
"PY Actual" = @ALLOCATE("PY Actual"->"Total Expenses",@CHILDREN("Total
Expenses"), "Budget",,share);
ENDFIX
```

Example 4-5 Inserting Using Member Context

The following example updates the revised payroll based on previous year context.

```
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 4-6 Performing a Custom Calculation

The following example runs a custom calculation on an aggregate storage database.

```
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])-
0.00000020e7
   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 4-7 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
 'IIF(IS([Organisation].CurrentMember, [ORG00])
  AND IS([Cost Centre].currentMember,[CC19000])
  AND [Amount Type].[AT1] < 0, [Amount Type].[AT1] * -1, Missing)'

 MEMBER [Amount Type].[AT6]
 AS
 'IIF (IS([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)
      )
     )
    )}
   ON ROWS
   FROM [Gl].[Basic]
```

```
  WHERE ([ANLT],[OUTT],[Scenario],[PUBT],[FRED],[Feb-08])
);
```

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 4-14 MDX EXPORT Clause Elements

Item	Description
file_name	The name of a text file in which to save the exported MDX query results. A file extension is not required.
OVERWRITE	Optional keyword specifying that if *file_name* already exists, overwrite it.
USING COLUMNDELIMITER *delimiter_character*	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.

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):

```
 Axis-1              (File)
+-------------------+-------------------
 (Mdx Export)                          1
```

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 4-15 Mathematical Operators in MDX

Operator	Definition
+	Adds. Also can be used as a unary operator.
-	Subtracts. Also can be used as a unary operator; for example, `-5`, `-(Profit)`.
*	Multiplies.
/	Divides.
%	Evaluates percentage. For example, Member1%Member2 evaluates Member1 as a percentage of Member2. **Note**: Aggregate storage outline formulas cannot contain the % operator. In outline formulas, replace % with expression: `(value1/value2)*100)`

Conditional and Logical Operators

Conditional operators take two operands and check for relationships between them, returning TRUE or FALSE.

Table 4-16 Conditional and Logical Operators in MDX

Operator	Definition
>	Data value is greater than.

Table 4-16 (Cont.) Conditional and Logical Operators in MDX

Operator	Definition
<	Data value is less than.
=	Data value is equal to.
<>	Data value is not equal to.
>=	Data value is greater than or equal to.
<=	Data value is less than or equal to.

IN	The syntax for the IN operator is as follows:
	
	`<property> IN <member>| <character_string_literal>`
	
	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:
	
	`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 4-17 Boolean Operators in MDX

Operator	Definition
AND	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 IsValid.
OR	Logical OR linking operator for multiple value tests. Result is TRUE if either condition is TRUE. Otherwise the result is FALSE.
NOT	Logical NOT operator. Result is TRUE if condition is FALSE. Result is FALSE if condition is TRUE. For an example using NOT, see IsEmpty.
XOR	Logical XOR linking operator for multiple value tests. Result is TRUE if *only* one condition is TRUE. Otherwise the result is FALSE.

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 4-18 PROPERTY_EXPR Parameters

Parameter	Description
dimension name	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.
property_name	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).
member_value_expression	Member value expression. See `<member_value_expression> ::=` in MDX Grammar Rules.
display_name	Character string literal. The display name to use for the queried properties information in the query output.

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(),
```

```
      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 4-19 Output Grid from MDX Example

(axis)	Axis-1.properties	100
[New York]	Parent_level_1 = East, Parent_level_2 = market	3498
[Massachusetts]	Parent_level_1 = East, Parent_level_2 = market	5105
[Florida]	Parent_level_1 = East, Parent_level_2 = market	2056
...	...	...

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.

NONEMPTYTUPLE

```
NONEMPTYTUPLE "("nonempty_member_list")"
```

where *nonempty_member_list* is one or more comma-separated member names or calculated member names, each from different dimensions.

If any formula-dependent dimension is omitted from *nonempty_member_list*, it may lead to incorrect results, as not all dimensions will be added to the formula cache.

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 4-20 Output Grid from MDX Example

(axis)	Items Per Package	3 Month Units
(017589, Carrie)	610	1808
(020408, Debra)	584	1778
(020486, Kalluri)	551	1670
(047108, Kimberley)	593	1723
(051273, Madhukar)	541	1642
(056098, Melisse)	607	1750
...	...	...

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.

```
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.

```
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.

```
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.

```
 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.

```
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:

```
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 4-21 MDX Query Limit Concepts

Concept	Description
NON EMPTY processing	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.

Table 4-21 (Cont.) MDX Query Limit Concepts

Concept	Description

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.
	`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.
	`UNION( CROSSJOIN(Products.LEVELS(0).MEMBERS, [Market].LEVELS(0).MEMBERS) , {([Product].Product_1, [Market].Market_1)} )`

Compact set	A set is stored in compact form if it can be internally represented as a cluster or symmetric set.

Table 4-21 (Cont.) MDX Query Limit Concepts

Concept	Description

Flattened set	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.
	`{(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)}`

Asymmetric set	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.
	`UNION({(Colas, East)}`
	`CROSSJOIN(`
	`[Product].CHILDREN,`
	`[Market].CHILDREN))`

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 4-22 MDX Query Limit Descriptions and Units

Limitations	Units
Number of cells in a query region defined by all axis sets in an MDX query with NON EMPTY clause	2^{640}
Number of cells that can be returned to a client after NON EMPTY processing	2^{32}
Number of cells in a query region defined by all axis sets in an MDX query with no NON EMPTY clause	2^{32}
Number of tuples in an axis set with NON EMPTY directive after NON EMPTY processing	2^{28}
Size of a set in compact form	2^{640}
Size of a set in flattened form	2^{32}
Number of elements in a set	2^{32}
Number of members (from all dimensions) in a cluster element	2^{32}
Number of cells in a query after applying non empty cell processing	2^{32}
Size of a set that can be processed by the following functions: • Distinct • Except • Filter • Intersect • Ntile • Order • Percentile • Rank • TopPercent • BottomPercent • TopSum • BottomSum • Hierarchize • Union (with removal of duplicates) • NonEmptySubset (output set size) • TopCount (output set size) • BottomCount (output set size)	Less than 2^{28}

Table 4-22 (Cont.) MDX Query Limit Descriptions and Units

Limitations	Units
IEssOpMdxQuery Java API interface or EssMdx C API functions	• Maximum number of tuples/clusters on an axis—2^{29}-1 • Maximum number of cells (when cell status is requested)—2^{26}-1 • Maximum number of cells (when cell status is not requested)—approximately 2^{27}-1
MDX queries run through MaxL	• Maximum number of columns—2^{29}-1 • Maximum number of rows—2^{29}-1

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 4-23 provides specific information and examples. Then identify functions for which an indirect rewrite is required. Table 4-23 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 4-23 Calculator to MDX Function Mapping

Calculator	MDX	Remarks/Examples

@ABS	Abs	**Calculator**
		
		`@ABS(Actual-Budget)`
		
		MDX
		
		`Abs([Actual]-[Budget])`

@ALLANCESTORS	Ancestors	Shared members are not relevant to aggregate storage outlines.
@ALIAS	Not required.	In MDX, the argument to @ALIAS can be passed as-is to the outer function.

@ANCEST	Ancestor with CurrentMember as input. Use a tuple to combine the result with the optional third argument to the @ANCEST function.	**Calculator**
		
		`@ANCEST(Product,2,Sales)`
		
		MDX
		
		`(`
		`  Sales,`
		`  Ancestor(`
		`  Product.CurrentMember,`
		`  Product.Generations(2)`
		`  )`
		`)`

Table 4-23 (Cont.) Calculator to MDX Function Mapping

Calculator	MDX	Remarks/Examples

@ANCESTORS	Ancestors	**Calculator**
		
		`@ANCESTORS("New York")`
		
		MDX
		
		`Ancestors([New York].parent, [Market].levels(2))`

@ANCESTVAL	Ancestor with CurrentMember as input. Use a tuple to combine the result with the optional third argument to the @ANCESTVAL function.	**Calculator**
		
		`@ANCESTVAL(Product,2,Sales)`
		
		MDX
		
		`(Sales, Ancestor( Product.CurrentMember, Product.Generations(2) ) ).Value`

@ATTRIBUTE	Attribute	**Calculator**
		
		`@ATTRIBUTE(Can)`
		
		MDX
		
		`Attribute([Can])`

@ATTRIBUTEBVAL	[*BaseDim*] .CurrentMember.*Attribute Dim*	See About MDX Properties.**Calculator**
		
		`@ATTRIBUTEBVAL(Caffeinated)`
		
		MDX
		
		`Product.CurrentMember.Caffeinated`

Table 4-23 (Cont.) Calculator to MDX Function Mapping

Calculator	MDX	Remarks/Examples

@ATTRIBUTESVAL	[*BaseDim*] .CurrentMember.*AttributeDim*	See About MDX Properties.**Calculator**
		
		`@ATTRIBUTESVAL("Pkg Type")`
		
		MDX
		
		`Product.CurrentMember.[Pkg Type]`

@ATTRIBUTEVAL	[*BaseDim*] .CurrentMember *AttributeDim*	See About MDX Properties.**Calculator**
		
		`@ATTRIBUTEVAL(Ounces)`
		
		MDX
		
		`Product.CurrentMember.Ounces`

Table 4-23 (Cont.) Calculator to MDX Function Mapping

Calculator	MDX	Remarks/Examples

@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 4-23 (Cont.) Calculator to MDX Function Mapping

Calculator	MDX	Remarks/Examples

@AVGRANGE	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.	**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) ) )`

@CHILDREN	Children	**Calculator**
		
		`@CHILDREN(Market)`
		
		MDX
		
		`Children(Market)`
		
		or
		
		`Market.Children`

Table 4-23 (Cont.) Calculator to MDX Function Mapping

Calculator	MDX	Remarks/Examples

@CONCATENATE	Concat	**Calculator**
		
		`@MEMBER(@CONCATENATE("Qtr1", "1"));`
		
		MDX
		
		`Concat("01", "01")`

@CORRELATION	Not supported in MDX.	.

@COUNT	Use Count if SKIPNONE.
	Use NonEmptyCount if SKIPMISSING.
	For SKIPZERO, see the example in the next column.
	For SKIPBOTH, use Count (Filter(*set*, *value* <> 0 && *value* <> MISSING))
		Calculator
		
		`@COUNT(SKIPMISSING,@RANGE(Sales, Children(Product)))`
		
		MDX
		
		`NonEmptyCount(CrossJoin({Sales},{Product.Children}))`
		
		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 4-23 (Cont.) Calculator to MDX Function Mapping

Calculator	MDX	Remarks/Examples

@CURGEN	Generation (CurrentMember(*dimension*))	**Calculator**
		`@CURGEN(Year)`
		MDX
		`Year.CurrentMember.Generation`

@CURLEV	Level (CurrentMember(*dimension*))	**Calculator**
		`@CURLEV(Year)`
		MDX
		`Year.CurrentMember.Level`

@CURRMBR	CurrentMember	**Calculator**
		`@CURRMBR(Product)`
		MDX
		`[Product].CurrentMember`

@CURRMBRRANGE	RelMemberRange	**Calculator**
		`@CURRMBRRANGE(Year, LEV, 0, -1, 1)`
		MDX
		`RelMemberRange (Year.CurrentMember, 1, 1, LEVEL)`

@DESCENDANTS	Descendants (*member*)	See MDX Descendants documentation for examples.

Table 4-23 (Cont.) Calculator to MDX Function Mapping

Calculator	MDX	Remarks/Examples

@EXP	Exp	**Calculator**
		
		`@EXP("Variance %"/100);`
		
		MDX
		
		`Exp([Scenario].[Variance %]/100)`

@FACTORIAL	Factorial	**Calculator**
		
		`@FACTORIAL(5)`
		
		MDX
		
		`Factorial(5)`

@GEN, @LEV	Generation, Level	.
@GENMBRS, @LEVMBRS	layer.Members	.
@IALLANCESTORS	Ancestors	Shared members are not relevant to aggregate storage outlines.
@IANCESTORS	Ancestors	Shared members are not relevant to aggregate storage outlines.

@ICHILDREN	Union(*member*, *member*.Children)	**Calculator**
		
		`@ICHILDREN(Market)`
		
		MDX
		
		`Union({Market}, {Market.children})`

@IDESCENDANTS	Descendants(*member*)	**Calculator**
		
		`@IDESCENDANTS(Market)`
		
		MDX
		
		`Descendants(Market)`

Table 4-23 (Cont.) Calculator to MDX Function Mapping

Calculator	MDX	Remarks/Examples

@ILSIBLINGS	MemberRange (*member*.FirstSibling,*member*)	**Calculator**
		`@ILSIBLINGS(Florida)`
		MDX
		`MemberRange(Florida.FirstSibling, Florida.Lag(1))`

@INT	Int	**Calculator**
		`@INT(104.504)`
		MDX
		`Int(104.504)`

@ISACCTYPE	IsAccType	See MDX IsAccType documentation for examples.

@ISANCEST	IsAncestor	**Calculator**
		`@ISANCEST(California)`
		MDX
		`IsAncestor(Market.CurrentMember, California)`

@ISCHILD	IsChild	See MDX IsChild documentation for examples.

Table 4-23 (Cont.) Calculator to MDX Function Mapping

Calculator	MDX	Remarks/Examples

@ISDESC	See examples.	**Calculator**
		
		`@ISDESC(Market)`
		
		MDX
		
		`IsAncestor([Market], [Market].Dimension.CurrentMember)`
		
		or
		
		`Count(Intersect({Member.Descendants},`
		
		`{Member.dimension.CurrentMember}) = 1`

@ISGEN	IsGeneration	**Calculator**
		
		`@ISGEN(Market, 2)`
		
		MDX
		
		`IsGeneration(`
		`  Market.CurrentMember, 2)`

@ISIANCEST	IIF(Is(*member*, *ancestormember*) OR IsAncestor(*member*, *ancestormember*), <true-part>, <false-part>)	**Calculator**
		
		`@ISIANCEST(California)`
		
		MDX
		
		`IIF(`
		`  IS(Market.CurrentMember, California)`
		`   OR`
		
		`IsAncestor(Market.CurrentMember, California),`
		`  <true-part>, <false-part>`
		`)`

@ISIBLINGS	Siblings(*member*)	Returns a set that includes the specified member and its siblings.

Table 4-23 (Cont.) Calculator to MDX Function Mapping

Calculator	MDX	Remarks/Examples

@ISICHILD	IIF(Is(*member*, *childmember*) OR IsChild(*member*, *childmember*), <true-part>, <false-part>)	**Calculator**
		
		`@ISICHILD(South)`
		
		MDX
		
		`IIF(`
		
		`Is(Market.CurrentMember,South)`
		`  OR`
		
		`IsChild(Market.CurrentMember,South),`
		` <true-part>, <false-part>`
		`)`

@ISIDESC	See examples.	**Calculator**
		
		`@ISIDESC(South)`
		
		MDX
		
		`(Count(Intersect({[South].Descendants}, {South}) = 1`
		`OR`
		`Is(CurrentMember, [South]))`

@ISIPARENT	IIF(Is(*member*, *parentmember*)	**Calculator**
		
		`@ISIPARENT(Qtr1)`
		
		MDX
		
		`IIF(`
		`  Is(Time.CurrentMember, [Qtr1])`
		`  OR`
		`  IsChild([Qtr1], Time.CurrentMember),`
		`  <true-part>, <false-part>)`

Table 4-23 (Cont.) Calculator to MDX Function Mapping

Calculator	MDX	Remarks/Examples

@ISISIBLING	IsSibling(*member*, *siblingmember*)	**Calculator**
		
		`@ISISIBLING(Qtr2)`
		
		MDX
		
		`IIF( IsSibling( [Qtr2], Time.CurrentMember ), <true-part>, <false-part> )`

@ISLEV	IsLevel	.

@ISMBR	IIF(Count(Intersect (*member-set*, *member*)) = 1, *true-part*, *false-part*)	Calculator allows a collection of members or cross members that do not subscribe to the rules of an MDX set to appear as the second argument. This functionality cannot be easily replicated without enumerating each element of the second set and testing for intersection.
		However, if the second argument subscribes to MDX set rules then the translation is easier, as shown. For example:
		Calculator
		
		`@ISMBR("New York":"New Hampshire")`
		
		MDX
		
		`IIF( Count( Intersect( {MemberRange([New York],[New Hampshire])}, {Market.CurrentMember} ) ) = 1, <true-part>, <false-part> )`

Table 4-23 (Cont.) Calculator to MDX Function Mapping

Calculator	MDX	Remarks/Examples

@ISPARENT	Use IsChild.	**Calculator**
		`@ISPARENT("New York")`
		MDX
		`IsChild(Market.CurrentMember,[New York])`

@ISSAMEGEN,@ISSAMELEV	IIF (*member*.Generation = CurrentMember(*dimension*).Generation, <true-part>, <false-part>)	**Calculator**
		`@ISSAMEGEN(West)`
		MDX
		`IIF( Ordinal( Market.CurrentMember.Generation ) = Ordinal(West.Generation), <true-part>, <false-part> )`

@ISSIBLING	IsSibling	See MDX IsSibling documentation for examples.
@ISUDA	IsUda	See MDX IsUda documentation for examples.
@LIST	.	If the member set does not subscribe to MDX set rules, then explicit enumeration is required. For *rangelist* use CrossJoin(*member*, *set*).
@LN, @LOG, @LOG10	Ln, Log, Log10	.

Table 4-23 (Cont.) Calculator to MDX Function Mapping

Calculator	MDX	Remarks/Examples

@LSIBLINGS
@RSIBLINGS
	MemberRange(*member*.FirstSibling, *member*.Lag(1))
	MemberRange(*member*.Lead(1), *member*.LastSibling)
		Calculator
		`@LSIBLINGS(Qtr4)`
		MDX
		`MemberRange([Qtr4].FirstSibling, [Qtr4].Lag(1))`
		Calculator
		`@RSIBLINGS(Qtr1)`
		MDX
		`MemberRange([Qtr1].Lead(1), [Qtr1].LastSibling)`

@MATCH	.	.

@MAX	Max	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)`
		MDX
		`Max(MemberRange([Jan], [Mar]))`

Table 4-23 (Cont.) Calculator to MDX Function Mapping

Calculator	MDX	Remarks/Examples

@MAXRANGE	Max	**Calculator**
		
		`@MAXRANGE(Sales, @CHILDREN(Qtr1))`
		
		MDX
		
		`Max( CrossJoin( {Sales}, {[Qtr1].Children} ) )`
		
		OR
		
		`Max([Qtr1].Children, Sales)`

@MAXS	Max	**Calculator**
		
		`@MAXS(SKIPMISSING,Sales,@CHILDREN(Qtr1))`
		
		MDX
		
		`Max( Children([Qtr1]),Sales) )`
		
		For SKIPZERO, the translation is:
		
		`Max (Children ([Qtr1]), IIF (Sales = 0, MISSING, Sales))`
		
		For SKIPBOTH, the translation is the same as for SKIPZERO, because Max skips missing values by default.

Table 4-23 (Cont.) Calculator to MDX Function Mapping

Calculator	MDX	Remarks/Examples

@MAXSRANGE	Max	**Calculator**
		
		`@MAXSRANGE(SKIPMISSING, Sales, @CHILDREN(Qtr1))`
		
		MDX
		
		`Max( Children([Qtr1]),Sales) )`
		
		For SKIPZERO, the translation is:
		
		`Max (Children ([Qtr1]), IIF (Sales = 0, MISSING, Sales))`
		
		For SKIPBOTH, the translation is the same as for SKIPZERO, because Max skips missing values by default.

@MDANCESTVAL	Use Ancestor, Value, and Currentmember as shown in the example.	**Calculator**
		
		`@MDANCESTVAL(2, Market, 2, Product, 2, Sales)`
		
		MDX
		
		Construct a tuple consisting of Sales from the Measures dimension, the ancestor of the current member along the Market dimension, and the ancestor of the current member along the Product dimension. Then get the value of the tuple.
		
		`(Sales, Ancestor(Market.CurrentMember, 2), Ancestor(Product.CurrentMember, 2)).Value`

Table 4-23 (Cont.) Calculator to MDX Function Mapping

Calculator	MDX	Remarks/Examples

@MDPARENTVAL	Use Parent, Value, and CurrentMember as shown in the example.	**Calculator**
		`@MDPARENTVAL(2, Market, Product, Sales)`
		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`

@MDSHIFT	See MDX equivalent for @NEXT, and repeat it for each dimension that needs to be shifted. CrossJoin the results from each dimension and get the value of the final tuple. See comments for @MDANCESTVAL.	.
@MEDIAN	Not supported in MDX.	.
@MEMBER	Not needed in MDX.	.

@MERGE	Union(*set1*,*set2*)	If the lists specified as inputs to @MERGE do not subscribe to the rules of an MDX set, then the @MERGE function cannot be translated. The following example assumes that the lists do subscribe to MDX set rules.
		Calculator
		`@MERGE(@CHILDREN(East),@CHILDREN(West))`
		MDX
		`{Union([East].Children, [West].Children)}`

Table 4-23 (Cont.) Calculator to MDX Function Mapping

Calculator	MDX	Remarks/Examples

@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} ) )`
		OR
		`Min([Qtr1].Children, Sales)`

Table 4-23 (Cont.) Calculator to MDX Function Mapping

Calculator	MDX	Remarks/Examples

@MINS	Min	**Calculator**
		
		`@MINS(SKIPMISSING,Sales,@CHILDREN(Qtr1))`
		
		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 4-23 (Cont.) Calculator to MDX Function Mapping

Calculator	MDX	Remarks/Examples

@MINSRANGE	Min	**Calculator**
		
		`@MINSRANGE(SKIPMISSING, Sales, @CHILDREN(Qtr1))`
		
		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 ) )`

@MOD	Mod	.
@MODE	Not supported in MDX.	.
@NAME	Not needed in MDX.	.

Table 4-23 (Cont.) Calculator to MDX Function Mapping

Calculator	MDX	Remarks/Examples

@NEXT	@NEXT(*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 NextMember (if *n*=1) or Lead/Lag will work as shown in the sample translation. This is probably the common case.
		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**
		`@PARENTVAL(Market, Sales)`
		MDX
		`([Sales], [Market].CurrentMember.Parent).Value`

@POWER	Power	.

Table 4-23 (Cont.) Calculator to MDX Function Mapping

Calculator	MDX	Remarks/Examples

@PRIOR	@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.
		Calculator
		`@Prior(Cash)`
		MDX
		`PrevMember(Year.CurrentMember, LEVEL), [Cash]).Value`
		Alternative:
		Calculator
		`@Prior(Cash, 2)`
		MDX
		`(Year.CurrentMember.Lag(2, LEVEL), [Cash]).Value`

@PRIORS	Not supported in MDX.	.

@RANGE	CrossJoin(*member*, *rangeset*)	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.
		Calculator
		`@RANGE(Sales, @CHILDREN(East))`
		MDX
		`CrossJoin({Sales}, {[East].Children})`

@RANK	Not supported in MDX. This is a vector function.	.
@REMAINDER	Remainder	.

Table 4-23 (Cont.) Calculator to MDX Function Mapping

Calculator	MDX	Remarks/Examples

@REMOVE	Except(*set1*, *set2*)	Translation will work only if *set1* and *set2* are true MDX sets.
		Calculator
		`@REMOVE(@CHILDREN(East),@LIST("New York",Connecticut))`
		MDX
		`Except ({[East].Children}, {[New York],[Connecticut]})`

@ROUND	Round	.
@SHIFT	See @PRIOR and @NEXT.	.
@SIBLINGS	Siblings	.
@STDEV, @STDEVP, @STDEVRANGE	Not supported in MDX.	.
@SUBSTRING	Not supported in MDX.	.
@SUM	Sum	Convert each element of the *explist* to a tuple so that collectively the tuples can form a set.

@SUMRANGE	Sum(CrossJoin(*member*, *Xrangelist*))	**Calculator**
		`@SUMRANGE("New York",Jan:Jun)`
		MDX
		`Sum(CrossJoin({[New York]}, {[Jan]:[Jun]}))`

@TODATE	Todate	.
@TRUNCATE	Truncate	.
@UDA	Uda	.
@VAR, @VARPER	*Arg1 - Arg2*	An aggregate storage outline has no expense tags. Therefore, variance functionality defaults to subtraction.
@VARIANCE, @VARIANCEP	Not supported in MDX.	.
@WITHATTR	WithAttr	.
@XRANGE	Not supported in MDX.	.
@XREF	Not supported in MDX.	.

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 4-24 MDX Member Functions

Function	Result
Ancestor	Returns a member that is an ancestor of the specified member, at a specified generation or level.
ClosingPeriod	Returns the last descendant of a layer, or the last child of the Time dimension.
Cousin	Returns a child member at a matching outline level and location as a member from another parent.
CurrentAxisMember	Returns the current axis member in the context of a member value expression argument.
CurrentMember	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.
DateToMember	Returns the date-hierarchy member specified by the input date.
DefaultMember	Returns the default member in the input dimension.
FirstChild	Returns the first child of the input member.
FirstSibling	Returns the first child of the input member's parent.
Lag	Using the default order of members in a database outline, returns a member that is *n* steps behind the input member.
LastChild	Returns the last child of the input member.

Table 4-24 (Cont.) MDX Member Functions

Function	Result
LastSibling	Returns the last child of the input member's parent.
Lead	Using the default order of members in a database outline, returns a member that is *n* steps past the input member.
NextMember	Returns the member (in the same layer) that is one step past the input member.
OpeningPeriod	Returns the first descendant of a layer, or the first child of the Time dimension.
ParallelPeriod	Returns a member from a prior time period as the specified or default time member.
Parent	Returns a member's parent.
PrevMember	Returns the member (in the same layer) that is one step prior to the input member.
StrToMbr	Converts a string to a member name.

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 4-25 MDX Pure Set Functions

Function	Result
CrossJoin	Returns a cross-section of two sets from different dimensions.
Distinct	Deletes duplicate tuples from a set.
Except	Returns a subset containing the differences between two sets.
Generate	For each tuple in *set1*, return *set2*.
Head	Returns the first *n* members or tuples present in a set.
Intersect	Returns the intersection of two input sets.
Subset	Returns a subset from a set, in which the subset is a numerically specified range of tuples.
Tail	Returns the last *n* members or tuples present in a set.

Table 4-25 (Cont.) MDX Pure Set Functions

Function	Result
TupleRange	Returns the range of tuples between (and inclusive of) two tuples at the same level.
Union	Returns the union of two input sets.

Metadata-based Set Functions

Functions in this category derive their results using metadata information from the cube.

Table 4-26 MDX Metadata-based Set Functions

Function	Result
Ancestors	Returns a set of ancestors up to a specified layer or distance.
Attribute	Returns all base members that are associated with the specified attribute member.
Children	Returns all child members of the input member.
Descendants	Returns the set of descendants of a member at specified layers.
DrilldownByLayer	Drills down members of a set that are at a specified layer.
DrilldownMember	Drills down on any members or tuples of <set1> that are also found in <set2>.
DrillupByLayer	Drills up the members of a set that are below a specified layer.
DrillupMember	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.
Extract	Returns a subset containing only the tuples of a specified dimensionality.
Hierarchize	Sorts members according to the default member ordering as represented in the database outline.
LastPeriods	Returns a set of members ending either at the specified member or at the current member in the time dimension.
MemberRange	Returns the range of members positioned between two input members (inclusive) at the same generation or level.
Members	Returns a set of all members of a given dimension, hierarchy, or layer.
PeriodsToDate	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.

Table 4-26 (Cont.) MDX Metadata-based Set Functions

Function	Result
RelMemberRange	Returns a set based on the relative position of the specified member.
Siblings	Returns the siblings of the input member.
Uda	Returns all members that share a specified user-defined attribute.
WithAttr	Returns all base members that are associated with an attribute member of the specified type.
AttributeEx	Given the varying attribute member and the perspective setting, returns the associated base member list.
WithAttrEx	Given the varying attribute dimension, condition, predicate, and perspective setting, returns the base member list satisfying the predicate.
xTD	Functions returning period-to-date values.

Data-based Set Functions

Functions in this category derive their results using data values from the cube.

Table 4-27 MDX Data-based Set Functions

Function	Result
BottomCount	Returns a set of *n* elements ordered from smallest to largest, optionally based on an evaluation.
BottomPercent	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.
BottomSum	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.
Case	Performs conditional expressions.
Filter	Returns those parts of a set which meet the criteria of a search condition.
IIF	Performs a conditional test, and returns an appropriate numeric expression or set depending on whether the test evaluates to true or false.
Leaves	Returns the set of level 0 (leaf) members that contribute to the value of the specified member.
Order	Sorts members of a set in order based on an expression.
TopCount	Returns a set of *n* elements ordered from largest to smallest, optionally based on an evaluation.

Table 4-27 (Cont.) MDX Data-based Set Functions

Function	Result
TopPercent	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.
TopSum	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.

MDX Functions that Return a Tuple

The following functions return a tuple.

Table 4-28 MDX Tuple Functions

Function	Result
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.
Item	Extracts a member from a tuple.

MDX Functions that Return a Number

The following functions return a value.

Table 4-29 MDX Numeric Value Functions

Function	Result
Abs	Returns absolute value of an expression.
Aggregate	Aggregates the Accounts member based on its Time Balance behavior.
Avg	Returns the average of values found in the tuples of a set.
Case	Performs conditional expressions.
CellValue	Returns the numeric value of the current cell.
CoalesceEmpty	Returns the first non #Missing value from the given value expressions.
Count	Returns the count of the number of tuples in a set.
DateDiff	Returns the difference between two input dates.
DatePart	Returns a number representing a date part (such as Week).
EnumText	Returns the text value corresponding to a numeric value in a text list.

Table 4-29 (Cont.) MDX Numeric Value Functions

Function	Result
EnumValue	Returns the internal numeric value for a text value in a text list.
Exp	Returns the exponent of an expression.
Factorial	Returns the factorial of an expression.
IIF	Performs a conditional test, and returns an appropriate numeric expression or set depending on whether the test evaluates to true or false.
InStr	Returns a number specifying the position of the first occurrence of one string within another.
Int	Returns the next lowest integer value of an expression.
Len	Returns length of a string.
Ln	Returns the natural logarithm of an expression.
Log	Returns the logarithm of an expression to a specified base.
Log10	Returns the base-10 logarithm of an expression.
Max	Returns the maximum of values found in the tuples of a set.
Median	Returns the value of the median tuple of a set.
Min	Returns the minimum of values found in the tuples of a set.
Mod	Returns the modulus (remainder value) of a division operation.
NonEmptyCount	Returns the count of the number of tuples in a set that evaluate to nonempty values.
NTile	Returns a division number of a tuple in a set.
Ordinal	Returns a number indicating depth in the hierarchy.
Percentile	Returns the value of the tuple that is at a given percentile of a set.
Power	Returns the value of the numeric value expression raised to *power*.
Rank	Returns the numeric position of a tuple in a set.
RealValue	Returns a value for the specified member or tuple without the inherited attribute dimension context.
Remainder	Returns the remainder value of the numeric value expression.
Round	Rounds a numeric value expression to the specified number of digits.
Stddev	Calculates standard deviation based on a sample.
Stddevp	Calculates standard deviation based on a population.

Table 4-29 (Cont.) MDX Numeric Value Functions

Function	Result
StrToNum	Converts a string to a number.
Sum	Returns the sum of values of tuples in a set.
Todate	Converts a date string to a value that is usable in calculations.
Truncate	Removes the fractional part of a numeric value expression, returning the integer.

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 4-30 MDX Layer Functions

Function	Result
Generation	Returns the generation of the input member.
Generations	Returns the generation specified by the input numerical depth and the input dimension or hierarchy.
Level	Returns the level of the input member.
Levels	Returns the level specified by the input numerical depth and the input dimension or hierarchy.

MDX Functions that Return a Boolean

The following functions return a Boolean (TRUE or FALSE).

Table 4-31 MDX Boolean Functions

Function	Result
Is	Returns TRUE if two members are identical.
IsAccType	Returns TRUE if the current member has the associated accounts tag.
IsAncestor	Returns TRUE if the first member is an ancestor of the second member.
IsChild	Returns TRUE if the first member is a child of the second member.
IsEmpty	Returns True if the value of an input numeric-value-expression is #MISSING.

Table 4-31 (Cont.) MDX Boolean Functions

Function	Result
IsGeneration	Returns TRUE if the member is in a specified generation.
IsLeaf	Returns TRUE if the member is a level-0 member.
IsLevel	Returns TRUE if the member is in a specified level.
IsSibling	Returns TRUE if the first member is a sibling of the second member.
IsUda	Returns TRUE if the member has the associated UDA tag (user-defined attribute).
IsValid	Returns TRUE if the specified element validates successfuly.
Contains	Returns TRUE if a tuple is found within a set.

MDX Functions that Return a Date

The following functions return a date.

Table 4-32 MDX Date Functions

Function	Result
DateRoll	To the given date, rolls (adds or subtracts) a number of specific time intervals, returning another date.
GetFirstDate	Returns the start date for a date-hierarchy member.
GetLastDate	Returns the end date for a date-hierarchy member.
GetNextDay	To the given date and the week day, gets the next date after input date that corresponds to the week day.
GetFirstDay	For a given date_part, returns the first day of the time interval for the input date.
GetLastDay	For a given date_part, returns the last day of the time interval for the input date.
TodateEx	Converts date strings to dates.
Today	Returns a number representing the current date.
JulianDate	For the given UNIX date, gets its Julian date.
UnixDate	For the given Julian date, gets its UNIX date.

MDX Functions that Return a String

The following functions return a string.

Table 4-33 MDX String Functions

Function	Result
FormatDate	Formats date strings.
Concat	Concatenates input strings.
Left	Returns a specified number of characters from the left side of the string.
Right	Returns a specified number of characters from the right side of the string.
LTrim	Trims whitespace on the left of the string.
RTrim	Trims whitespace on the right of the string.
Lower	Converts upper-case string to lower case.
Upper	Converts lower-case string to upper case.
Substring	Returns the substring between a starting and ending position.
NumToStr	Converts a double-precision floating-point value into a decimal string.

MDX Function List

Consult the Contents pane for a list of MDX functions by return value.

Table 4-34 MDX Function List

Alphabetical List of MDX Functions		
Abs	Generations	Min
Aggregate	GetFirstDate	Mod
Ancestor	GetFirstDay	NextMember
Ancestors	GetLastDate	NonEmptyCount
Attribute	GetLastDay	NonEmptySubset
AttributeEx	GetNextDay	NTile
Avg	GetRoundDate	NumToStr
BottomCount	Head	OpeningPeriod
BottomPercent	Hierarchize	Order
BottomSum	IIF	Ordinal
Case	InStr	ParallelPeriod
CellValue	InString	Parent
Children	Int	Percentile
ClosingPeriod	Intersect	PeriodsToDate
CoalesceEmpty	Is	Power
Concat	IsAccType	PrevMember
Contains	IsAncestor	Rank
Count	IsChild	RealValue
Cousin	IsEmpty	RelMemberRange
CrossJoin	IsGeneration	Remainder
CrossJoinAttribute	IsLeaf	Right
CurrentAxisMember	IsLevel	Round

Table 4-34 (Cont.) MDX Function List

Alphabetical List of MDX Functions		
CurrentMember	IsMatch	RTrim
CurrentTuple	IsSibling	Siblings
DateDiff	IsUda	Stddev
DatePart	IsValid	Stddevp
DateRoll	Item	StrToMbr
DateToMember	JulianDate	StrToNum
DefaultMember	Lag	Subset
Descendants	LastChild	Substring
Distinct	LastPeriods	Sum
Dimension	LastSibling	Tail
DrilldownByLayer	Lead	Todate
DrilldownMember	Leaves	TodateEx
DrillupByLayer	Left	Today
DrillupMember	Len	TopCount
DTS	Level	TopPercent
EnumText	Levels	TopSum
EnumValue	LinkMember	Truncate
Except	Ln	TupleRange
Exp	Log	Uda
Extract	Log10	Union
Factorial	Lower	UnixDate
Filter	LTrim	Upper
FirstChild	Max	Value
FirstSibling	Median	WithAttr
FormatDate	MemberRange	WithAttrEx
Generate	Members	xTD
Generation		

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:

```
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 4-35 Output Grid from MDX Example

(axis)	Jan	Feb	Mar
Actual	1323	1290	1234
Budget	1200	1100	1100
Variance	123	190	134
Absolute Variance	123	190	134

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]}'
 SET [GM] AS '{Children ( [Geography].[South] )}'
 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 4-36 Output Grid from MDX Example

(axis)	m1
(1st Half, DISTRICT OF COLUMBIA)	961107.26
(1st Half, DELAWARE)	245394.68
(1st Half, FLORIDA)	1446868.96
(1st Half, GEORGIA)	4766285.74
(1st Half, MARYLAND)	2496467.86
(1st Half, NORTH CAROLINA)	4660670.94
(1st Half, SOUTH CAROLINA)	2524777.6
(1st Half, VIRGINIA)	6253779.5
(1st Half, WEST VIRGINIA)	5009523.72

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.

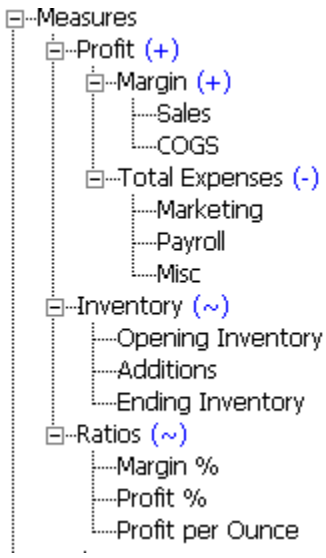
index
A number of hierarchical steps up from *member*, locating the ancestor you want returned.

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:



  - [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 `{[Market], [East], [New York]}`, the self-inclusive set of `[New York]` ancestors beginning with the ancestor that is located at level 2 of the Market dimension.

```
Ancestors ( [Feb], 1 )
```

returns `{[Qtr1],[Feb]}`, the self-inclusive set of ancestors beginning with the ancestor one step higher than `Feb`.

```
Ancestors ( [Feb], 0 )
```

returns {[Feb]}.

Using the ASOSamp.Sample database,

```
Ancestors ([94089], [Geography].generations(2))
```

returns `{[West], [CA], [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

```
Attribute ( member )
```

Parameters

member
Specification of a member from an attribute dimension.

Example

The following query

```
SELECT
 {[Year].Children}
ON COLUMNS,
 Attribute ([Ounces_12])
ON ROWS
FROM Sample.Basic
```

returns the grid:

Table 4-37 Output Grid from MDX Example

(axis)	Qtr1	Qtr2	Qtr3	Qtr4
Cola	5096	5892	6583	5206
Diet Cola	1359	1534	1528	1287
Old Fashioned	1697	1734	1883	1887
Sarsaparilla	1153	1231	1159	1093
Diet Cream	2695	2723	2855	2820

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
 'Avg ( [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 4-38 Output Grid from MDX Example

(axis)	Cola	Diet Cola	Caffeine Free Cola
California	678	118	145
Oregon	160	140	150
Washington	130	190	#Missing
Utah	130	190	170
Nevada	76	62	#Missing
Western Avg	234.8	140	93

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
 'Avg ( [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 4-39 Output Grid from MDX Example

(axis)	Cola	Diet Cola	Caffeine Free Cola
California	678	118	145
Oregon	160	140	150
Washington	130	190	#Missing
Utah	130	190	170
Nevada	76	62	#Missing
Western Avg	234.8	140	155

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

```
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 4-40 Output Grid from MDX Example

(axis)	Qtr1	Qtr2	Qtr3	Qtr4
200–40	2807	2922	2756	3265
100–30	3187	3182	3189	3283
400–30	3763	3962	3995	4041

Table 4-40 (Cont.) Output Grid from MDX Example

(axis)	Qtr1	Qtr2	Qtr3	Qtr4
300–20	4248	4638	4556	4038
200–30	4440	4562	4362	4195
100–20	7276	7957	8057	7179
100–20	7276	7957	8057	7179
400–20	7771	8332	8557	8010
400–10	8614	9061	9527	8957
300–30	8969	9105	9553	9342

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

```
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 4-41 Output Grid from MDX Example

(axis)	Apr	May	Jun
Birch Beer	954	917	1051
Caffeine Free Cola	1049	1065	1068
Strawberry	1314	1332	1316
Sarsaparilla	1509	1552	1501
Vanilla Cream	1493	1533	1612
Sum of all lowest prods	6319	6399	6548
Product	32917	33674	35088
Percent that lowest sellers hold of all product sales	0.192	0.194	0.187

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,
 {
  BottomSum(
  [Product].Members, 10000, [Year].[Qtr1]
  )
 }
ON ROWS
FROM Sample.Basic
WHERE ([Measures].[Sales])
```

This query returns the grid:

Table 4-42 Output Grid from MDX Example

(axis)	Qtr1	Qtr2
200-40	2807	2922
100-30	3187	3182
400-30	3763	3962
300-20	4248	4638

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...
[ else_clause ]
END
```

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 4-43 Output Grid from MDX Example

(axis)	ProductOunces
Product	0
Colas	0
Cola	12
Diet Cola	12
Caffeine Free Cola	16
Root Beer	0
Old Fashioned	12
Diet Root Beer	16
Sarsaparilla	12
Birch Beer	16
Cream Soda	0
Dark Cream	20

Table 4-43 (Cont.) Output Grid from MDX Example

(axis)	ProductOunces
Vanilla Cream	20
Diet Cream	12
Fruit Soda	0
Grape	32
Orange	32
Strawberry	32
Diet Drinks	0
Diet Cola	0
Diet Root Beer	0
Diet Cream	0

Example for Searched Case Expression

The following query divides products into different profit categories based on Profit, and returns categories for each product.

```
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 4-44 Output Grid from MDX Example

(axis)	ProfitCategory
Product	4
Colas	4
Cola	4
Diet Cola	3
Caffeine Free Cola	1
Root Beer	4
Old Fashioned	3
Diet Root Beer	4
Sarsaparilla	2
Birch Beer	2
Cream Soda	4
Dark Cream	4

Table 4-44 (Cont.) Output Grid from MDX Example

(axis)	ProfitCategory
Vanilla Cream	1
Diet Cream	4
Fruit Soda	4
Grape	4
Orange	3
Strawberry	1
Diet Drinks	4
Diet Cola	3
Diet Root Beer	4
Diet Cream	4

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.

```
/* 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.

```
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() <= 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

```
member.Children
```

```
Children ( member )
```

Parameters

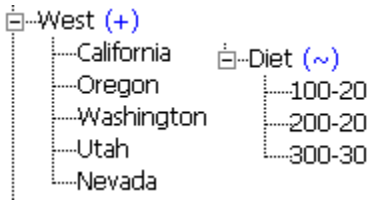
member
A member specification.

Notes

If the input member does not have any children (is a level-0 member), this function returns an emtpy set.

Example

This example uses the following parts of the Sample Basic outline:



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 4-45 Output Grid from MDX Example

(axis)	California	Oregon	Washington	Utah	Nevada
100-20	-1587	338	231	398	86
200-20	2685	1086	579	496	167
300-30	1328	288	1217	413	362

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 4-46 Output Grid from MDX Example

(axis)	100-10	100-10
(axis)	**Starting Inventory**	**Closing Inventory**
Jan	14587	14039
Feb	14039	13566
Mar	13566	13660
Qtr1	14587	13660
Apr	13660	14172

Table 4-46 (Cont.) Output Grid from MDX Example

(axis)	100-10	100-10
(axis)	Starting Inventory	Closing Inventory
May	14172	15127
Jun	15127	15580
Qtr2	13660	15580
Jul	15580	14819
Aug	14819	14055
Sep	14055	13424
Qtr3	15580	13424
Oct	13424	13323
Nov	13323	13460
Dec	13460	12915
Qtr4	13424	12915
Year	14587	12915

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
 Sample.Basic
```

```
WHERE
 ([Year].[Jan], [Market].[New York])
```

returns the grid:

Table 4-47 Output Grid from MDX Example

(axis)		Actual	Budget
Sales	**Colas**	678	640
	Root Beer	551	530
	Cream Soda	663	510
	Fruit Soda	587	620
	Diet Drinks	#Missing	#Missing
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]`.

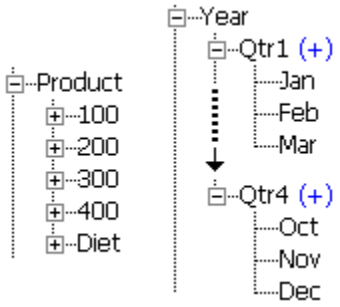
member2
An ancestor for which Cousin() should the return child member at the same position as *member1*.

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 4-48 Output Grid from MDX Example

(axis)	Oct
100	2317
200	2505
300	2041
400	1790
Diet	2379

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], [New York]), ([Qtr1], [California]),
 ([Qtr2], [New York]), ([Qtr2], [California])}
```

Therefore, the following query

```
SELECT
CrossJoin({[Qtr1], [Qtr2]}, {[New York], [California]})
ON COLUMNS
FROM sample.basic
```

returns the grid:

Table 4-49 Output Grid from MDX Example

Qtr1	Qtr1	Qtr2	Qtr2
New York	**California**	**New York**	**California**
1656	3129	2363	3288

Example 2

The following expression

```
CrossJoin({[Qtr1], [Qtr2], [Qtr3]}, {[New York], [California], [Texas]})
```

returns the set

```
{([Qtr1], [New York]), ([Qtr1], [California]), ([Qtr1], [Texas]),
([Qtr2], [New York]), ([Qtr2], [California]), ([Qtr2], [Texas]),
([Qtr3], [New York]), ([Qtr3], [California]), ([Qtr3], [Texas])}
```

Therefore, the following query

```
SELECT
CrossJoin({[Qtr1], [Qtr2], [Qtr3]}, {[New York], [California], [Texas]})
ON AXIS(0)
FROM Sample.Basic
```

returns the grid:

Table 4-50 Output Grid from MDX Example

Qtr1	Qtr1	Qtr1	Qtr2	Qtr2	Qtr2	Qtr3	Qtr3	Qtr3
New York	**California**	**Texas**	**New York**	**California**	**Texas**	**New York**	**California**	**Texas**
1656	3129	1582	2363	3288	1610	1943	3593	1703

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 4-51 Output Grid from MDX Example

(axis)	(axis)	East	West	South	Central
100–10	**Margin**	15762	8803	5937	8124
	Total Expenses	4633	4210	2361	4645
100–20	**Margin**	1785	3707	2767	7426
	Total Expenses	671	4241	1570	3495
100–30	**Margin**	871	1629	#Missing	3975
	Total Expenses	458	2139	#Missing	1895

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:

```
SELECT {} ON COLUMNS,
CrossJoinAttribute ({set},{set}) ON ROWS
FROM <cube_specification>
```

Example

The following query based on ASOSamp.Sample

```
SELECT
{} ON COLUMNS,
CrossJoinAttribute({[Great Buys].Children}, {[Square
Footage].Children} ) ON ROWS
FROM ASOSamp.Sample;
```

returns the grid

Table 4-52 Output Grid from MDX Example

(axis)
(004118, 10000)
(011683, 5000)
(017589, 10000)

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

```
dimension.CurrentMember


CurrentMember ( dimension )
```

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.

```
SELECT
Filter (
  [Year].Children,  -- outer loop
  Max (
      Except (
        [Year].CurrentMember.Children, -- current in outer loop
        { [Year].[Jan] }
      ),
      (  [Year].CurrentMember         -- current in Max loop
       /  [Year].CurrentMember.PrevMember)
  ) >= 1.03
)
ON axis(0)
FROM Sample.Basic
WHERE ([Measures].[Sales])
```

Returns the grid:

Table 4-53 Output Grid from MDX Example

Qtr2	Qtr4
101679	98141

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]
AS 'CrossJoin([Product].Children, [Market].Children)'
SELECT
   Filter([NewSet], NOT IsEmpty([NewSet].CurrentTuple))
ON COLUMNS
FROM Sample.Basic
WHERE
   {[Sales]}
```

This query returns the following grid:

Table 4-54 Output Grid from MDX Example

100				200	...	400	Diet			
East	West	South	Central	East	...	Central	East	West	South	Central
27740	28306	16280	33808	23672	...	33451	7919	36423	18676	42660

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.

```
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

```
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

DefaultMember (*dimension*)

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.

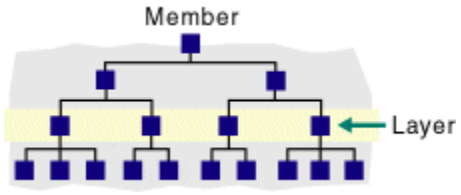
Desc_flags
Optional. Keywords which further indicate which members to return. These keywords are available only if *layer* or *index* is specified.
See Values for Desc_flags

Notes

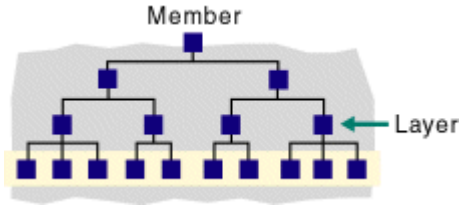
Values for Desc_flags

For all flags, SELF refers to *layer*; therefore, BEFORE indicates "before the layer" and AFTER indicates "after the layer."

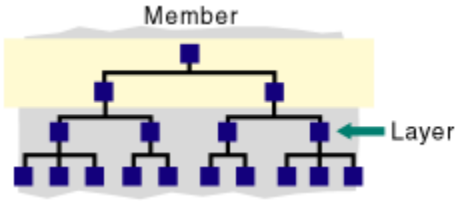
- SELF—Include only members in *layer*, including *member* only if *member* is in layer.



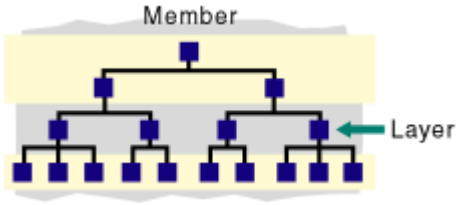
- AFTER—Include members below *layer*, but not the members of *layer*.



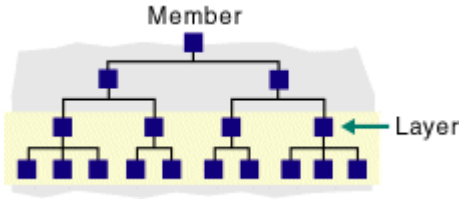
- BEFORE—Include *member* and all its descendants that are higher in the hierarchy than *layer*, excluding *layer* and anything below it.



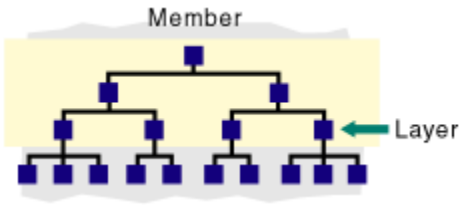
- BEFORE_AND_AFTER—Include *member* and all its descendants, down to level 0, but excluding members in *layer*.



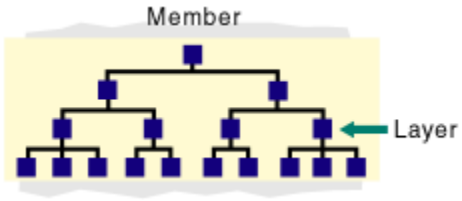
- SELF_AND_AFTER—Include members in *layer* and all descendants below *layer*.



- 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 4-55 Output Grid from MDX Example

Year	Qtr1	Jan	Feb	Mar	Qtr2	Apr	May	Jun	Qtr3	Jul	Aug	Sep	Qtr4	Oct	Nov	Dec
12656	2747	924	888	935	3352	1011	1071	1270	3740	1334	1304	1102	2817	907	884	1026

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:

```
{}
```

```
Descendants([Market], 10, SELF)
```

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 4-56 Output Grid from MDX Example

Product								
California	**Oregon**	**New York**	**South**	**Texas**	**Oklahoma**	**Louisiana**	**New Mexico**	**Washington**
12964	5062	8202	13238	6425	3491	2992	330	4641

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])
 }, 1
 )
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)
```

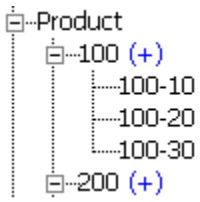
returns the set:

```
{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

```
DrilldownMember
 ( {([100],[California]), ([200],[Washington])},
   { [100] }
 )
```

returns the set of tuples:

```
{ ([100],California), ([100-10],California), ([100-20],California),
 ([100-30],California), ([200],Washington)}
```

Therefore, the following query

```
SELECT
DrilldownMember
 ( {([100],[California]), ([200],[Washington])},
   { [100] }
 )
ON COLUMNS
FROM Sample.Basic
```

returns the grid:

Table 4-57 Output Grid from MDX Example

100	100-10	100-20	100-30	200
California	**California**	**California**	**California**	**Washington**
999	3498	-1587	-912	1091

DrillupByLayer

Drills up the members of a set that are below a specified layer.

Syntax

```
DrillupByLayer ( set [,layer] )
```

Parameters

set
The set in which the drill-up should occur.

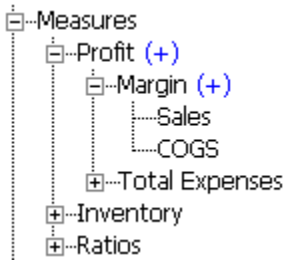
layer
The layer of the members that should be drilled up. If omitted, the set is drilled up to the second lowest level found in the set.

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:

```
SELECT
  DrillupByLayer
  (
   {[Measures],[Profit],
    [Margin], [Sales], [COGS]
   }, Generations([Measures], 2)
  )

ON COLUMNS
FROM Sample.Basic
```

This query returns the grid:

Table 4-58 Output Grid from MDX Example

Measures	Profit
105522	105522

Example 2

With no *layer* specified, the following query drills up the members of *set* to the second lowest level found in *set*:

```
SELECT
  DrillupByLayer
  (
   {[Measures],[Profit],
    [Margin], [Sales], [COGS]
   }
  )

ON COLUMNS
FROM Sample.Basic
```

This query returns the grid:

Table 4-59 Output Grid from MDX Example

Measures	Profit	Margin
105522	105522	221519

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

```
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}
 )
```

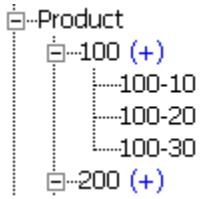
returns the set:

```
{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:



The following expression

```
DrillupMember
 ({Product, [100], [100-10]},
  {[Product]}
)
```

returns the set:

```
{Product}
```

The following expression

```
DrillupMember
 ({Product, [100], [100-10]},
  {[100]}
 )
```

returns the set:

```
{Product, [100]}
```

DTS

Calculates period-to-date values using built-in Dynamic Time Series functionality on block storage databases.

Syntax

`DTS (`*`dts-operation-specification`*`, `*`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 4-60 Output Grid from MDX Example

(axis)	Profit	Opening Inventory	Ratios
Jan	8024	117405	55.1017819772972
Feb	8346	116434	55.3868221647073
Mar	8333	115558	55.2665073107131
Apr	8644	119143	55.4181729805268
QuarterToDate_April	8644	119143	55.4181729805268

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

enum_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 comparet 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 4-61 Output Grid from MDX Example

(axis)	Sales	Profit
Connecticut	3472	920
New Hampshire	1652	202
Oregon	5058	1277
Washington	4835	1212
Utah	4209	744
Nevada	6516	775
Oklahoma	2961	718
Louisiana	2906	773
New Mexico	1741	4
Wisconsin	4073	913
Missouri	3062	399
Iowa	6175	2036

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 4-62 Output Grid from MDX Example

(axis)	Variance %	Index
East	10.700	1.113
West	10.914	1.115
South	3.556	1.036
Central	3.595	1.037

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 4-63 Output Grid from MDX Example

Qtr1	Qtr2
24703	27107

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 4-64 Output Grid from MDX Example

(axis)	Profit
100-10	22777

Table 4-64 (Cont.) Output Grid from MDX Example

(axis)	Profit
100-20	5708
100-30	1983
200-10	7201
200-20	12025
200-30	4636
200-40	4092
300-10	12195
300-20	2511
300-30	11093
400-10	11844
400-20	9851
400-30	-394
100-20	5708
200-20	12025
300-30	11093

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 4-65 Output Grid from MDX Example

(axis)	Profit
400-30	-394

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])
```

```
ON ROWS
FROM Sample.Basic
```

This query returns profit for the members that are caffeinated:

Table 4-66 Output Grid from MDX Example

(axis)	Profit
100-10	22777
100-20	5708
200-10	7201
200-20	12025
300-10	12195
300-20	2511
300-30	11093

To understand the search condition, `Product.CurrentMember.[Caffeinated]`, it may be helpful to read it right to left: Filter is searching for presense 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 evaluted the search condition for each tuple. So to see how the previous query results were generated, it would be useful to see first 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 4-67 Output Grid from MDX Example

(axis)	IsCaffeinated
100-10	1
100-20	1
100-30	0
200-10	1
200-20	1

Table 4-67 (Cont.) Output Grid from MDX Example

(axis)	IsCaffeinated
200-30	0
200-40	0
300-10	1
300-20	1
300-30	1
400-10	0
400-20	0
400-30	0
100-20	0
200-20	0
300–30	0

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 4-68 Output Grid from MDX Example

(axis)	Sales
100-20	2153
400-30	1862
100-20	2153

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 4-69 Output Grid from MDX Example

Cola	Dark Cream
16048	11993

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 4-70 Output Grid from MDX Example

(axis)	Sales
Apr	8685
May	8945
Jun	9557
Jul	9913
Aug	9787
Sep	8844
Dec	8772

FirstChild

Returns the first child of the input member.

Syntax

member.FirstChild

FirstChild (*member*)

Parameters

member
A member specification. If a level-0 member, the output of FirstChild is an empty member.

Example

```
SELECT
  {[Qtr1].firstchild}
ON COLUMNS,
  {[Market].[Central].lastchild}
ON ROWS
FROM Sample.Basic
```

Table 4-71 Output Grid from MDX Example

(axis)	Jan
Colorado	585

See Also

LastChild

FirstSibling

FirstSibling

Returns the first child of the input member's parent.

Syntax

```
FirstSibling ( 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 4-72 Output Grid from MDX Example

(axis)	Inventory Level Since Beginning of Quarter
Jan	-971
Feb	-1847
Mar	1738
Apr	6740
May	17002
Jun	24315
Jul	-871
Aug	-1243
Sep	-1608
Oct	2000
Nov	5308
Dec	4474

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])
          )
         )
  , Sales)'
SELECT
 {[first 10 days sales]}
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 4-73 Output Grid from MDX Example

(axis)	East			West			South			Central		
(axis)	Product	Colas	Root Beer	Product	Diet Drinks	Cream Soda	Product	Root Beer	Diet Drinks	Product	Diet Drinks	Colas
Qtr1	20621	6292	5726	31674	8820	8043	12113	5354	4483	31412	10544	8074
Qtr2	224499	7230	5902	33572	9086	8982	12602	5535	4976	33056	10809	8701
Qtr3	22976	7770	5863	35130	9518	9616	13355	5690	4947	33754	10959	8894
Qtr4	21352	6448	6181	32555	8999	8750	12776	5429	4450	31458	10348	8139

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 4-74 Output Grid from MDX Example

(axis)	Qtr1	Qtr2	Qtr3	Qtr4
100	7048	7872	8511	7037
200	6721	7030	7005	7198
300	5929	6769	6698	6403
400	5005	5436	5698	5162
Diet	7017	7336	7532	6941

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

```
SELECT
 [Year].[Qtr1].Generation.Members
ON COLUMNS,
 [Product].Generations(2).Members
ON ROWS
FROM Sample.Basic
```

returns the grid:

Table 4-75 Output Grid from MDX Example

(axis)	Qtr1	Qtr2	Qtr3	Qtr4
100	7048	7872	8511	7037
200	6721	7030	7005	7198
300	5929	6769	6698	6403
400	5005	5436	5698	5162
Diet	7017	7336	7532	6941

See Also

Generation

Levels

GetFirstDate

Returns the start date for a date-time dimension member.

Syntax

```
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

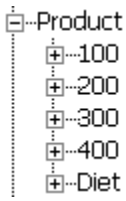
set
The set from which to take items.

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:



The following expression

```
[Product].children
```

returns the set:

```
{ [100], [200], [300], [400], [Diet] }
```

Therefore, the following expression

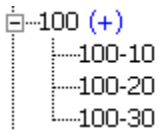
```
 Head (
  [Product].children, 2)
```

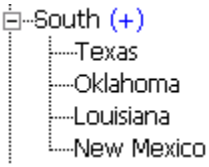
returns the first two members of the previous result set:

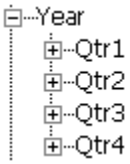
```
{ [100], [200] }
```

Example 2

This example uses the following parts of the Sample Basic outline:







The following expression

```
 CrossJoin ( [100].children, [South].children )
```

returns the set:

```
{ ([100-10], Texas), ([100-10], Oklahoma), ([100-10], Louisiana),
([100-10], [New Mexico]),
  ([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

```
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),
([100-10], [New Mexico]),
  ([100-20], Texas), ([100-20], Oklahoma), ([100-20], Louisiana),
([100-20], [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,
Head (
 CrossJoin (
  [100].children, [South].children), 8
     )
ON ROWS
FROM Sample.Basic
```

returns the grid:

Table 4-76 Output Grid from MDX Example

(axis)		Qtr1	Qtr2	Qtr3	Qtr4
100–10	**Texas**	489	536	653	547
	Oklahoma	87	92	128	211
	Louisiana	93	106	128	137
	New Mexico	76	101	122	70
100–20	**Texas**	206	199	152	82
	Oklahoma	84	66	55	79
	Louisiana	119	158	171	104
	New Mexico	–103	–60	–98	–18

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 4-77 Output Grid from MDX Example

Apr	May	Jun
8644	8929	9534

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 4-78 Output Grid from MDX Example

Qtr2	Apr	May	Jun
27107	8644	8929	9534

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 4-79 Output Grid from MDX Example

Apr	May	Jun	Qtr2
8644	8929	9534	27107

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 4-80 Output Grid from MDX Example

(axis)	Feb	Qtr1	Apr	Jun	Qtr2	Dec	Year
Margin	17762	52943	18242	19457	56317	18435	221519
Sales	32069	95820	32917	35088	101679	33342	400855

IIF

Performs a conditional test, and returns an appropriate numeric expression or set depending on whether the test evaluates to true or false.

Syntax

```
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 *value_expression* or a set. IIF returns this expression if the search condition evaluates to TRUE (something other than zero).
The *value_expression* can be a numeric value expression or a string value expression.

false_part
A *value_expression* or a set. IIF returns this expression if the search condition evaluates to FALSE (zero).
The *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.

```
WITH MEMBER
 [Scenario].[Revised Budget]
AS
 'IIF (
      [Product].CurrentMember.Caffeinated,
      Budget * 1.1, Budget
 )'
SELECT
 {[Scenario].[Budget], [Scenario].[Revised Budget]}
ON COLUMNS,
 [Product].Levels(0).Members
ON ROWS
FROM Sample.Basic
WHERE ([Measures].[Sales], [Year].[Qtr3])
```

This query returns the grid:

Table 4-81 Output Grid from MDX Example

(axis)	Budget	Revised Budget
100-10	18650	20515
100-20	8910	9801
100-30	3370	3370

Table 4-81 (Cont.) Output Grid from MDX Example

(axis)	Budget	Revised Budget
200-10	11060	12166
200-20	9680	10648
200-30	3880	3880
200-40	2660	2660
300-10	10600	11660
300-20	3760	4136
300-30	8280	9108
400-10	7750	7750
400-20	6800	6800
400-30	3290	3290
100-20	8910	8910
200-20	9680	9680
300-30	8280	8280

Example 2

The following query calculates a Revised Budget equaling Budget for caffeinated products, and Actual for non-caffeinated products.

```
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 4-82 Output Grid from MDX Example

(axis)	Budget	Revised Budget
Cola	18650	18650
Diet Cola	8910	8910
Caffeine Free Cola	3370	3189

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 4-83 Output Grid from MDX Example

(axis)	Jan	Feb	Mar
California	38.643	37.984	38.370
Oregon	17.500	16.129	16.107
Washington	29.231	30.986	32.000
Utah	23.077	23.077	20.968
Nevada	-3.947	-6.757	-5.333
West_approx	104.000	101.00	102.000

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*.

ALL
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

```
Intersect({[New York], [California], [Oregon]},
          {[California], [Washington], [Oregon]})
```

returns the set:

```
{[California], [Oregon]}
```

Therefore, the following query

```
SELECT
Intersect({[New York], [California], [Oregon]},
          {[California], [Washington], [Oregon]})
ON COLUMNS
FROM Sample.Basic
```

returns the grid:

Table 4-84 Output Grid from MDX Example

California	Oregon
12964	5062

Example 2

The following expression

```
Intersect( { [New York], [California], [Florida], [California] },
           { [Oregon], [Washington], [California], [Florida],
[California] }, ALL)
```

returns the set:

```
{ [California], [Florida], [California] }
```

Therefore, the following query

```
SELECT
Intersect( { [New York], [California], [Florida], [California] },
           { [Oregon], [Washington], [California], [Florida],
[California] }, ALL)
ON COLUMNS
FROM Sample.Basic
```

returns the grid:

Table 4-85 Output Grid from MDX Example

California	Florida	California
12964	5029	12964

The matching duplicate element [California] is duplicated in the result.

However, the following expression

```
Intersect( { [New York], [California], [Florida], [California] },
           { [Oregon], [Washington], [California], [Florida] }, ALL)
```

would return only

```
{ [California], [Florida] }
```

because only one match exists between `[California]` in set1 and `[California]` in set2.

Is

Returns TRUE if two members are identical.

Syntax

```
IS ( member1 , member2 )
```

```
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 4-86 Output Grid from MDX Example

New York	Massachusetts	Florida	Connecticut	New Hampshire
8202	6712	5029	3093	1125

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

```
SELECT
Filter([Measures].Members, IsAccType([Measures].CurrentMember, First))
ON COLUMNS
FROM Sample.Basic
```

This query returns the following grid:

Table 4-87 Output Grid from MDX Example

Opening Inventory
117405

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 4-88 Output Grid from MDX Example

Market	East
105522	24161

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 4-89 Output Grid from MDX Example

Market	East	Florida
105522	24161	5029

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.

```
SELECT
 Filter([Market].Members, IsChild([Market].CurrentMember, [East]))
ON COLUMNS
FROM Sample.Basic
```

Table 4-90 Output Grid from MDX Example

New York	Massachusetts	Florida	Connecticut	New Hampshire
8202	6712	5029	3093	1125

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.

```
SELECT
 Filter([Market].Members, IsChild([Market].CurrentMember, [East],
INCLUDEMEMBER))
ON COLUMNS
FROM Sample.Basic
```

Table 4-91 Output Grid from MDX Example

East	New York	Massachusetts	Florida	Connecticut	New Hampshire
24161	8202	6712	5029	3093	1125

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.

```
WITH SET [NewSet]
AS 'CrossJoin([Product].Children, [Market].Children)'
SELECT
   Filter([NewSet], NOT IsEmpty([NewSet].CurrentTuple))
ON COLUMNS
FROM Sample.Basic
WHERE
   {[Sales]}
```

This query returns the following grid:

Table 4-92 Output Grid from MDX Example

	100			...	400			Diet			
East	West	South	Central	...	East	West	Central	East	West	South	Central
27740	28306	16280	33808	...	15745	35034	33451	7919	36423	18676	42660

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 4-93 Output Grid from MDX Example

East	West	South	Central
24161	29861	13238	38262

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 4-94 Output Grid from MDX Example

New York	Massachusetts	Florida	...	Missouri	Iowa	Colorado
8202	6712	5029	...	1466	9061	7227

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 4-95 Output Grid from MDX Example

East	West	South	Central
24161	29861	13238	38262

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":

```
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":

```
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 "*":

```
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":

```
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

```
IsSibling( member1, member2 [, INCLUDEMEMBER])
```

Parameters

member1
A member specification.

member2
A member specification.

INCLUDEMEMBER
Optional. Use this keyword if you want 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 `IsSibling([Market].CurrentMember, [California])` returns TRUE; in other words, the query returns all states that are siblings of California.

```
SELECT
 Filter([Market].Members, IsSibling([Market].CurrentMember,
[California]))
ON COLUMNS
FROM Sample.Basic
```

Table 4-96 Output Grid from MDX Example

Oregon	Washington	Utah	Nevada
5062	4641	3155	4039

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 `IsSibling([Market].CurrentMember, [California])` returns TRUE; in other words, the query returns all states that are siblings of California, including California itself.

```
SELECT
 Filter([Market].Members, IsSibling([Market].CurrentMember,
[California], INCLUDEMEMBER))
ON COLUMNS
FROM Sample.Basic
```

Table 4-97 Output Grid from MDX Example

California	Oregon	Washington	Utah	Nevada
12964	5062	4641	3155	4039

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 4-98 Output Grid from MDX Example

East	New York	Massachusetts	Florida	California	Texas	Central	Illinois	Ohio	Colorado
24161	8202	6712	5029	12964	6425	38262	12577	4384	7227

IsValid

Returns TRUE if the specified element validates successfuly.

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 4-99 Output Grid from MDX Example

100-10	100-20	200-10	200-30	300-30
22777	5708	7201	4636	11093

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] ( index )
```

```
Item ( tuple, index )
```

Syntax that Returns a Tuple—one of the following:

```
set[.Item] ( index )
```

```
Item ( 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 *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 *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

```
SELECT
{( [Qtr1], [Sales], [Cola], [Florida], [Actual] ).Item(3)}
ON COLUMNS
FROM Sample.Basic
```

returns:

Table 4-100 Output Grid from MDX Example

Florida
5029

```
SELECT
 {Item(( [Qtr1], [Sales], [Cola], [Florida], [Actual] ), 2)}
ON COLUMNS
FROM Sample.Basic
```

returns:

Table 4-101 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, (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
 '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

```
member.Lag (index [,layertype ] [, hierarchy ] )
```

```
Lag ( 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, `[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:

```
[Jun].lag (3)
```

returns the member that is 3 steps prior to Jun:

```
[Mar]
```

The following expression:

```
[Jun].lag (-3)
```

returns the member that is 3 steps following Jun:

```
[Sep]
```

For every month, the following query displays the sales and average over the last three months.

```
WITH MEMBER
 [Measures].[Average Sales in Last 3 months]
AS
'Avg(
     {[Year].CurrentMember,
      [Year].CurrentMember.Lag(1),
      [Year]. CurrentMember.Lag(2)
     },
     [Measures].[Sales]
 )'
SELECT
 {[Measures].[Sales],
  [Measures].[Average Sales in Last 3 months]
 }
ON COLUMNS,
 [Year].Levels(0).Members
ON ROWS
FROM Sample.Basic
```

This query returns the grid:

Table 4-102 Output Grid from MDX Example

(axis)	Sales	Average Sales in Last 3 Months
Jan	31538	31538
Feb	23069	31803.500
March	32213	31940
April	32917	32399.667
May	33674	32934.667
Jun	35088	33893
Jul	36134	34965.333
Aug	36008	35743.333
Sep	33073	35071.667
Oct	32828	33969.667
Nov	31971	32624
Dec	33342	32713.667

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 4-103 Output Grid from MDX Example

(axis)	Jan
Colorado	585

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 4-104 Output Grid from MDX Example

(axis)	Sales	Rolling Sales
Qtr2	27401	27014
Apr	8969	8960
May	9071	8997
Jun	9361	9133.667

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

```
member.Lead (index [,layertype ] [, hierarchy ])
```

```
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.

```
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 4-105 Output Grid from MDX Example

(axis)	Actual	Budget
(axis)	Marketing	Expected Sales in Next Month
Jan	5223	30000
Feb	5289	30200
Mar	5327	30830
Apr	5421	31510
May	5530	32900
Jun	5765	33870
Jul	5985	33820
Aug	6046	31000
Sep	5491	29110
Oct	5388	29540
Nov	5263	30820
Dec	5509	#Missing

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:

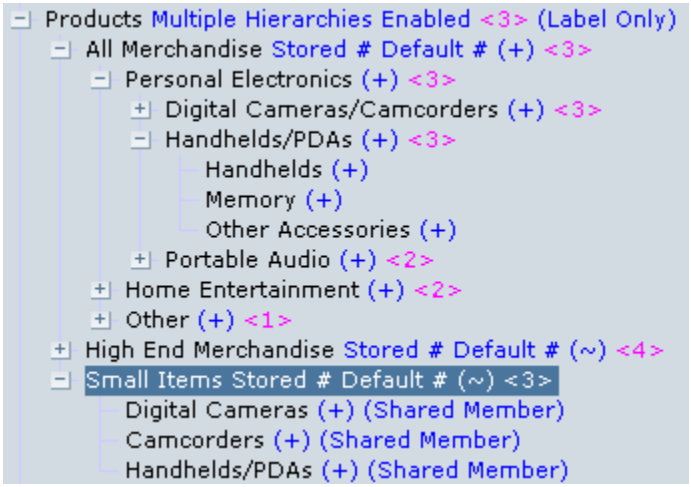
Table 4-106 Output Grid from MDX Example

(axis)	Items Per Package
Digital Cameras	3041
Camcorders	3830
Photo Printers	6002
Memory	23599
Other Accessories	117230
Boomboxes	10380
Radios	20009

[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.



The following query

```
SELECT
{Units} ON COLUMNS,
Leaves ([Small Items]) ON ROWS
FROM [Asosamp.Sample]
```

Returns the the following grid:

Table 4-107 Output Grid from MDX Example

(axis)	Items Per Package
Digital Cameras	3041
Camcorders	3830
Memory	23599
Other Accessories	117230

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 4-108 Output Grid from MDX Example

(axis)	Qtr1	Qtr2	Qtr3	Qtr4
100-10	5096	5892	6583	5206
100-20	1359	1534	1528	1287
100-30	593	446	400	544
200-10	1697	1734	1883	1887
200-20	2963	3079	3149	2834
200-30	1153	1231	1159	1093
200-40	908	986	814	1384
300-10	2544	3231	3355	3065
300-20	690	815	488	518
300-30	2695	2723	2855	2820
400-10	2838	2998	3201	2807
400-20	2283	2522	2642	2404
400-30	-116	-84	-145	-49
100-20	1359	1534	1528	1287
200-20	2963	3079	3149	2834
300-30	2695	2723	2855	2820

See Also

Generation

Levels

IsLevel

Levels

Returns the level specified by the input level number.

Syntax

```
dimension.Levels ( index )
```

```
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

```
SELECT
 [Year].[Qtr1].Level.Members
ON COLUMNS,
 [Product].Levels(0).Members
ON ROWS
FROM Sample.Basic
```

returns the grid:

Table 4-109 Output Grid from MDX Example

(axis)	Qtr1	Qtr2	Qtr3	Qtr4
100-10	5096	5892	6583	5206
100-20	1359	1534	1528	1287
100-30	593	446	400	544
200-10	1697	1734	1883	1887
200-20	2963	3079	3149	2834
...	...	...	...	...
300-30	2695	2723	2855	2820

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
FROM
 Sample.Basic
WHERE
  ([Market].[East], [Product].[Cola])
```

returns the following grid:

Table 4-110 Output Grid from MDX Example

(axis)	Jan	Feb	...	Nov	Dec
Sales	1812	1754	...	1708	1841
Ln_Sales	7.502	7.470	...	7.443	7.518

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

```
Max ( 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 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 4-111 Output Grid from MDX Example

(axis)	Max Qtr2 Sales
Colas	27187
Root Beer	27401
Cream Soda	25736
Fruit Soda	21355
Diet Drinks	26787

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.

```
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

```
SELECT
 {[Measures].[Sales], [Measures].[Profit]}
ON COLUMNS,
 MemberRange([Year].[Feb], [Year].[Nov])
ON ROWS
FROM Sample.Basic
```

returns the grid:

Table 4-112 Output Grid from MDX Example

(axis)	Sales	Profit
Feb	32069	8346

Table 4-112 (Cont.) Output Grid from MDX Example

(axis)	Sales	Profit
Mar	32213	8333
Apr	32917	8644
May	33674	8929
Jun	35088	9534
Jul	36134	9878
Aug	36008	9545
Sep	33073	8489
Oct	32828	8653
Nov	31971	8367

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:

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 4-113 Output Grid from MDX Example

East	West	South	Central
24161	29861	13238	38262

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 4-114 Output Grid from MDX Example

(axis)	Minimum Sales in Quarter
Qtr1	31538
Qtr2	32917
Qtr3	33073
Qtr4	31971

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 4-115 Output Grid from MDX Example

(axis)	Margin %	Profit %	Factor
Jan	55.102	25.44	4.217
Feb	55.387	26.025	3.337
Mar	55.267	25.868	3.530

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 ) ]
```

```
NextMember ( member [,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 4-116 Output Grid from MDX Example

(axis)	Sales	Delta from Previous Month	Delta from Next Month
Jan	2105	2105	44
Feb	2061	-44	-65
Mar	2126	65	-132
Apr	2258	132	-89
May	2347	89	-278
Jun	2625	278	-110
Jul	2735	110	62
Aug	2673	-62	311
Sep	2362	-311	268
Oct	2094	-268	28
Nov	2066	-28	-222
Dec	2288	222	2288

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 4-117 Output Grid from MDX Example

(axis)	Number of Markets
100-10	20
100-20	16
100-30	8
200-10	20
200-20	17
200-30	9
200-40	3

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 presense 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),
  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 4-118 Output Grid from MDX Example

(axis)	Items Per Package
Digital Cameras,12201	4
Camcorders,12201	3
Photo Printers, 12201	2
Digital Recorders, 12201	2
Desktops,12201	3
Digital Cameras,12212	5
Camcorders,12212	2

Table 4-118 (Cont.) Output Grid from MDX Example

(axis)	Items Per Package
Photo Printers, 12212	3
Flat Panel, 12212	1
HDTV,12212	1
Home Theater, 12212	1
Desktops, 12212	2
Notebooks,12212	1
Digital Cameras,12223	1
Camcorders,12223	1
Photo Printers,12223	4
HTDV,12223	1
Notebooks,12223	1
Camcorders,12229	4
HDTV,12229	1
Home Theater,12229	3
Desktops,12229	1
Digital Cameras,12249	2
Photo Printers,12249	3
Projection TVs,12249	1
HDTV,12249	2
Home Theater,12249	1
Digital Recorders,12249	1
Notebooks,12249	1
Camcorders,12257	2
Photo Printers,12257	4
Projection TVs,12257	2
HDTV,12257	1
Home Theater,12257	3
Digital Recorders,12257	1

NTile

Returns a division number of a tuple in a set. This function only applies to aggregate storage databases.

Syntax

```
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

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

layer
A layer specification. If omitted, the first descendant of *member* is used. If *member* is omitted, the first child of the Time dimension is assumed.

member
Optional. A member specification. If omitted, the first child of the Time dimension is assumed (for example, `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 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 *layer* and *member* arguments are given as input, Openingperiod returns the first descendant of the input member at the input layer. For example, `Openingperiod(Year.generations(3), Qtr3)` returns `Jul`. If the input *member* and *layer* are the same layer, the output is the input member. For example, `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.

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 4-119 Output Grid from MDX Example

(axis)	Budget	Budget
(axis)	Marketing	Sales
400-30	510	3240
100-30	450	3400
300-20	550	3800
200-40	310	2830
200-30	550	4060

Table 4-119 (Cont.) Output Grid from MDX Example

(axis)	Budget	Budget
(axis)	Marketing	Sales
100-20	1160	8800
100-20	1160	8800
200-10	2090	10330
400-20	880	6590
300-10	1450	10080
300-30	1080	7880
300-30	1080	7880
400-10	790	7410
200-20	1080	9590
200-20	1080	9590
100-10	1800	17230

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 4-120 Output Grid from MDX Example

(axis)	ProdGen	ProdLev
Product	3	0
100	2	1
100-10	3	0
100-20	3	0
100-30	3	0
200	3	0
200-10	2	1
200-20	3	0
200-30	3	0
200-40	3	0
300	2	1
300-10	3	0
300-20	3	0
300-30	3	0
400	2	1
400-10	3	0
400-20	3	0
400-30	3	0
Diet	2	1
100-20	3	0
200-20	3	0
300-30	3	0

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

```
Ancestor(member, layer)
```

The member ANCESTOR2 is then computed as

```
Lag(ANCESTOR1, index)
```

The return value of this function is then computed as

```
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

```
member.Parent [(hierarchy) ]
```

```
Parent ( member [, hierarchy ] )
```

Parameters

member
A member specification.

hierarchy
Optional. A specific hierarchy within the time dimension.

Example

Example 1

```
SELECT
 {Parent ([100-10])}
ON COLUMNS
FROM
  sample.basic
```

returns the parent of 100-10:

Table 4-121 Output Grid from MDX Example

100
30468

Example 2

The following query uses Filter to find the months in which Sales for [Product].[100] are higher than 8,570. The Parent function is used with Generate to create a set consisting of the parents (quarters) of the high-sales months.

```
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
WHERE
 ([Product].[100])
```

This query returns the grid:

Table 4-122 Output Grid from MDX Example

(axis)	Sales
Qtr2	27187
Qtr3	28544
Qtr4	25355

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 4-123 Output Grid from MDX Example

(axis)	Sales	Delta from Previous Month	Delta from Next Month
Jan	2105	2105	44
Feb	2061	-44	-65
Mar	2126	65	-132
Apr	2258	132	-89
May	2347	89	-278
Jun	2625	278	-110
Jul	2735	110	62
Aug	2673	-62	311
Sep	2362	-311	268
Oct	2094	-268	28
Nov	2066	-28	-222
Dec	2288	222	2288

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]}
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 4-124 Output Grid from MDX Example

(axis)	Margin %	Margin % Rem
Product	55.262	0.262
100	57.273	0.273
100-10	61.483	0.483
100-20	51.479	0.479
100-30	50.424	0.424

Table 4-124 (Cont.) Output Grid from MDX Example

(axis)	Margin %	Margin % Rem
200	55.540	0.540
200-10	54.270	0.270
200-20	56.436	0.436
200-30	56.450	0.450
200-40	55.753	0.753
300	54.238	0.238
300-10	55.816	0.816
300-20	42.992	0.992
300-30	57.551	0.551
400	53.600	0.600
400-10	57.354	0.354
400-20	56.299	0.299
400-30	39.477	0.477
Diet	55.397	0.397
100-20	51.479	0.479
200-20	56.436	0.436
300-30	57.551	0.551

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

```
SELECT
CrossJoin (
         Union (
            Siblings ([Old Fashioned]),
            {([Root Beer]), ([Cream Soda])}
         ),
         {(Budget), ([Variance])}
      )
ON COLUMNS
from Sample.Basic
```

returns the grid:

Table 4-125 Output Grid from MDX Example

Old Fashioned		Diet Root Beer		Sarsaparilla		Birch Beer		Root Beer		Cream Soda	
Budget	Variance	Budget	Variance	Budget	Variance	Budget	Variance	Budget	Variance	Budget	Variance
11640	-4439	14730	-2705	5050	-414	4530	-438	35950	-7996	29360	-3561

Example 2

The following examples are based on a Years – Quarters – Months Time hierarchy.

```
Siblings([Feb 2000], LEFT, INCLUDEMEMBER)
```

Returns {[Jan 2000], [Feb 2000]}.

```
Siblings([Feb 2000], RIGHT, EXCLUDEMEMBER)
```

Returns {[Mar 2000]}.

```
Siblings([Mar 2000], LEFT)
```

Returns {[Jan 2000], [Feb 2000], [Mar 2000]}.

```
Siblings([May 2000], RIGHT)
```

Returns {[May 2000], [Jun 2000]}.

```
Siblings([Mar 2000])
```

OR

```
[Mar 2000].Siblings
```

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 4-126 Output Grid from MDX Example

(axis)	Actual	Budget
(Sales, 100)	678	640
(Sales, 200)	551	530
(Sales, 300)	663	510
(Sales, 400)	587	620
(Sales, Diet)	#Missing	#Missing
(Sales, Product)	2479	2300
(Std Deviation, Product)	60.723	64.55

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

```
Stddevp ( 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 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 4-127 Output Grid from MDX Example

(axis)	Actual	Budget
(Sales, 100)	678	640
(Sales, 200)	551	530
(Sales, 300)	663	510
(Sales, 400)	587	620
(Sales, Diet)	#Missing	#Missing
(Sales, Product)	2479	2300
(Std Deviation, Product)	52.59	55.9

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

```
Subset ({Product.Members},0)
```

returns the set:

```
{ 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], 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

```
Subset ({Product.Members},0,4)
```

returns the set:

```
{ Product, [100], [100-10], [100-20] }
```

Therefore, the following query

```
Select
 Subset ({Product.Members},0,4)
on columns
from sample.basic
```

returns the grid:

Table 4-128 Output Grid from MDX Example

Product	100	100-10	100-20
105522	30468	22777	5708

Substring

Returns the substring between a starting and ending position. Both the positional arguments are 1-based.

Syntax

```
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

```
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: *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 numeric value expression is evaluated in the context of that tuple and the resulting values are summed up.

The return value of Sum is `#MISSING` if either of the following is true:

- The input set is empty.
- All tuple evaluations result in `#MISSING` values.

Example

```
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])
```

returns the grid:

Table 4-129 Output Grid from MDX Example

(axis)	Total Expenses
New York	8914
Massachusetts	3412

Table 4-129 (Cont.) Output Grid from MDX Example

(axis)	Total Expenses
Florida	5564
East	25310
California	11737
Texas	4041
Illinois	6900
Ohio	5175
Colorado	6131
Central	34864
Sum Expense for Main States	51874

See Also

Aggregate

Tail

Returns the last *n* members or tuples present in a set.

Syntax

```
Tail ( set [,index ] )
```

Parameters

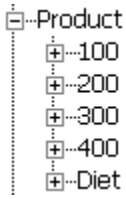
set
The set from which to take items.

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

```
[Product].children
```

returns the set:

```
{ [100], [200], [300], [400], [Diet] }
```

Therefore, the following expression

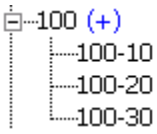
```
 Tail (
  [Product].children, 2)
```

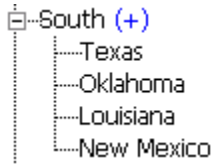
returns the last two members of the previous result set:

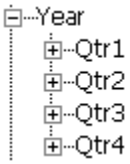
```
{ [400], [Diet] }
```

Example 2

This example uses the following parts of the Sample Basic outline:







The following expression

```
 Crossjoin ( [100].children, [South].children )
```

returns the set:

```
{ ([100-10], Texas), ([100-10], Oklahoma), ([100-10], Louisiana),
([100-10], [New Mexico]),
  ([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:

```
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 4-130 Output Grid from MDX Example

(axis)	(axis)	Qtr1	Qtr2	Qtr3	Qtr4
100–20	**Texas**	206	199	152	82
	Oklahoma	84	66	55	79
	Louisiana	119	158	171	104
	New Mexico	–103	–60	–97	–18
100–30	**Texas**	#Missing	#Missing	#Missing	#Missing
	Oklahoma	#Missing	#Missing	#Missing	#Missing
	Louisiana	#Missing	#Missing	#Missing	#Missing
	New Mexico	#Missing	#Missing	#Missing	#Missing

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 4-131 Output Grid from MDX Example

(axis)		Qtr1	Qtr2	Qtr3	Qtr4
100-20	**Texas**	206	199	152	82
100-20	**Oklahoma**	84	66	55	79
100-20	**Louisiana**	119	158	171	104
100-20	**New Mexico**	-103	-60	-97	-18

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.

```
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 4-132 Output Grid from MDX Example

Axis-1	Axis-1.properties	Budget	Revised Budget
200-10	(Intro Date = 09-27-1995, type: TIME,)	11060	11060
200-20	(Intro Date = 07-26-1996, type: TIME,)	9680	10648
200-30	(Intro Date = 12-10-1996, type: TIME,)	3880	4268
200-40	(Intro Date = 12-10-1996, type: TIME,)	2660	2926

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-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`)

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`

8. `15-01-06`
9. `15 January 06`
10. `15 Jan 06`
11. `January 15 06`
12. `Jan 15 06`
13. `01-15-06`
14. `06/01/15`
15. `060115`
16. `15 January 2006`
17. `15 Jan 2006`
18. `2006-01-15`
19. `2006/01/15`
20. `Sunday, January 15, 2006`
21. `1/8/06` (m/d/yy)

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.

```
SELECT
 {[Sales]}
ON COLUMNS,
```

```
 {
   StrToMbr(
        FormatDate(
                TodateEx("mm.dd.yyyy", "05.31.2005"),
                "Mon dd yyyy"
        )
   )
 }
ON ROWS
FROM Mysamp.basic
WHERE (Actual, California, Cola);
```

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

```
TopCount ( set , index [,numeric_value_expression ] )
```

Parameters

set
The set from which the top *n* elements are selected.

index
The number of elements to include in the set (*n*).

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.

```
SELECT
 CrossJoin(
           [Product].[Diet].Children,
           [Year].Children
          )
ON COLUMNS,
 TopCount(
          [Market].Levels(0).Members,
          5,
          [Product].[Diet]
         )
ON ROWS
FROM Sample.Basic
WHERE ([Scenario].[Actual], [Measures].[Sales])
```

This query returns the grid:

Table 4-133 Output Grid from MDX Example

(axis)	100-20	100-20	100-20	100-20	200-20	200-20	200-20	200-20	300-30	300-30	300-30	300-30
(axis)	**Qtr1**	**Qtr2**	**Qtr3**	**Qtr4**	**Qtr1**	**Qtr2**	**Qtr3**	**Qtr4**	**Qtr1**	**Qtr2**	**Qtr3**	**Qtr4**
Illinois	755	958	1050	888	1391	1520	1562	1402	675	755	859	894
California	367	491	506	468	1658	1833	1954	1706	700	802	880	673

Table 4-133 (Cont.) Output Grid from MDX Example

(axis)	100-20	100-20	100-20	100-20	200-20	200-20	200-20	200-20	300-30	300-30	300-30	300-30
Colorado	700	802	880	673	549	465	412	539	1006	921	892	991
Washington	637	712	837	704	459	498	597	514	944	799	708	927
Iowa	162	153	121	70	129	129	129	129	1658	1833	1954	1706

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]
```

```
            )
ON ROWS
FROM Sample.Basic
WHERE ([Scenario].[Actual],
       [Measures].[Sales])
```

This query returns the grid:

Table 4-134 Output Grid from MDX Example

(axis)	100-20	100-20	100-20	100-20	200-20	200-20	200-20	200-20	300-30	300-30	300-30	300-30
(axis)	**Qtr1**	**Qtr2**	**Qtr3**	**Qtr4**	**Qtr1**	**Qtr2**	**Qtr3**	**Qtr4**	**Qtr1**	**Qtr2**	**Qtr3**	**Qtr4**
Illinois	755	958	1050	888	1391	1520	1562	1402	675	755	859	894
California	367	491	506	468	1658	1833	1954	1706	700	802	880	673
Colorado	700	802	880	673	549	465	412	539	1006	921	892	991

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 4-135 Output Grid from MDX Example

(axis)	100-20	100-20	100-20	100-20	200-20	200-20	200-20	200-20	300-30	300-30	300-30	300-30
(axis)	**Qtr1**	**Qtr2**	**Qtr3**	**Qtr4**	**Qtr1**	**Qtr2**	**Qtr3**	**Qtr4**	**Qtr1**	**Qtr2**	**Qtr3**	**Qtr4**
Illinois	755	958	1050	888	1391	1520	1562	1402	675	755	859	894
California	367	491	506	468	1658	1833	1954	1706	700	802	880	673
Colorado	700	802	880	673	549	465	412	539	1006	921	892	991
Washington	637	712	837	704	459	498	597	514	944	799	708	927
Iowa	162	153	121	70	129	129	129	129	1658	1833	1954	1706
Florida	620	822	843	783	548	611	657	577	332	323	260	159
Oregon	389	303	277	322	1006	921	892	991	263	231	197	184

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]),
```

```
([2005], [May]),
([2005], [Jun]),
([2005], [Jul]),
([2005], [Aug]),
([2005], [Sep]),
([2005], [Oct]),
([2005], [Nov]),
([2005], [Dec]),
([2006], [Jan]),
([2006], [Feb])}
```

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 4-136 Output Grid from MDX Example

(axis)	Sales	Profit
East	2248	1156
New York	912	370
Massachusetts	665	564
Florida	286	104
California	912	370
Texas	567	206
Central	1392	369
Illinois	567	208
Ohio	85	18
Colorado	199	70

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 4-137 Output Grid from MDX Example

(axis)	Sales	Profit
East	2248	1156
New York	912	370
Massachusetts	665	564
Florida	286	104

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 4-138 Output Grid from MDX Example

Old Fashioned		Diet Root Beer		Sarsaparilla		Birch Beer		Root Beer		Cream Soda	
Budget	**Variance**	**Budget**	**Variance**	**Budget**	**Variance**	**Budget**	**Variance**	**Budget**	**Variance**	**Budget**	**Variance**
11640	-4439	14730	-2705	5050	-414	4530	-438	35950	-7996	29360	-3561

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] <= 0.00` is a shortcut for `[Qtr1].VALUE <= 0.00`.

Example

Example 1

```
[Sales].Value
```

Returns the value of the Sales measure.

```
([Product].CurrentMember, [Sales]).Value
```

Returns the value of the Sales measure for the current member of the Product dimension.

> **Note:**
>
> The Value keyword is optional. The above expressions can also be entered as:

```
[Sales]
```

Which is equivalent to `[Sales].Value`

```
([Product].CurrentMember, [Sales])
```

Which is equivalent to `([Product].CurrentMember, [Sales]).VALUE`

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 4-139 Output Grid from MDX Example

(axis)	Sales
100-10	62824
300-10	46956
200-10	41537
200-20	38240
200-20	38240
300-30	36969
300-30	36969
400-10	35799
400-20	32670
100-20	30469
100-20	30469
200-30	17559
300-20	17480
400-30	15761
100-30	12841
200-40	11750

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 4-140 Output Grid from MDX Example

Cola	Diet Cola	Diet Cream
22777	5708	11093

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` 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 ( [member ] )`

Parameters

xTD
Values:

Parameter	Value
HTD	History-To-Date (H-T-D)
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

member
Member specification. Should be a member from the time dimension.

Notes

- `xTD ( [member ] )` is equivalent to `PeriodsToDate ( layer, [member])` where *layer* is assumed to be the value set in the corresponding Dynamic Time Series member in the database outline.

  For example, in Sample Basic, `QTD ([member])` is equivalent to `PeriodsToDate ( Year.Generations(2) [,member])`, because Q-T-D is Generation 2 in the Year dimension.

- The *x*TD functions YTD, QTD, MTD, etc. are not relevant for use in aggregate storage databases, because the *x*TD 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 PeriodsToDate function with aggregate storage databases in place of the *x*TD functions.

  For example,

```
YTD(May) is equivalent to
PeriodsToDate(Year.Generations(1), May)
```

```
QTD(May) is equivalent to
PeriodsToDate(Year.Generations(2), May).
```

Example

```
QTD([Feb])
```

returns the set `{[Jan], [Feb]}`.

`QTD([Feb])` is equivalent to `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]}`.

`HTD([May])` is equivalent to `PeriodsToDate([Year].Generations(1), [May])`, because the dynamic-time-series member H-T-D is defined as Generation 1 of the Year dimension.

> **Note:**
>
> 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.