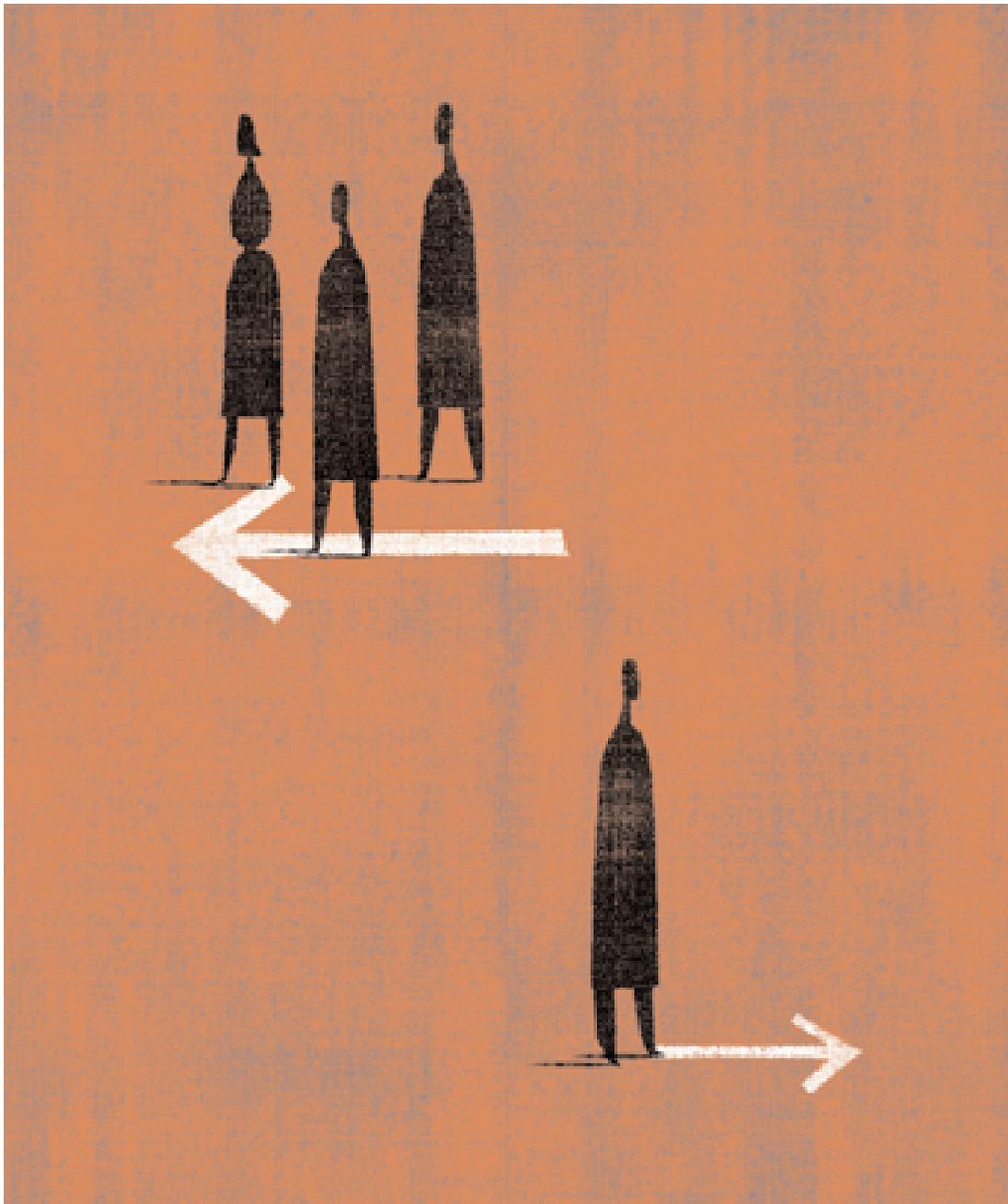


MDX ALLOWS COMPLEX,  
MULTIDIMENSIONAL QUERIES



A HYPERION WHITE PAPER

APRIL  
2004



---

## TABLE OF CONTENTS

---

3	INTRODUCING MDX TO REPORT WRITER USERS
4	REPORT LAYOUT
4	LABELS AND PROPERTIES
4	MEMBER SELECTION
5	SORTING
5	CALCULATIONS
7	WHY USE MDX?
9	API AND APPLICATION CONSIDERATIONS
10	MDX QUERY EXECUTION ORDER
14	MDX CALCULATED MEMBER AND SOLVE-ORDER
16	MDX CALCULATED MEMBERS AND SECURITY
17	MDX AND MEMBER ORDERING
17	MDX AND DYNAMIC TIME SERIES MEMBERS SUPPORT
17	MDX AND SORTING
18	MDX AND ADVANCED FILTERING OPTIONS
20	MDX AND SUPPRESSION OF MISSING VALUES IN OUTPUT
22	APPENDIX:
22	REPORT LAYOUT COMMANDS
23	MEMBER SELECTION COMMANDS
24	SORTING COMMANDS
25	DATA ORDERING AND DATA RANGE COMMANDS
26	FORMAT COMMANDS
27	COLUMN OR ROW CALCULATION COMMANDS
28	MEMBER NAMES AND ALIASES

**M**DX (Multidimensional Expressions) is a new data extraction mechanism introduced in Hyperion Essbase Analytic Services release 7.0 (herein referred to as “Essbase”), that provides a method for forming complex and robust multidimensional queries.

MDX allows you to:

- Use declarative query as input and set of structures as output
- Transfer more calculations and selections into the query from the client
- Use the same set of APIs for data and metadata queries
- Return cell properties and properties of the members in the same query
- Automate analysis with fewer steps

In this document, we will provide an overview of MDX, its critical implementation aspects, and compare its interface with a report writer language, highlighting similarities and differences as relevant. We will also describe specific details of MDX query execution semantics through examples. A solution section to demonstrate migration of existing report writer-based queries into the MDX syntax is included. All the example queries assume the Sample: Basic database as the queried data source.

---

INTRODUCING MDX TO  
REPORT WRITER USERS

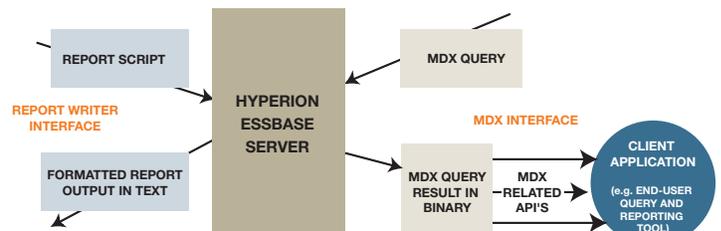
---

MDX (Multidimensional Expressions) is a declarative query language that can be used to

extract information from an Essbase database. The output of an MDX query has the form of a cube itself. The query captures both the output specification and the dimensionality of the resulting cube. MDX-related procedural APIs allow a query to be submitted for execution and the navigation of the query result consists of member combinations and corresponding data value. A query and reporting application will use these APIs to format and display the result.<sup>1</sup>

Report writer is also a declarative scripting language that can be used to report on an Essbase database. You can combine report writer's selection, layout, and formatting commands to build reports with different output specifications.

In this section, we will provide a brief overview of the two query interfaces comparing and contrasting some commonly used functions and execution paradigms.



---

---

**REPORT LAYOUT**

Report writer commands enable a report to be defined in terms of columns and rows across one or more pages. The equivalent construct in MDX is the axis – columns, rows and pages are considered three of the possible axes. A query may have only one or two axes as well as three or more (the maximum number that can be used within a single query is 64). An optional slicer axis defines a point of view for the query.

MDX does not support functionality similar to the report-writer formatting commands such as PAGEHEADING. The resultant data structure is not a text report, and it is left to the client application to format results for presentation.

**LABELS AND PROPERTIES**

Both MDX and report writer support the ability to display query output using either member name or alias. MDX supports the display and use of both in the same query. As described later, member attributes and cell properties can also be presented to the client through the MDX APIs.

**MEMBER SELECTION**

Report writer member selection commands enable construction of queries based on member (hierarchical) relationships in an outline. MDX supports a much richer set of functions for the same purpose. For example, the member relationship function, descendants, is exposed in the report writer in two variants – Descendants (all members in a sub-tree/sub-hierarchy rooted at the member whose descendants are desired) and Idescendants (same as descendants but includes the rooted member in the result set). The MDX Descendants func-

tion, on the other hand, provides several variants that enable a more comprehensive selection.

For example, Descendants (<member>, <layer>, SELF) returns the members of a particular generation or a level in a sub-tree/sub-hierarchy rooted at the <member>. To return the same set of members in report writer, you have to use Descendants (<member>). To return all the members in the sub-tree rooted at the member, use Gen or Lev command to return all the members of a generation or a level of the entire dimension, and, finally, use the Link command with an And operator.

Another example is the MDX Uda function that takes as its argument the root member of a sub-hierarchy in a dimension of interest. The function returns members with the specified UDA in the sub-tree. The report writer UDA command, in contrast, only works for the top-most member of the dimension. As a result, you have to use the Link command to prune out members not in the sub-tree in the dimension of interest.

MDX also provides functions that have no counterparts in the report writer interface, such as the LastPeriods function, which can be used to analyze a sales trend over the last n months starting from a given month. Another example is the ParallelPeriod function used to compare same-period metrics in a prior year. OpeningPeriod and ClosingPeriod can be used to achieve Time Balance functionalities similar to those provided by the custom TB First, TB Last outline tags.

1. For more information on a complete list of MDX functions implemented, please refer to

the MAXL DML documentation in the Essbase XTD Analytic Services 7.0 Technical Reference.

**SORTING**

MDX, like report writer, supports both metadata and data-based sorting. An example of metadata-based sorting is sorting by member name or generation number. MDX functions like Filter, TopCount, and others provide functionality similar to report writer data range commands to restrict the range of selected data.

**CALCULATIONS**

Whereas report writer allows custom calculations to be performed on the result set of a query, MDX allows custom calculations to be defined within the scope of a query itself and produces the result in the output. This is accomplished through calculated members. A calculated member is a member with a specific custom formula defined for the duration of a query and is similar to a dynamic-calc member defined in an Essbase outline. A calculated member can use a rich set of computation and referencing primitives.

Unlike dynamic-calc members, calculated members are not considered part of the Essbase outline and will not be returned in functions that return a set of members from metadata. For example, the result of the MDX expression Year.Members includes the member Qtr1, which is a dynamic calc member. In contrast, a MDX calculated member is defined only for the duration of a query. It is not part of the database outline, and metadata functions do not include such members in their result sets.

Consider a calculated member definition such as:

```
MEMBER [Year].[MyQtr] as 'Feb +
Mar + Apr'
```

The calculated member MyQtr will not appear in the result of the function Year.Members, which will be resolved based on the database in Essbase. Therefore, a calculated member has to be explicitly specified whenever it is needed in a query.

For example, assume you must create a new type of scenario that will indicate the difference with the last quarter's value.

```
WITH
MEMBER Scenario.[Qtr to Qtr Variance] AS
'IIF (Is (Year.CurrentMember, Qtr1),
    0,
    Year.CurrentMember - Year.CurrentMember.PrevMember
) '
SELECT
{ Scenario.Children, [Qtr to Qtr Variance] } on Columns,
Year.Children on Rows
WHERE (East, [100], Sales)
```

East	100	Sales			
	Actual	Budget	Variance	Variance %	Qtr to Qtr Variance
Qtr1	6292	5870	422	7.1891	0
Qtr2	7230	6760	470	6.95266	938
Qtr3	7770	7300	470	6.43836	540
Qtr4	6448	5570	878	15.763	-1322

Note that [Qtr to Qtr Variance] is specified explicitly in the column axis. For Qtr1, there is no previous quarter – so the calculated member formula handles a special case for Qtr1 and evaluates the [Qtr to Qtr Variance] as 0.

The expression IsValid (Year.CurrentMember.PrevMember) returns True if the current mem-

ber along Year dimension has a previous member at that generation – for the first member, for example January or Qtr1, IsValid will return False. Thus, you can rewrite the same formula so that it works on members of all generations in Year dimension as follows.

```
WITH
MEMBER [Scenario].[Qtr to Qtr Variance] as
'IIF (IsValid (Year.CurrentMember.PrevMember),
Year.CurrentMember - Year.CurrentMember.PrevMember,
0
) '
SELECT
{ Scenario.Children, [Qtr to Qtr Variance] } on Columns,
Year.Members on Rows
FROM Sample.Basic
WHERE (East, [100], Sales)
```

East	100	Sales			
	Actual	Budget	Variance	Variance %	Qtr to Qtr Variance
Year	27740	25500	2240	8.78431	0
Qtr1	6292	5870	422	7.1891	0
Jan	2105	1960	145	7.39796	0
Feb	2061	1920	141	7.34375	-44
Mar	2126	1990	136	6.83417	65
Qtr2	7230	6760	470	6.95266	938
Apr	2258	2110	148	7.01422	132
May	2347	2190	157	7.16895	89
Jun	2625	2460	165	6.70732	278
Qtr3	7770	7300	470	6.43836	540

The addition of a calculated member in a query does not affect the database outline and hence no database restructuring is needed. In addition, this new member (assuming it is along dense dimensions) does not increase the size of the big block – the unit of data that Essbase server internally needs to allocate in memory for calculating dynamic-calc data values -- and hence has no effect on the dynamic calculation cache.

Readers familiar with the calculation language in Essbase will notice similarities between the behavior of certain calculation functions and corresponding MDX functions.

Comparison of MDX to Essbase calculation language is outside the scope of this document. In the MDX Calculated Members and Security section, we will describe other differences in terms of the security model between a calculated member in an MDX query and a formula attached to a dynamic calc member in the Essbase outline.

In this section we provided an overview of MDX with the report writer functionality as the backdrop. We will now discuss why it may be appropriate to transition your custom query interfaces to use MDX.

---

#### WHY USE MDX?

---

If you are familiar with reports and are wondering why you may want to switch to MDX, this section describes specific concepts in MDX not available in reports.

### 1. Declarative query as input and set of structures as output

Report writer interface lets you define the query in a declarative fashion – but the report writer output is in text. As a result, if you are developing a custom application using the report writer interface, you have to parse the report output before you can do any application-specific processing. You may have to use special formatting commands, to identify information like member name/alias/page break etc. Providing an application with a spreadsheet grid using the grid API requires a number of API calls that use a number of data structures in order to specify the query, which can be complex when the query is not based on plain, hierarchy operations. The only equivalent of MDX is the Grid API's free-form APIs such as `EssGBeginReport()`. However, this works only for limited kinds of reports and will be deprecated in future Essbase releases. MDX supports a powerful expressive language as well as a set of APIs to introspect the contents of the result set.

### 2. Transfer more calculations and selections into the query from the client

One of the unique aspects of MDX is the ability to write a query that can access sets of members/data values that is completely different from the set of members/values reported in the final query output.

For example, assume you are responsible for the Eastern Region, and you must analyze the budgeted marketing dollars for products in Q2 that had Q1 sales in excess of \$1,000. You wish to return cells for Budget and Q2 for East based on data from Actual and Q1 for East.

This query can be written in MDX as follows.

```
SELECT
{(Budget, Marketing)} on Columns,
Filter (
  Product.Levels(0).Members,
  (Qtr1, Actual, Sales) > 1000
) on Rows
WHERE (East, Qtr2)
```

East	Qtr2
	Budget
	Marketing
100-10	490
200-10	910
200-40	310
300-10	390
300-20	250
400-10	130
400-30	90

The entire query uses data for the Eastern regions. The Filter() expression prunes out those Level-0 products based on the Actual Sales numbers in Qtr1 – note that the query outputs the Budget numbers for Qtr2, whereas the Actual values for Qtr 1 were used during query execution.

In a subsequent section (MDX Query Execution Order), we will describe in more detail the concept of a query context as it applies to MDX query execution.

### 1. Same set of APIs for data and metadata queries

The querying syntax as well as output processing

interfaces does not change, regardless of whether you are interested in the actual data values or just the metadata combinations. For example, you may want to query for states in the Eastern region, but are not interested in the actual data values that may exist for those states. In this case, you can execute the following query in data-less mode:

```
SELECT
Children (East) on Columns
FROM Sample.Basic
```

The next query is a data query retrieving the actual sales in Qtr1 for Colas in the Eastern-regional states:

```
SELECT
Children (East) on Columns,
Children (Colas) on Rows
FROM Sample.Basic
WHERE (Qtr1, Actual, Sales)
```

---

---

Note that in both cases the query uses the same children operator to query for states in the Eastern region. The output processing is also identical. Logically, an MDX-API based client program may be interested in

- a.) finding the number of tuples (known in MDX as members and cross-members ) on the column axis(number of states in this case)
- b.) finding the tuple at ith position ( i.e. get name of the ith state).

Invoking the function `EssMdxSetDataLess()` before submitting a query for execution tells the server that the query is interested in retrieving just the metadata. Metadata queries may be beneficial in certain types of screening. Note that although a query may be interested in obtaining only metadata, the selection criterion for the metadata itself could be based on data. For instance, a query may retrieve just those Product-Market combinations for which quarter-to-quarter variance is significant.

### **1. Property support: the ability to return cell properties and properties of the members in the same query**

The MDX language includes constructs that can be used to request Grid-API style cell-properties in the query output. Such properties identify whether a cell is read-only, dynamically calculated etc. In addition, MDX has constructs that allow certain member properties to be displayed in the output as well. MDX extends the notion of properties to include user-defined attributes (UDA) and attribute dimensions in Essbase. For example, you can request Color and Size properties for every member of the Product dimension (Color and Size are attributes of product) or if you want to

know whether a market is a “Major Market” for every member from Market dimension in the query result. Report writer does not support cell-properties. You can get some of the member properties (e.g. use {IndentGen} and parse the output to figure out the generation number). However, such a task is extremely cumbersome and error-prone in report writer. Neither Grid API nor report writer supports the MDX style of exposing user-defined attributes or attribute dimensions as properties.

### **2. Automating analysis with fewer steps**

You can write MDX queries to automate the production of reports that would otherwise need substantial human intervention. For example, you may want to report on the markets at the regional level – you want the state level details for the regions with a negative variance from last quarter. You might want to report on the states with substantial (30% and above for example) contribution to its region’s sales. The richness of the MDX expression model makes these queries straightforward.

---

#### API AND APPLICATION CONSIDERATIONS

---

3. MDX is not an end-user interface for churning out end-user reports from the Essbase Server. Report writer has a rich formatting support and you can write a report to output the query result in a presentable fashion. Of course, you can also use the powerful Essbase spreadsheet Add-in interface to design robust Excel spreadsheets to retrieve data from Essbase.

On the other hand, MDX lets the user specify the query and supports a set of APIs to navigate over the query results. The Essbase Multidimensional Access Language (MAXL),

for example, chooses to display the query results in a two-dimensional format. Use of these APIs is required to create a reporting interface.

4. MDX is not an interface for churning out database exports, or very large production-type reports from the Essbase Server. The report writer interface supports streaming of the result set. Neither the server nor the client needs to hold on to the entire result set. You may potentially write a report to back up a portion of the database without needing the server to go offline. The Grid API (used by the spreadsheet interface) does not stream in the server and in the client. As a result, when a large query is executed, the server has to hold on to the entire result set before sending it to the client.

MDX, on the other hand, supports server-side streaming and as a result the server does not need to hold on to the entire result set. The server is capable of sending partial results to the client. In order to provide easy navigation over the query results, the API accumulates the entire result, before returning control to the caller. Even though the API stores the query results in an optimized compressed form, the client may not have enough memory to hold on to the entire result set.

---

MDX QUERY EXECUTION ORDER

---

Recall that an MDX query consists of a data request clause (SELECT), a context clause (FROM) and an optional point-of-view clause (WHERE). The SELECT clause determines the axis dimensions and up to 64 different axes can be specified in a single query. In practice, however, between 2 and 4 axes are more common.

An axis consists of members retrieved from one or more dimensions. Two different axes may not select members from the same dimension for output. However, an axis may internally use a dimension that is already specified as an output dimension in a different axis for computation/member-filtering purposes. In addition to members and cross-members (referred to as tuples in MDX), MDX introduces a new entity called SET. A set is an ordered collection of tuple(s). All tuples within a set must have the same dimensionality and the dimension order of members within each tuple in a set must be identical.

MDX query execution involves independent processing of the set definitions along each axis present in the query and then returning the data points at the intersection of all possible combinations of members (member combinations) from all axis. Let us define MDX context as a calculation context that holds one member from every dimension present in the cube. Contents of MDX context are well defined by the query execution semantics as explained below.

MDX query execution begins with a default MDX context. The context has the dimension root members for all regular dimensions. Hence, the smallest valid MDX query supported in Essbase is “SELECT” which returns the cell-value at the intersection of top members from each dimension (and no members from attribute dimensions). In Sample/Basic, the intersection is (Year, Measures, Product, Market, Scenario).

Query:	SELECT
Result:	(Year, Measures, Product, Market, Scenario) 105,522

Then the Slicer clause (i.e. singleton set definition in the WHERE clause), if present, is processed. Member(s) corresponding to the dimension(s) present in the slicer override the default context to form the new MDX context. This new context is used for the rest of the MDX query execution. Note that the slicer clause can only specify singleton members from each dimension.

For example, the following query

```
SELECT
WHERE (EAST, QTR1)
```

(East, Qtr1, Measures, Product, Scenario)
5,38
0

returns the data point at the intersection of (Qtr1, East, Measures, Product, Scenario). Note that Measures, Product and Scenario are the root members of the respective dimensions not explicitly specified in the query, and are part of the POV by default.

MDX also has an optional WITH clause that can be used to define calculated members and named sets.

The named sets are processed in order in which they are defined in the query. Note that named set evaluations may or may not need the MDX context. For example, if the named set definition is

```
With
:
Set Favourite_Colas as '{[100-10],
[100-30] }',
:
```

the MDX execution does not need the MDX context to resolve the set definition.

In contrast, if the named set definition is the following, then the TopCount function execution needs to use MDX context (since the set construction is a function of data values in the cube).

```
With
Set Favourite_Colas as 'TopCount
(Children ([100]), 2, Sales)'
:
```

If the slicer specifies to use Qtr1, we will get the top 2 children based on Qtr1 sales, whereas if the slicer specifies to use Qtr2, we will get the top 2 children based on Qtr2 sales.

Let us change the above query to add a slicer clause as follows.

```
With
Set Favourite_Colas as 'TopCount
(Children ([100]), 2, Sales)'
:
:
Where {(EAST, QTR1)}
```

First, Children ([100]) will be resolved directly from the outline into [100-10], [100-20] and [100-30]. Next, the set will be pruned to include just the top 2 products based on their sales values.

In order to get any cell data values from Essbase, we need the members from all regular dimensions and, optionally, from attribute dimensions. The sales value for [100-10] does not resolve into one data point – representative members from the missing dimension(s) are required and hence are taken from the MDX context.

So the comparison to find top 2 will take place among the data values below:

(QTR1, Sales, [100-10], EAST, Scenario),

(QTR1, Sales, [100-20], EAST, Scenario),

(QTR1, Sales, [100-30], EAST, Scenario)

1. As an optimization the named-set definitions are processed only if they are later referenced in the query. If they're not actually used, Essbase spends no time building them.

After any necessary named sets are resolved, the individual axis sets (i.e. set definitions along multiple axes) are resolved. The axis set resolution may or may not require MDX context for the same reason explained for named sets.

After all the axes sets are resolved (i.e. all the member combinations/tuples are found for a set), the final retrieval stage pulls data-values

for all possible combinations of members (i.e. member combinations) from all the axes, using MDX context, as needed.

Let us next consider a slightly more advanced role that the MDX context plays using a class of functions called Iterative functions.

An iterative function in MDX is a function that works on a set of tuples performing an operation on every tuple in an iterative fashion. For example, the Sum function

SUM (<input-set>, <numeric-value-expression>)

evaluates a numeric-value-expression for every tuple in the input set, adds all these values up to return the total. The following are some examples of MDX functions that operate on sets iteratively:

SUM	MIN	MAX
AVG	NONEMPTYCOUNT	FILTER
TOPCOUNT	BOTTOMCOUNT	TOPSUM
BOTTOMSUM	TOPPERCENT	BOTTOMPERCENT
ORDER	GENERATE	

All the functions listed above take an expression argument like SUM. However, the expression type is different for each function. For example, the ORDER function requires either a numeric or string expression, GENERATE takes a set as the second argument and the FILTER function takes a conditional expression. The expression is numeric for the remaining functions.

Logically, an iterative function works as follows:

1. For every tuple in input set
2. Begin
3. Save the current MDX context (C1)
4. Create new MDX context (C2) overriding the iterating tuples on C1.
5. Evaluate the expression using C2
6. :
7. : Iterative function specific tasks e.g. cumulatively add the values in case of SUM
8. Restore the saved context (C1) i.e. the current MDX context at this point is C1.
9. End

Now, let us consider an example to illustrate the above logic in steps. The following query sorts the Colas (i.e., Children of [100]) based on Actual Sales in Qtr1 and displays the sorted colas' Budgeted Sales and Marketing in Qtr2

```
SELECT
CrossJoin ({Budget}, {Marketing,
Sales}) on Columns,
Order (Children ([100]), (Qtr1,
Actual)) on Rows
WHERE (Qtr2)
```

Qtr2	Market	
	Budget	Budget
	Marketing	Sales
100-30	450	3400
100-20	1160	8800
100-10	1800	17230

The ORDER function is specified in the set 'Children ([100])' (i.e. {[100-10], [100-20], [100-30]}) to iterate upon.

The expression used for ordering is '(Qtr1, Actual)' – a numeric expression that needs to be computed for each element of the set. The fact that the cell value of (Qtr1, Actual) is required is implicit because it is a numeric expression. Recall that each axis in an MDX query is evaluated independently. Thus, the context at the beginning of the execution (using the above logic) of the ORDER function is the tuple ([Qtr2], [Measures], [Product], [Market], [Scenario]) where Qtr2 is obtained from the slicer clause.

In the first iteration the [Product] is replaced in context with [100-10], the first member in the set. Thus the tuple for the first iteration of ORDER is ([Qtr2], [Measures], [100-10], [Market], [Scenario]).

The numeric value expression is evaluated with this context, and thus the expression translates into the retrieval of the data point ([Qtr1], [Measures], [100-10], [Market], [Actual]).

The second iteration starts with the same MDX Context as the first iteration, namely, ([Qtr2], [Measures], [Product], [Market], [Scenario]). This time, [100-20] will replace [Product] to form the new MDX context.

So the expression evaluation will use ([Qtr2], [Measures], [100-20], [Market], [Scenario]) as the context and thus the expression translates into retrieval of the data point ([Qtr1], [Measures], [100-20], [Market], [Actual]).

The ORDER function finally sorts input tuples based on these values and arranges these tuples in ascending order (the default order).

All other iterative functions operate similarly. After evaluating the expression, the action taken depends on the function. In case of ORDER, the action is to sort tuples in input set based upon values evaluated. In case of SUM, the action is to add up all the value. In case of GENERATE the action is to create an output set by performing a “UNION” of all sets.

Continuing the discussion on current context, it is worthwhile to point out a special function called CURRENTMEMBER (which is no different from the corresponding calculator language function) which takes as input a dimension and is used to refer to the member present in the current MDX context during the course of execution.

The calculated member evaluation is like any other numeric expression evaluation. Depending on the context in which such a member is used, it can be resolved either at axes resolution time or during the final retrieval stage. For example, consider the query:

To return ([100-10], [Revised Budget]) cell value, the expression ‘Budget \* 0.5’ will be evaluated in the context of ([Qtr1], [Sales], [100-10], [East], [Scenario]).

To return ([100-20], [Revised Budget]) cell value, the same expression will be evaluated in the context of ([Qtr1], [Sales], [100-20], [East], [Scenario]).

Next, we explore the notion of SOLVE\_ORDER, a construct that enables removal of potential ambiguities in a query execution.

---

M D X C A L C U L A T E D M E M B E R A N D  
S O L V E \_ O R D E R

---

A SOLVE\_ORDER clause can be specified following the definition of a calculated member to disambiguate the order of evaluation of calculated members in a query that has more than one such definition. We illustrate the use of such a clause by means of an example.

Q1 and M are two calculated members defined on Year and Measures dimensions respectively in the following query.

```
With
Member [Scenario].[Revised Budget] as 'Budget * 0.5'
:
:
SELECT
{Actual, [Revised Budget]} on Columns,
Children ([100]) on Rows
FROM (Qtr1, Sales, East)
```

```
WITH
MEMBER [Year].[Q1] as 'Jan + Feb + Mar'
MEMBER [Measures].[M] as 'Margin * 100 / Sales'
SELECT
{M} on Columns,
{Q1} on Rows
Where (Product, Market, Actual)
```

Product	Market	Actual	
	Margin	Sales	M = Margin * 100/Sales
Jan	17,378	31,538	55.1
Feb	17,762	32,069	55.38
Mar	17,803	32,213	55.26
Q1 = Jan + Feb + Mar	52,943	95,820	???

We have a question as to whether to calculate the (M, Q1) cell by the M formula or by the Q1 formula.

Let us compute the data value at the intersection (Q1, M, Product, Market, Actual) using the query execution rules we have learned thus far. There are two potential ways in which the data value can be computed: , using the definition of the calculated member Q1 or the definition of M.

If the formula for Q1 is evaluated first, then the result will be calculated as follows:

$$\begin{aligned} &\text{Value of (Q1, M, Product, Market, Actual)} = \\ &\text{Value of (Jan, M, Product, Market, Actual)} + \\ &\text{Value of (Feb, M, Product, Market, Actual)} + \\ &\text{Value of (Mar, M, Product, Market, Actual)} \end{aligned}$$

In this case, the value would be calculated as  $55.1 + 55.38 + 55.26 = 165.74$

If, however, the formula for M is evaluated first, then the result will be calculated as follows:

$$\begin{aligned} &\text{Value of (Q1, M, Product, Market, Actual)} = \\ &\text{Value of (Q1, Margin, Product, Market, Actual)} * 100 / \\ &\text{Value of (Q1, Sales, Product, Market, Actual)} \end{aligned}$$

In this case, the value would be calculated as  $52943 * 100 / 95820 = 55.25$

As can be seen from the above two examples, there is a need for disambiguation and the solve order clause enables override of the calculation order to achieve the desired end result. In the presence of SOLVE\_ORDER clause, Formula corresponding to the member with highest solve order will be evaluated first. So, if the above query were altered as follows,

```
WITH
:
MEMBER [Year].[Q1] as 'Jan + Feb + Mar', SOLVE_ORDER = 10
MEMBER [Measures].[M] as 'Margin * 100 / Sales', SOLVE_ORDER = 20
:
```

then the result will be 55.25, which is desired in this example.

The default SOLVE\_ORDER of a calculated member is assumed to be 0.

If multiple members at the intersection have the same SOLVE\_ORDER (or no explicit SOLVE\_ORDER clause is present), for example,

```
:  
MEMBER [Year].[Q1] as 'Jan + Feb +  
Mar', SOLVE_ORDER = 10  
MEMBER [Measures].[M] as 'Margin *  
100 / Sales', SOLVE_ORDER = 10  
:
```

then the tie is resolved using the order of respective dimensions as they appear in the Outline.

Thus, in the above example, since the Year dimension appears before Measures in the Outline, the formula for Q1 will be evaluated first and the resulting value will be 165.74. If you run the example query without specifying the SOLVE\_ORDER, both the members are assumed to have SOLVE\_ORDER 0 and the tie has to be resolved using dimension information. Hence, the result will be the same in previous case, 165.74.

---

#### MDX CALCULATED MEMBERS AND SECURITY

---

Recall that MDX supports calculated members to define dynamic calculations.

Calculated members are similar to custom formulas that can be attached to dynamic-calc members defined in the outline. The advantage of using a calculated member is that no outline

change is required, nor is the cube structure impacted to execute the custom calculation. While calculated members are functionally similar to custom member formulas, there is an important difference in data access privileges between the two.

In the case of dynamic-calc members, the administrator defines the member and the calculation formula associated with the member. Since the administrator is also responsible for enforcing user-level metadata and data security through appropriate filter definitions, it is assumed that the administrator is cognizant of security requirements for the dynamically calculated members as well. Thus, the formula associated with a dynamically calculated member does not enforce metadata or data security-members and cell data which is otherwise hidden from the user can be used in the calculation.

In MDX, in contrast, the end-user can define the calculation in a query. The administrator may not know about the formula and the dependent data-accesses during query execution. Therefore, the MDX implementation ensures that execution of a calculated member formula only uses data and metadata that the user has access to. The value of any data-point referred by the formula execution that the user has no access to is assumed to be MISSING.

For example, while evaluating a calculated member formula, if the Budgeted Sales for Qtr2 is not accessible to the user (suppressed via a data security filter), it will be assumed to be MISSING.

---



---

MDX AND MEMBER ORDERING

---

Functions such as <IDesc and <Gen in report writer and Descendants() and Members() in MDX return a collection of members. Between a report script and an equivalent MDX query, there is a difference in the order in which members are returned in the result set. Whereas the report writer interface returns members of the dimension tree in an order where all children of members appear before the parent, MDX metadata-based set-functions return members in an order where the parent appears before the children. The MDX function Hierarchize (<set>, POST) allows you to reorder the elements of <SET> in post-order form if desired.

---

MDX AND DYNAMIC TIME  
SERIES MEMBERS SUPPORT

---

Report writer and other interfaces like Grid-API use dynamic time series (DTS) members defined in the outline along with another time dimension member that specifies the latest time-period. MDX does not recognize DTS members defined in the Essbase outline directly. However, MDX can return the same value as returned by requests such as Q-T-D(Aug) (i.e. sum of the values up to month of August in the current Quarter) through functions such as QTD(), YTD(), etc.. These functions take the time dimension member (that one would specify to identify the latest time period in the report writer interface) as an argument. Internally, MDX uses the generation number for the corresponding Essbase member Q-T-D/Y-T-D. For example, the MDX expression QTD(Aug) returns the set of all time dimension members of the generation of Aug up to and

including Aug, that fall in the quarter that includes Aug.

We can define a calculated member to return sum/min/max/average of the values for the set returned by the set expressions QTD(), YTD(), etc. Defining a calculated member to Sum over the set returned by MDX QTD() function is logically equivalent to Q-T-D followed by <LATEST latestmember syntax in report writer. For example:

WITH

```
MEMBER Measures.[QTD Margin] AS
'Sum (QTD(), Margin)' ...
```

You can alternately use PeriodsToDate (<Layer>, <member>) in place of xTD() functions. For example, if you have a generation named Quarter already in your database, then PeriodsToDate (Quarter, Aug) is equivalent to QTD (Aug). Note that PeriodsToDate works on any dimension whereas the xTD() functions can work only on a Time dimension.

---

MDX AND SORTING

---

There are major differences in the way sorting commands work in MDX and report writer. In report writer, there are different commands to sort the members of a dimension and the rows of a report. In MDX, there is one function ORDER (<set>, <expression>) that performs all sorting. It evaluates the input value expression iteratively for every tuple in the set, and the expression can provide names, generation or level numbers, property or cell values to compare. Furthermore, sorting on multiple criteria can be carried out by nesting calls to Order. For example, the following report snippet and MDX snippet are logically equivalent:

```
<SORTGEN
<IDESCENDANTS Market
```

```
ORDER (DESCENDANTS (Market),
Market.CURRENTMEMBER. GEN_NUMBER)
```

However, the report writer sorting commands are stateful. That is, once a sort command is specified in a script, the sorting requirement will be applicable to all subsequent member selection commands in the script. The MDX Order() function, on the other hand, is applied to the input set to the function, and the result of the Order() function is a sorted set. Any other set expressions appearing in the same MDX query will not be effected by the presence of a prior ORDER command. Thus while <SORTNONE functionality is required in report writer language, such a command is meaningless in MDX.

Secondly, report writer sorting commands are applied to the members added with a member command such as <Children, <Descendants, etc. The MDX Order() function works for the entire input set, regardless of how it was created. For example,

```
<SORTMBRNames
Nov
<Children Qtr1
```

in report writer returns members in the following order: Nov, Feb, Jan, Mar.

Whereas

```
Order ({ Nov, Children(Qtr1)},
Year.CurrentMember.Member_name)
```

returns the members in the following order: Feb, Jan, Mar, Nov

The MDX expression

```
{ Nov, Order (Children (Qtr1),
Year.CurrentMember.Member_name) }
```

returns the members in the order{Nov, Feb, Jan, Mar}, since the Order() function is only applied to the children of Qtr1.

MDX AND ADVANCED FILTERING OPTIONS

We will compare report writer commands such as RESTRICT/TOP/BOTTOM /ORDERBY that are used to accomplish filtering with corresponding functionality in MDX.

The report writer retrieves all the data values as specified in the query and then executes the data- related filtering commands on the result. In other words, these commands are post-processing operations in report writer.

On the other hand, equivalent MDX functions are executed as a part of the individual axis processing. Since each of the axis is processed independently, the context of the execution may not be intuitive from report writer user's perspective. Context processing rules described in section <> apply for MDX execution.

Consider a simple example,

```
SELECT
CrossJoin ({Qtr1, Qtr2}, {Sales})
on Columns,
TopCount (Market.Levels(0).Members,
5) on Rows
From Sample.Basic
```

The set definition along rows will be executed independently of the contents of what is specified on Columns. Since the <numeric

value expression>, which is used to identify the top 5 members along Markets, is not specified, TopCount function will retrieve the value using the current context of execution. In this particular case, the root member along Measures dimension will be used for evaluation, which is different from Sales specified on the columns.

Secondly, each of these report writer commands can be used once in a script in effect. If two or more different commands (TOP/BOTTOM and ORDERBY) are specified, then all of them have to work on the same row dimension grouping. On the other hand, the equivalent MDX functions have no dependency among each other. For example, TOPCOUNT implementation has no assumption about the operations/functions used to create the input set.

Lastly, the report writer TOP/BOTTOM and ORDERBY commands work on a group of members from the innermost row dimension by default, though the grouping dimension can also be input explicitly. The equivalent MDX functions TOP(/BOTTOM)COUNT or ORDER have no such assumption about the dimension grouping. For example, the following report will get Top 5 Products for every market.

```
<Sym
<Column (Scenario, Year)
Actual Budget
Jan Dec
<Top ( 5, @DataCol(4))
<Row (Market, Product)
<children Market
<Idescendants Product
!
```

Whereas the MDX expression

```
TopCount (
    CrossJoin (Children (Market),
Descendants (Product)),
    5,
    (Budget, Dec)
)
```

will return only 5 tuples of (Market, Product) combinations.

In order to get the same result as the above report script i.e. top 5 Products for every Market member, use Generate function as shown in the following MDX query:

```
SELECT
CrossJoin ( {Actual, Budget},
{Jan, Dec}) on Columns,
Generate (
    Children (Market),
    CrossJoin (
        {Market.CurrentMember},
        TopCount (
            Descendants (Product),
            5,
            (Budget, Dec)
        )
    )
)) on Rows
```

Measures		Actual	Actual	Budget	Budget
		Jan	Dec	Jan	Dec
East	Product	1732	2037	2080	2120
East	100	924	1026	960	990
East	100-10	837	867	860	830
East	400	466	501	600	580
East	200-40	267	383	310	400
West	Product	2339	2448	2980	2710
West	300	755	971	830	950
West	200	752	820	850	860
West	Diet	663	629	850	730
West	100	378	223	830	530
South	Product	997	1141	1330	1270
South	100	329	432	540	640
South	100-10	244	327	370	460
South	Diet	355	404	490	430
South	200	480	496	520	390
Central	Product	2956	3154	3550	3570
Central	Diet	1080	1064	1340	1300
Central	200	751	753	1060	1220
Central	100	724	792	900	890
Central	300	790	824	930	810

The Generate function will iterate through each child of Market, and for each of those will combine (using CrossJoin) that market child with the top 5 descendants of Product.

---

MDX AND SUPPRESSION OF MISSING VALUES  
IN OUTPUT

---

Any multidimensional query language must have first class support to handle processing of missing or non-existent cell values.

The SUPMISSINGROWS command in report writer suppresses the display of all rows that contain only #MISSING values.

The equivalent construct in MDX is the NON EMPTY clause, which, if present on an axis specification will result in suppression of the tuple(s) that contain only #MISSING values for all possible combination of tuples from other axes.

SUPMISSINGROWS only works for dimensions formatted to appear along rows, whereas the NON EMPTY clause can be specified for any axis. For example, NON EMPTY on columns will check and suppress, if needed, tuples present along the columns. In addition, NON EMPTY can be specified for any combination of the axes present in the query.

Secondly, SUPMISSINGROWS logic works on every page in the report. On the other hand, NON EMPTY functionality logically works on all possible member combinations on all axes.

Consider an example,

```
<Page (Market)
<Children Market
<Row
<Idescendants Product
{SUPMISSINGROWS}
<Column (Scenario, Measures)
Actual Budget
Sales      COGS Profit
!
```

The SUPMISSINGROWS will cause display of only the eligible Products per Market page in the report. (It may so happen that one product suppressed in one page (market) is displayed on another page (market), depending on the data distribution). By contrast, NON EMPTY logically takes into account all possible member combinations on other axes.

For example:

NON EMPTY will suppress the products missing for every (Scenario, Measure) combination specified in the query for all the markets. It may so happen that one product has all missing values for some markets, but not the others. Such a product is not suppressed from the output as the product has at least one non-MISSING value for at least one market.

Conceptually, the Report writer output can be viewed as outputting a multi-cube structure, whereas the MDX output is a hypercube.

```
SELECT
Crossjoin ({Actual, Budget}, {Sales, COGS, Profit}) on Columns,
NON EMPTY Descendants (Product) on Rows,
Children (Market) on Pages
From Sample.Basic
```

APPENDIX

REPORT LAYOUT COMMANDS

REPORT WRITER	MDX	COMMENTS
<p>&lt;COLUMN &lt;PAGE &lt;ROW</p>	<p>On COLUMNS On ROWS On PAGES On CHAPTERS On SECTIONS On AXIS (n) where n is the axis number</p>	<p>Report writer supports up to 3 axes. MDX supports up to 64 axes. The first five axes are named as COLUMNS (0), ROWS (1), PAGES (2), CHAPTERS (3) and SECTIONS (5) respectively. The axes numbering increases from faster to slower moving axes.</p> <p>Operations like suppress missing or filtering can only be done on member combinations along Row in report writer.</p> <p>MDX treats all axes uniformly in the sense that all such operations are valid for sets on any axis.</p>
<p>&lt;ASYM &lt;SYM</p>		<p>MDX lets you define the set in any way you want.</p> <p>For example, the MDX set expression “{(Actual, Sales), {Budget, Profit}}” is logically equivalent to the report-script snippet “&lt;ASYM Actual Budget Sales Profit”.</p> <p>Whereas “CROSSJOIN ({Actual, Budget}, {Sales, Profit})” along columns will produce a symmetric output like report snippet “&lt;SYM Actual Budget Sales Profit”.</p> <p>MDX does not need to assume symmetric/asymmetric columns, as it deals with the way the set is defined.</p>

[ A HYPERION WHITE PAPER ]

MEMBER SELECTION COMMANDS		
REPORT WRITER	MDX	COMMENTS
<ALLINSAMEDIM mbrName	[mbrName].DIMENSION.members or [mbrName].Members	
<ALLSIBLINGS mbrName	[mbrName].Siblings or Siblings ([mbrName])	
<ANCESTORS mbrName		No equivalent MDX construct available in 7.0
<ATTRIBUTE attMbrName	Attribute ( [attMbrName] )	
<CHILDREN mbrName		
<CURRENCY targetCurrency	[mbrName].Children or Children ([mbrName])	No equivalent MDX construct available in 7.0
<DESCENDANTS mbrName	Descendants ([mbrName], [mbrName].GENERATION, AFTER)	Please see the section on MDX member Ordering.
<DIMBOTTOM mbrName	[mbrName].DIMENSION.Levels(0).Members	
<DIMTOP mbrName	Ancestor (mbrName, mbrName.Dimension.Generations(1))	
GEN name, dimension LEV name,dimension	[name].Members	In case of MDX, the name refers to user defined Generation / Level Name. Default generation/level names like Gen0, Gen1, Lev0, Lev1 are not supported in MDX. Use dimension.Generations(genNumber).Members Or dimension.Levels(levelNumber).Members syntax to select members of a particular generation or level.
<GEN(dimension,genNumber) <LEV(dimension,levNumber	dimension.GENERATIONS( genNumber ).Members dimension.LEVELS (levNumber).Members	
<IANCESTORS mbrName		No equivalent MDX construct available in 7.0
<ICHILDREN mbrName	{CHILDREN ([mbrName], mbrName )	
<IDESCENDANTS mbrName	Descendants([mbrName])	Please see the section MDX and Sorting.
<IPARENT mbrName	{[mbrName], [mbrName].PARENT}	
<LATEST mbrName		MDX does not support D-T-S members in ESSBASE. Please see the section MDX and Dynamic Time Series Members Support.
<LINK (extractionCommand [operator extractionCommand])	UNION, INTERSECTION, EXCEPT, EXTRACT	
<MATCH ("Member" "Gen" "Level","Pattern")		No equivalent MDX construct available in 7.0
<OFSAMEGEN mbrName	[mbrName].GENERATION.Members	
<ONSAMELEVELAS mbrName	[mbrName].LEVEL.Members	
<PARENT mbrName	[mbrName].PARENT	
<TODATE (formatString, dateString)	TODATE (formatString, dateString)	
<UDA (dimName, udaStr)	UDA (dimName, udaStr)	
<WITHATTR (dimName, "operator", value)	WITHATTR (member, STRING_LITERAL, [ signed_numeric_literal   character_string_literal ] )	

SORTING COMMANDS		
REPORT WRITER	MDX	COMMENTS
<SORTNONE		Equivalent MDX construct not required.
<SORTASC <SORTDESC <SORTGEN <SORTLEVEL <SORTMBRNames <SORTALTNames	ORDER (<set>, <value_expression>, BASIC   BDESC )  The value expression has to be written using <dimension>.CURRENTMEMBER.property Where property can be GEN_NUMBER LEVEL_NUMBER MEMBER_NAME, MEMBER_ALIAS	Please see the section MDX and Sorting

DATA ORDERING AND DATA RANGE COMMANDS		
<i>REPORT WRITER</i>	<i>MDX</i>	<i>COMMENTS</i>
<ORDERBY	ORDER (<set>, <numeric_value_expression>, BASC BDESC)	Please see the section MDX and Advanced Filtering Options.
<RESTRICT	FILTER (<set>, <conditional_expression>)	Please see the section MDX and Advanced Filtering Options
<TOP	TOPCOUNT ( set , index [,numeric_value_expression ] )	Please see the section MDX and Advanced Filtering Options
<BOTTOM	BOTTOMCOUNT ( set , index [,numeric_value_expression ] )	Please see the section MDX and Advanced Filtering Options

FORMAT COMMANDS		
REPORT WRITER	MDX	COMMENTS
SUPMISSINGROWS	NON EMPTY	Please see the section MDX and Suppression of Missing Values in Output
All other report writer format commands like WIDTH, ZERO-TEXT etc.		MDX does not support any formatting commands.

---

---

COLUMN OR ROW CALCULATION COMMANDS

---

<i>REPORT WRITER</i>	<i>MDX</i>	<i>COMMENTS</i>
All Column or Row Calculation commands		MDX does not support any such commands. Calculated members can be used to perform custom calculations in query.

---

DATA ORDERING AND DATA RANGE COMMANDS		
<i>REPORT WRITER</i>	<i>MDX</i>	<i>COMMENTS</i>
OutMbrAlt		Use the Api EssMdxSetMbrldType and specify the property MEMBER_NAME or MEMBER ALIAS in DIMENSION PROPERTIES clause for every axis. The current alias table will be used to return the alias.
OutAltMbr		Use the Api EssMdxSetMbrldType and specify the property MEMBER_NAME or MEMBER ALIAS in DIMENSION PROPERTIES clause for every axis. The current alias table will be used to return the alias.
OutAlt		Use the Api EssMdxSetMbrldType to set ESS_MDX_MEMBERIDTYPE_ALIAS. The current alias table will be used to return the alias.
OutAltNames		Not needed in MDX.
OutAltSelect		No equivalent MDX construct available in 7.0.

---

---

**FOOTNOTES**

- 1 Note that Hyperion also supports the production of MDX results in XML format through its support for XML for Analysis (XMLA). XMLA opens up new possibilities in the integration of Essbase-held information in an enterprise. This paper will not discuss XMLA; please refer to the XML for Analysis API Reference in the Essbase Deployment Services technical documentation.

---

---

## ABOUT HYPERION

---

FOR MORE INFORMATION, VISIT HYPERION AT [WWW.HYPERION.COM](http://WWW.HYPERION.COM)

---

Hyperion is the global leader in Business Performance Management software. More than 9,000 customers – including 91 of the FORTUNE 100 – rely on Hyperion software to translate strategies into plans, monitor execution and provide insight to improve financial and operational performance. Hyperion combines the most complete set of interoperable applications with the leading business intelligence platform to support and create Business Performance Management solutions. A network of more than 600 partners provides the company's innovative and specialized solutions and services.

Named one of the FORTUNE 100 Best Companies to Work For 2004, Hyperion employs approximately 2,600 people in 20 countries. Distributors represent Hyperion in an additional 25 countries. Headquartered in Sunnyvale, California, Hyperion – together with recently acquired Brio Software Inc. – generated combined annual revenues of \$612 million for the 12 months ending June 30, 2003. Hyperion is traded under the Nasdaq symbol HYSL. For more information, please visit [www.hyperion.com](http://www.hyperion.com), [www.hyperion.com/contactus](http://www.hyperion.com/contactus) or call 800 286 8000 (U.S. only).



**HYPERION** 1344 CROSSMAN AVENUE SUNNYVALE, CA 94089

---

TEL 408.744.9500

[WWW.HYPERION.COM](http://WWW.HYPERION.COM)

FAX 408.744.0400