**MDX Tutorial (Gentle Introduction)**

This tutorial describes MDX and its main concepts. It is meant to be a gentle introduction and is targeted to any person who wants to get a quick grasp of MDX capabilities.

**Overview**

MDX stands for 'Multi-Dimensional Expressions' and is the standard language defined by Microsoft to query OLAP servers.

At first glance, it may appear similar to SQL. However, MDX is a completely new language. SQL was designed to query dimensional data structures, called tables, where data are organized in rows and columns. In OLAP, data are organized around multiple measures, dimensions, hierarchies, and levels.

MDX is a language used to perform calculations and analysis around OLAP structures. MDX includes a rich set of functions for 'performing' statistical analysis. Unlike SQL, MDX does not have DDL (Data Definition) or DML (Data Manipulation) capabilities. OLAP structures are defined and modified in XMLA (XML for Analytics). MDX is purely for analyzing and reading data.

As OLAP servers are mainly using relational databases as their sources of data, we will sometimes use SQL concepts to describe functionality.

Similar to the way tables and columns are central to SQL, dimensions, hierarchies, and levels, are the centerpieces of MDX. They are mapping business models into language-specific concepts (e.g. a list of countries will be mapped as an MDX dimension).

The most natural way to explain these concepts is with an example.

Let's introduce a classic sales related business problem. We would like to generate some charts on our sales department and the information we have is as follows:

* Pierre Dupont sold one corporate license 2009 Q1 in Geneva
* Pierre Dupont sold one corporate license 2009 Q3 in Geneva
* Pierre Dupont sold one corporate license 2010 Q1 in Paris
* Rosa Maza sold one corporate license 2009 Q2 in Madrid
* Rudolf von Richthofen sold one partnership license 2009 Q1 in Zurich
* Rudolf von Richthofen sold one partnership license 2009 Q3 in Zurich
* John Bin sold one corporate license 2009 Q2 in New York
* Patty Bing sold one corporate license 2009 Q4 in Los Angeles

With this example we've introduced the concepts of 'Sales People', 'License', 'Time', 'City', and 'amount of licenses'. In a relational database we could model this with one table per concept and foreign keys in a 'amount\_of\_licenses' table.

In MDX 'Sales People', 'License', 'Time', and 'City', will be modeled as dimensions and the 'amount of licenses' as a measure. Measures in MDX are a special case of dimensions and they hold numeric values.

The concrete value of a dimension (e.g. Paris) is a member of the dimension and is similar to the value of an SQL table.

An OLAP cube, then, is a collection of *dimensions* indexing a list of *measures*.

You should not try to visualize the OLAP cube as a geometrical cube. A geometrical cube can have only three dimensions, whereas an OLAP cube can have many dimensions. In our example above, we have four dimensions ('Sales People', 'License', 'City', and 'Time') and one numeric measurable quantity ('License Quantity Sold'). Of course, real-life business problems can have many more dimensions and measures.

To start, let's look at an MDX statement that gives us a table with the amount of licenses sold per country per year:

SELECT

([Geography].[Geo].[Country].members) ON 0,

([Time].[Year].members ) ON 1

FROM

[Sales]

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Spain** | **Switzerland** | **France** | **United States** | **Canada** | **Mexico** | **Venezuela** |
| **2010** | 3.0 | 248.0 | 4 | 768.0 |  |  |  |
| **2011** |  |  |  |  |  |  |  |

As we would like to compute some statistics on this data, we will enrich the information with additional details.

We would like to organize the countries as a geographical tree (Continent, Country, City) and economic partnerships (EU, NAFTA, None).

For the time dimension, we're interested in years and quarters.

## Dimensions, Hierarchies, and Levels

Dimensions, attributes, hierarchies, and levels, are the way we define our business model in MDX. They represent a meta-definition of our data, similar to tables and columns in SQL.

Dimensions represent our main business concepts and are a generalization of concrete entities (Geography, Time, or Products). Attributes are used to capture relevant details about the dimension. For example, color, category, sub-category, price, and size, are all attributes which are used to capture the details of the Dimension Product. Similarly, date, month, year, hour, minute, and seconds, are the attributes used to capture the details of the Time dimension.

Attributes have relationships with one another. A day has 24 hours, an hour has 60 minutes, and a minute has 60 seconds. When the attributes are organized in order to represent their relationship with one another, a hierarchy is formed.

Hierarchies are not specific to time dimension. A continent has countries, a country has states, and states have cities.

One of the key tasks when doing dimension modeling is to identify the attributes which are needed to fully capture the data related to a dimension. And to identify how those attributes relate to one another.

It is not necessary for a dimension to have a single hierarchy. In common business cases you will see that dimensions can contain many hierarchies.

As we would like to generate some reports taking into account the geographical as well as the economic properties of the countries, we will define two separate hierarchies (i.e. geographical and economical) within that dimension.

For the geographical hierarchy, [Geography], we want to see the continents, countries, and cities. This directly defines the three levels of our hierarchy, [Continent], [Country], and [City].

For the economic hierarchy, [Economy], we want to see the economic partnerships (EU, NAFTA) and the countries. This defines two levels of our hierarchy, [Partnership] and [Country].

Both hierarchies do not need to define the same number of levels nor share the same levels. In our example, we could have the economic hierarchy without the country level and/or have two of the same country in two economic partnerships.

Using MDX syntax, we will represent this structure:

Dimension

[Geography]

Hierarchy

[Geography].[Geo] and [Geography].[Economy]

Levels

[Geography].[Geo].[Continent],

[Geography].[Geo].[Country],

[Geography].[Geo].[City],

[Geography].[Economy].[Partnership],

[Geography].[Economy].[Country]

**Members, Tuples, and Sets**

Members represent the concrete data of this meta-definition (e.g. Switzerland as a country). Each level of a given hierarchy will 'produce' a member with a parent-child relationship.

The object representing the upper level of Switzerland (continent) is also a member in MDX:

[Geography].[Geo].[Europe]: is a continent (with all European countries as children)

[Geography].[Geo].[Europe].[Switzerland]: is a country whose parent member is Europe )

[Geography].[Geo].[Europe].[Switzerland].[Geneva]: is a city (whose parent member is Switzerland)

Representing the full hierarchy on rows as a tree, would appear as follows:

Select

[Measures].[Amount] on 0,

[Geography].[Geo].members on 1

From

[Sales]

|  |  |
| --- | --- |
|  | **Amount** |
| **Europe** | 255.0 |
| **Spain** | 3.0 |
| **Madrid** | 1 |
| **Barcelona** | 2 |
| **Valencia** |  |
| **Switzerland** | 248.0 |
| **Lausanne** | 56.0 |
| **Geneva** | 128 |
| **Zurich** | 64 |
| **France** | 4 |
| **Paris** | 4 |
| **America** | 768.0 |
| **United States** | 768.0 |
| **New York** | 768.0 |
| **San Francisco** |  |
| **Los Angeles** |  |
| **Canada** |  |
| **Quebec** |  |
| **Toronto** |  |
| **Mexico** |  |
| **Mexico** |  |
| **Venezuela** |  |
| **Caracas** |  |

A tuple is the intersection of one (and only one) member taken from each of the dimensions in the cube. A tuple identifies a single cell in the multi-dimensional matrix.

The easiest way to understand a tuple is to look at an Excel sheet. In the Excel sheet a cell is identified by A1, A2, A3, B1, B2, B3, etc. Here A represents a coordinate from the columns and 1 represents a coordinate from the rows. We put these together and we can uniquely identify a cell in the sheet. Here A1, B1 are examples of tuples. In a OLAP cube there are many dimensions and a tuple is defined as a list of a single hierarchy member taken from all the dimensions (e.g. [New York], [2009]). The net effect is the same. The tuple always points to a single cell in the OLAP cube.

When defining our business case, we defined several dimensions and one measure. Those dimensions can be seen as discrete coordinates (discrete as they have a limited amount of possibilities) of a cube. Just as 'x,y,z' represent the coordinates of points in space, members of our dimensions represent the coordinates in our cube. The measure being the value of this cube.

For example, 'Patty Bing sold one corporate license 2009 Q4 in Los Angeles' is represented by the tuple:

([People].[Patty Bing], [Prod].[License].[Corporate], [Time].[Calendar].[Q4 2009], [Geography].[Geo].[Los Angeles], [Measures].[Amount])

A tuple dimensionality is defined as the ordered list of the member's hierarchies. For example, the previous tuple dimensionality is:

( [People].[People], [Product].[Prod], [Time].[Calendar], [Geography].[Geo], [Measures] )

Within a hierarchy, intermediate members (a.k.a. parent members) do not represent actual coordinates of measures in our cube. Therefore, the MDX aggregation and calculation engine is going to compute those measures by aggregating the measures' values of their children (and grandchildren).

This means that ( [Patty Bing], [2009] ) is going to return the 'amount of licenses' sold by Patty during the whole year 2009 (aggregating the amount for each quarter of 2009).

Sets are an ordered collection of 0 or more tuples (note that a member is considered to be a tuple containing a single element) with the same dimensionality. Unlike a mathematical set, an MDX set may contain duplicates. So, the same member/tuple can be in a list twice. Sets are defined using curled brackets:

{ [Geography].[Geo].[Europe].[Switzerland], [Geography].[Geo].[Europe].[France] }

Members of a set must be from the same hierarchy. For example, the following set is invalid:

{ [Geography].[Geo].[Europe].[Switzerland], [Product].[License].[Corporate] }

The following set is invalid because of different tuple dimensionalities:

{ ( [Time].[Calendar].[2010], [Geography].[Geo].[Europe].[Switzerland] ),

( [Time].[Calendar].[2010], [Product].[License].[Corporate] ) }

## Member Name

Let's take a little break and look at the different ways we have of selecting members in MDX. One disruptive issue (when starting out as well as later when continuing on with MDX) is the fact that dimension, hierarchies, levels, and members, are mixed in the syntax.

What does this expression refer to, a dimension, a hierarchy, a level, or a member?

[Something].[What is This?]

The correct answer is that it could be any of the four. This may not be the clearest solution but this is the way it works.

Within MDX expressions, members are referred to using case-insensitiveidentifiers: Both Madrid and MADRID refer to the city Madrid within the Geography dimension.

When the name of a member contains reserved MDX characters (e.g. ]) and/or separators, then a delimited identifier has to be used. For example in [Swiss German]. The ']' character must be escaped: [product[0]]] is naming the member 'product[0]'.

Unique member names are used to uniquely refer to a member within the cube and its dimensions/hierarchies. A unique namemay be thought of as a path composed of local names.

A possible unique name is:

[Geography].[Geo].[City].[Madrid]

where Madrid has been referenced right from the City level.

Let's imagine that all the continentswere organized into an extra level representing the whole world. We could use the following unique name to refer to Madrid:

[Geography].[Geo].[All].[Europe].[Spain].[Madrid]

You'll notice that no level name has been specified.

## SELECT Statement

The SELECT statement/expression is the MDX way of querying multi-dimensional data, as described previously. One of its simplest forms could be as follows:

SELECT

[Measures].Members ON COLUMNS,

[Geography].[Geo].Members ON ROWS

FROM

[Sales]

Let's dissect this query in order to understand it better.

The SELECT keyword marks the beginning of the query. It specifies what you want to select.

The ON keyword is used to organize the selected data. MDX utilizes the concept of axes. The data selected from a dimension can be put on a axis. In the query above, we selected the members of the [Measure] dimension and put them on axis 0, or columns. And we selected the members of the [Geography] dimension and put them on axis 1, or rows. However, a SELECT is not limited to two axes. We could have columns, rows, pages, chapters, and sections. And you could still continue beyond these by specifying a number for the axis.

Note: A query that uses axis 1 must also use axis 0. A query that uses axis 2 must also use axis 1 and axis 0. You cannot skip an axis in a query.

In the FROM clause, we specify the cube which will be used for querying the data specified in the SELECT clause.

You may have a question regarding tuples when you read the query above. We have explained that a tuple is a list of hierarchy members taken from every dimension. However, in the query above, we are using members only from [Geography].[Geo] and [Time].[Calendar].[Year]. Shouldn't we take members from all the dimensions to create the tuple? The answer is that MDX uses the concept of a default member. Each hierarchy has a default member. This is usually the top of the hierarchy. If a member from a dimension is not specified, then the default member is automatically taken.

Just to elaborate some more on the concept of default members, the query below does not specify any member of any hierarchy and yet it is a perfectly valid MDX query. It will return the default [Measures] value indexed by the default members from all the other hierarchies:

SELECT FROM [Sales]

## WHERE (a.k.a. Slicer) Clause

The SELECT expression can also specify a WHERE clause in order to reduce the scope of a query. When a WHERE clause is not specified in a query, the entire cube space is taken into account when calculating results. When we do specify a WHERE clause the scope of the query is reduced. For example:

SELECT

[Measures].Members ON 0,

[Geography].[Geo].Members ON 1

FROM

[Sales]

WHERE

[Time].[Calendar].[Year].[2010]

In the query above, only that section of the cube where the year is 2010 is taken into account as the scope of the query. The rest of the cube is ignored. This is why the WHERE clause is called the 'slicer' - because it takes a 'slice' from the entire cube and performs calculations only on that slice of data.

To get more details about the slicer, several illustrated working examples are available from our MDX documentation pages [here](http://www.iccube.com/support/documentation/mdx/WhereClause.html).

## FROM as a Slicer

Besides the WHERE clause, MDX provides a second method of slicing the cube. This is the FROM clause. And the FROM clause is not limited only to a cube. A SELECT (a.k.a. sub-select, sub-query) can be used instead:

SELECT

[Measures].Members ON 0,

[Geography].[Geo].Members ON 1

FROM

(SELECT [Time].[Calendar].[Year].[2010] ON 0 FROM [Sales])

This is equivalent to the previous query slicing the results for the [Year].[2010]. Sub-selects/queries allow you to filter the dimensions (and the facts) of the cube to be queried.

Example:

|  |  |
| --- | --- |
| SubQueries | (standard MDX) |

MDX subQueries (aka. subSelect).

Description

SubQueries (or subSelects) are one of the latest addition to the MDX language. It is a powerful feature that allows for querying an MDX query instead of a cube.  
  
As opposed to the WHERE clause, subQueries also change the hierarchies structure. Hierarchies defined in a subQuery are filtered (accepting all descendants and ascendants members defined in the subQuery).  
  
See example 'SubQueries in Axis - Branch' for more information.

SELECT  
 [Measures].members ON 0  
 FROM [NON VISUAL]  
 ( SELECT [Geography].[Switzerland] ON 0 FROM [SALES] )

The 'NON VISUAL' flags control whether aggregation is using all members (NON VISUAL) or only the filtered members.

## Hierarchy Navigation

One of the most powerful features of MDX is that it is location aware. This makes navigating the hierarchy a simple task.

### Relative Cell Referencing

Consider a cell in the cube which contains the sales of icCube licenses in the year 2010. This cell is represented by the tuple:

( [Product].[Prod].[License], [Measures].[Amount], [Time].[Year].[2010] )

The MDX expression below gives the sales of licenses for the year 2009. In the expression below, we did not have to specify 2009 in the query. MDX automatically referenced the member 2009, which comes before 2010 in the year attribute:

{ [Product].[Prod].[License], [Measures].[Amount], [Time].[Year].PrevMember }

Similarly, we can navigate to the next member in the year level:

{ [Products].[Clothes], [Measures].[Amount], [Time].[Year].NextMember }

This offers great functionality because it allows you to write dynamic code which can navigate the hierarchy without you having to hard code the member names.

To further harness the power of relative referencing, you can use the feature called CurrentMember. This feature gives you the current location of a cell with reference to a hierarchy.

For example, instead of writing the first cell as ( [Product].[Prod].[License], [Measures].[Amount], [Time].[Year].[2010]), we can write it as ( [Product].[Prod].[License], [Measures].[Amount], [Time].[Year].CurrentMember). This acts as a pointer to the current cell and allows us to write dynamic MDX code. Dynamic MDC code can navigate the hierarchies without hard coding or knowing member names.

### Children

The Children function dynamically identifies the members that are one level below the current level:

[Product].[Prod].CurrentMember.Children

The MDX expression above takes the current member of the category's attribute and then extracts all child members of that attribute. This can also be written in a short hand format:

[Product].[Prod].Children

We can also identify the first and the last child by MDX expressions [Product].[Prod].FirstMember and [Product].[Prod].LastMember. And we can create a set from the child members using the following range expression:

{ [Product].[Prod].FirstChild : [Product].[Prod].LastChild }

### Parent

The Parent function is the opposite of the Children function. The Children function returns members who are under the current node. The Parent function will reference the node which is the parent of the current node.

Together these Children and Parent functions let you move up and down the hierarchy. And the PrevMember and NextMember functions let you move left and right in the hierarchy. This navigation capability, combined with the ability of MDX to determine its current position by CurrentMember, gives rise to a powerful query mechanism where the entire cube space can be navigated dynamically.

Functional Language Support

As you might have noticed in the previous example, an MDX expression may contain function calls. The syntax for a function call is either post-fixed as in the following example:

[Time].[Calendar].[Year].[2010].Children

and/or pre-fixed as in this example:

Head( [Time].[Calendar].[Year].[2010].Children, 3 )

Post-fixed function calls can be chained as follows:

[Time].[Calendar].[Year].[2010].PrevMember.Children

## Common Calculations

Let's study some common requirements, which occur in real-life projects, and apply what we have learned so far.

### Average

What are the average sales of Products, per quarter, for any given year?

Let's break this down into small pieces. In MDX, any given year will be represented by [Time].[Year].CurrentMember . Since we know that quarters come under years in a hierarchy of time, we can find the quarters which come under a particular year by using the descendants function:

Descendants( [Time].[Year].CurrentMember, [Time].[Quarter] )

The function above gives us all the descendants, at the quarter level, which come under a given year.

Averaging the sales across these quarters is also easy:

Avg( Descendants( [Time].[Year].CurrentMember, [Time].[Quarter] ), [Measures].[Amount] )

### Contribution to Total

What percentage of sales is allocated to one particular product as compared to all products?

First, get the sales of the particular product:

( [Measures].[Amount], [Product].[Prod].CurrentMember )

And now, get the sales of the parent. This can be done by:

( [Measures].[Amount], [Product].[Prod].Parent )

Calculating the contribution ratio is simply a matter of dividing the first figure by the second.

### Period To Date

What are the sale figures of a particular product from the beginning of the quarter up to a specific date?

The current date can be represented by [Time].[Calendar].CurrentMember. How do you get the date for the first day of the current quarter? By looking at the time hierarchy. We know that days roll into months and months roll into quarters. So, [Time].[Calendar].Parent.Parent, would take us to the Quarter node of the current date. Now, to get to the first day of this quarter, we simply have to use the descendants function and first child function as shown below:

Descendants( [Time].[Calendar].Parent.Parent, [Time].[Calendar].[Day] ).FirstChild

To get the sum of sales from the first day of a quarter up to a specific date, do as follows:

Sum( { Descendants( [Time].[Calendar].Parent.Parent, [Time].[Calendar].[Day] ).FirstChild : [Time].[Calendar].CurrentMember }, [Measures].[Amount] )

## Calculated Members

Calculated members allow you to define an MDX expression/formula that is attached to a new dimension's member. Note that a calculated member within this document is referring either to a calculated measures (i.e. a new [Measures] being calculated) or a calculated member (i.e. a new dimension member being calculated).

Calculated members are never aggregated and instead are always evaluated according to the current context.

Calculated members may be defined at cube level (they are then available as regular members for every SELECT query) but also within the scope of a given SELECT expression only as follows:

WITH

MEMBER [Measures].[Market Share] AS ([Measures].[Amount]) / ( [Product].CurrentMember.parent ,[Measures].[Amount] ), FORMAT\_STRING = "Percent"

SELECT

[Product].[icCube].Children ON 0,

[Measures].[Market Share] ON 1

FROM

[Sales]

As you can see above, there are three main parts of a calculated member.

Member Identifier: This specifies the name of the calculated member, the dimension to which the calculated member belongs, and its hierarchical positioning. Remember that [Measures] is also a dimension in MDX.

The main part of the definition is the formula which tells us how the results are derived.

Optional properties can provide additional information such as the display format of the values.

When calculated members are defined in the query by using the WITH clause (like the example above), the scope of the calculation is limited to the scope of the query. The MDX engine does not remember the member once the query execution is over.

With the new calculated measure being defined, each cell (indexed by a tuple containing that calculated measure) is going to be calculated using the MDX formula of the new calculated measure.

But what if the indexing tuple contains more than one calculated member? There has to be a way to define the calculated members' precedence. The SOLVE\_ORDER property is available for this purpose; the calculated member with the highest SOLVE\_ORDER is going to be evaluated first.

WITH

MEMBER [Measures].[Profit] AS [Measures].[Amount] - [Measures].[Cost], SOLVE\_ORDER = 0

MEMBER [Time].[Calendar].[Year].[2010/2009] AS [Time].[Calendar].[Year].[2010] / [Time].[Calendar].[Year].[2009], SOLVE\_ORDER = 1

SELECT

[Measures].[Profit] ON COLUMNS,

[Time].[Calendar].[Year].[2010/2009] ON ROWS

FROM

[Sales]

The example above is selecting a single cell indexed by ( [Time].[Year].[2010 / 2009], [Measures].[Profit] ). The SOLVE\_ORDER property will be used to determine which formula must be applied to get a consistent result. [2010 / 2009] having the highest SOLVE\_ORDER, is going to be evaluated as follows: ( [Profit], [2010] ) / ( [Profit], [2009] ).

Note that calculated members can reference other calculated members.

**Named Sets**

Similar to calculated members, a named set is an MDX expression defining the content of a set that can then be used wherever a set is expected. Named sets can be defined within the scope of a SELECT expression. Note that they're evaluated in the context of the slicer, if available:

WITH

SET [European Countries] AS [Geography].[Geo].[Europe].Children

SELECT

[Measures].[Amount] ON COLUMNS,

[European Countries] ON ROWS

FROM

[Sales]

The example above is selecting the amount of sales for each European country.

The named sets also have scope. When the set is defined in the query by using the SET clause, as shown above, the scope of the set is only limited to the lifetime of the query. When the query executes again, the set is re-calculated. When the SET is defined in the cube itself, it can be of either session or of global scope.

We've now reached the end of our gentle MDX introduction. The following pages describe in more detail each MDX function and several features of MDX.

* [Functions](http://www.iccube.com/support/documentation/mdx/functions.html)
* [Language](http://www.iccube.com/support/documentation/mdx/language.html)

# MDX Functions Reference

This section describes the standard MDX functions. Each function is illustrated with several working examples.

## Group : color

* [HSV](http://www.iccube.com/support/documentation/mdx/HSV.html)
* [InterpolateHSVColors](http://www.iccube.com/support/documentation/mdx/InterpolateHSVColors.html)
* [InterpolateRGBColors](http://www.iccube.com/support/documentation/mdx/InterpolateRGBColors.html)
* [RGB](http://www.iccube.com/support/documentation/mdx/RGB.html)

## Group : logical

* [CoalesceEmpty](http://www.iccube.com/support/documentation/mdx/CoalesceEmpty.html)
* [IsAncestor](http://www.iccube.com/support/documentation/mdx/IsAncestor.html)
* [IsEmpty](http://www.iccube.com/support/documentation/mdx/IsEmpty.html)
* [IsGeneration](http://www.iccube.com/support/documentation/mdx/IsGeneration.html)
* [IsLeaf](http://www.iccube.com/support/documentation/mdx/IsLeaf.html)
* [IsSibling](http://www.iccube.com/support/documentation/mdx/IsSibling.html)

## Group : member

* [Ancestor](http://www.iccube.com/support/documentation/mdx/Ancestor.html)
* [ClosingPeriod](http://www.iccube.com/support/documentation/mdx/ClosingPeriod.html)
* [Cousin](http://www.iccube.com/support/documentation/mdx/Cousin.html)
* [CurrentMember](http://www.iccube.com/support/documentation/mdx/CurrentMember.html)
* [DefaultMember](http://www.iccube.com/support/documentation/mdx/DefaultMember.html)
* [FirstChild](http://www.iccube.com/support/documentation/mdx/FirstChild.html)
* [FirstSibling](http://www.iccube.com/support/documentation/mdx/FirstSibling.html)
* [Lag](http://www.iccube.com/support/documentation/mdx/Lag.html)
* [LastChild](http://www.iccube.com/support/documentation/mdx/LastChild.html)
* [LastSibling](http://www.iccube.com/support/documentation/mdx/LastSibling.html)
* [Lead](http://www.iccube.com/support/documentation/mdx/Lead.html)
* [LinkMember](http://www.iccube.com/support/documentation/mdx/LinkMember.html)
* [LookupByKey](http://www.iccube.com/support/documentation/mdx/LookupByKey.html)
* [NameToSet](http://www.iccube.com/support/documentation/mdx/NameToSet.html)
* [NextMember](http://www.iccube.com/support/documentation/mdx/NextMember.html)
* [OpeningPeriod](http://www.iccube.com/support/documentation/mdx/OpeningPeriod.html)
* [ParallelPeriod](http://www.iccube.com/support/documentation/mdx/ParallelPeriod.html)
* [Parent](http://www.iccube.com/support/documentation/mdx/Parent.html)
* [PrevMember](http://www.iccube.com/support/documentation/mdx/PrevMember.html)
* [Properties](http://www.iccube.com/support/documentation/mdx/Properties.html)

## Group : numeric

* [Aggregate](http://www.iccube.com/support/documentation/mdx/Aggregate.html)
* [Avg](http://www.iccube.com/support/documentation/mdx/Avg.html)
* [Count](http://www.iccube.com/support/documentation/mdx/Count.html)
* [Count(Dimensions)](http://www.iccube.com/support/documentation/mdx/Count(Dimensions).html)
* [Count(Level)](http://www.iccube.com/support/documentation/mdx/Count(Level).html)
* [Count(Levels)](http://www.iccube.com/support/documentation/mdx/Count(Levels).html)
* [CurrentCellValue](http://www.iccube.com/support/documentation/mdx/CurrentCellValue.html)
* [Distinct](http://www.iccube.com/support/documentation/mdx/Distinct.html)
* [DistinctCount](http://www.iccube.com/support/documentation/mdx/DistinctCount.html)
* [Matrix](http://www.iccube.com/support/documentation/mdx/Matrix.html)
* [Max](http://www.iccube.com/support/documentation/mdx/Max.html)
* [Median](http://www.iccube.com/support/documentation/mdx/Median.html)
* [Min](http://www.iccube.com/support/documentation/mdx/Min.html)
* [Sum](http://www.iccube.com/support/documentation/mdx/Sum.html)
* [Value](http://www.iccube.com/support/documentation/mdx/Value.html)
* [Vector](http://www.iccube.com/support/documentation/mdx/Vector.html)
* [Wavg](http://www.iccube.com/support/documentation/mdx/Wavg.html)

## Group : other

* [FirstNotAllLevel](http://www.iccube.com/support/documentation/mdx/FirstNotAllLevel.html)
* [LastLevel](http://www.iccube.com/support/documentation/mdx/LastLevel.html)

## Group : set

* [AddCalculatedMembers](http://www.iccube.com/support/documentation/mdx/AddCalculatedMembers.html)
* [AllMembers](http://www.iccube.com/support/documentation/mdx/AllMembers.html)
* [Ancestors](http://www.iccube.com/support/documentation/mdx/Ancestors.html)
* [Ascendants](http://www.iccube.com/support/documentation/mdx/Ascendants.html)
* [BottomCount](http://www.iccube.com/support/documentation/mdx/BottomCount.html)
* [BottomPercent](http://www.iccube.com/support/documentation/mdx/BottomPercent.html)
* [BottomSum](http://www.iccube.com/support/documentation/mdx/BottomSum.html)
* [Children](http://www.iccube.com/support/documentation/mdx/Children.html)
* [CompactSet](http://www.iccube.com/support/documentation/mdx/CompactSet.html)
* [Crossjoin](http://www.iccube.com/support/documentation/mdx/Crossjoin.html)
* [Current](http://www.iccube.com/support/documentation/mdx/Current.html)
* [CurrentOrdinal](http://www.iccube.com/support/documentation/mdx/CurrentOrdinal.html)
* [Descendants](http://www.iccube.com/support/documentation/mdx/Descendants.html)
* [Except](http://www.iccube.com/support/documentation/mdx/Except.html)
* [Exists](http://www.iccube.com/support/documentation/mdx/Exists.html)
* [Extract](http://www.iccube.com/support/documentation/mdx/Extract.html)
* [Filter](http://www.iccube.com/support/documentation/mdx/Filter.html)
* [Generate](http://www.iccube.com/support/documentation/mdx/Generate.html)
* [Head](http://www.iccube.com/support/documentation/mdx/Head.html)
* [Hierarchize](http://www.iccube.com/support/documentation/mdx/Hierarchize.html)
* [Iif](http://www.iccube.com/support/documentation/mdx/Iif.html)
* [Intersect](http://www.iccube.com/support/documentation/mdx/Intersect.html)
* [LastPeriods](http://www.iccube.com/support/documentation/mdx/LastPeriods.html)
* [MeasureGroupMeasures](http://www.iccube.com/support/documentation/mdx/MeasureGroupMeasures.html)
* [Members](http://www.iccube.com/support/documentation/mdx/Members.html)
* [Mtd](http://www.iccube.com/support/documentation/mdx/Mtd.html)
* [NonEmpty](http://www.iccube.com/support/documentation/mdx/NonEmpty.html)
* [Order](http://www.iccube.com/support/documentation/mdx/Order.html)
* [PeriodsToDate](http://www.iccube.com/support/documentation/mdx/PeriodsToDate.html)
* [Qtd](http://www.iccube.com/support/documentation/mdx/Qtd.html)
* [Rank](http://www.iccube.com/support/documentation/mdx/Rank.html)
* [Siblings](http://www.iccube.com/support/documentation/mdx/Siblings.html)
* [StripCalculatedMembers](http://www.iccube.com/support/documentation/mdx/StripCalculatedMembers.html)
* [Subset](http://www.iccube.com/support/documentation/mdx/Subset.html)
* [Tail](http://www.iccube.com/support/documentation/mdx/Tail.html)
* [TopCount](http://www.iccube.com/support/documentation/mdx/TopCount.html)
* [TopPercent](http://www.iccube.com/support/documentation/mdx/TopPercent.html)
* [TopSum](http://www.iccube.com/support/documentation/mdx/TopSum.html)
* [Union](http://www.iccube.com/support/documentation/mdx/Union.html)
* [VisualTotals](http://www.iccube.com/support/documentation/mdx/VisualTotals.html)
* [Ytd](http://www.iccube.com/support/documentation/mdx/Ytd.html)

## Group : string

* [MemberToStr](http://www.iccube.com/support/documentation/mdx/MemberToStr.html)
* [SetToStr](http://www.iccube.com/support/documentation/mdx/SetToStr.html)
* [StrToHierarchy](http://www.iccube.com/support/documentation/mdx/StrToHierarchy.html)
* [StrToLevel](http://www.iccube.com/support/documentation/mdx/StrToLevel.html)
* [StrToMember](http://www.iccube.com/support/documentation/mdx/StrToMember.html)
* [StrToSet](http://www.iccube.com/support/documentation/mdx/StrToSet.html)
* [StrToTuple](http://www.iccube.com/support/documentation/mdx/StrToTuple.html)
* [StrToValue](http://www.iccube.com/support/documentation/mdx/StrToValue.html)
* [TupleToStr](http://www.iccube.com/support/documentation/mdx/TupleToStr.html)

## Group : tuple

* [Root](http://www.iccube.com/support/documentation/mdx/Root.html)

## Group : utility

* [Java Functions](http://www.iccube.com/support/documentation/mdx/Java%20Functions.html)

Example:

|  |  |
| --- | --- |
| HSV | (standard MDX) |

Create a color from its HSV values.

Parameters

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Cardinality** | **Default value** |
| hue | value-expression | one |  |
| saturation | value-expression | one |  |
| value | value-expression | one |  |

Return

color

Description

The function returns a color in the HSV color space; e.g., yellow-color = RGV( 60, 1.0, 1.0).  
  
The saturation and value components should be floating-point values between zero and one (numbers in the range 0.0-1.0). The hue component can be any floating-point number.

**Query**

WITH

MEMBER [Measures].[YELLOW] AS HSV( 60.0, 1.0, 1.0 )

SELECT

[Measures].[YELLOW] ON 0

FROM

[Sales]

# MDX Language Features

This section describes several aspects of the MDX language.

* [Annotations](http://www.iccube.com/support/documentation/mdx/Annotations.html)
* [Axis](http://www.iccube.com/support/documentation/mdx/Axis.html)
* [Calculated Members](http://www.iccube.com/support/documentation/mdx/Calculated%20Members.html)
* [Cell Properties](http://www.iccube.com/support/documentation/mdx/Cell%20Properties.html)
* [Comments](http://www.iccube.com/support/documentation/mdx/Comments.html)
* [Create (Function)](http://www.iccube.com/support/documentation/mdx/Create%20(Function).html)
* [Create (Member)](http://www.iccube.com/support/documentation/mdx/Create%20(Member).html)
* [Create (Set)](http://www.iccube.com/support/documentation/mdx/Create%20(Set).html)
* [Declared Functions](http://www.iccube.com/support/documentation/mdx/Declared%20Functions.html)
* [Drillthrough](http://www.iccube.com/support/documentation/mdx/Drillthrough.html)
* [Drop (Function)](http://www.iccube.com/support/documentation/mdx/Drop%20(Function).html)
* [Drop (Member)](http://www.iccube.com/support/documentation/mdx/Drop%20(Member).html)
* [Drop (Set)](http://www.iccube.com/support/documentation/mdx/Drop%20(Set).html)
* [Existing](http://www.iccube.com/support/documentation/mdx/Existing.html)
* [Implicit Conversions](http://www.iccube.com/support/documentation/mdx/Implicit%20Conversions.html)
* [Measure Group](http://www.iccube.com/support/documentation/mdx/Measure%20Group.html)
* [Range](http://www.iccube.com/support/documentation/mdx/Range.html)
* [SubQueries](http://www.iccube.com/support/documentation/mdx/SubQueries.html)
* [Update Cube](http://www.iccube.com/support/documentation/mdx/Update%20Cube.html)
* [Where Clause](http://www.iccube.com/support/documentation/mdx/Where%20Clause.html)
* [Where Clause Exists](http://www.iccube.com/support/documentation/mdx/Where%20Clause%20Exists.html)
* [Where Set Clause Exists](http://www.iccube.com/support/documentation/mdx/Where%20Set%20Clause%20Exists.html)

|  |  |
| --- | --- |
| Annotations | (icCube MDX extension) |

* Annotations allows for adding some sort of meta-data to MDX elements.
* Description
* icCube annotations are adding meta-data facility to the MDX elements. They can be placed in different parts of a SELECT statements to modify their behaviors. For example, the **evaluation cache** can be activated or deactivated using an annotation. The following examples are giving more details about existing annotations.
* **Query**
* //#NoCache
* WITH MEMBER [Measures].[foo] AS [Measures].[Amount]
* SELECT [Measures].[foo] ON 0 FROM [Sales]
* **Result**

|  |
| --- |
| **foo** |
| 1023 |

**Assertion : MDX Equals**

WITH MEMBER [Measures].[foo] AS [Measures].[Amount]

SELECT [Measures].[foo] ON 0 FROM [Sales]

|  |  |
| --- | --- |
| Axis | (standard MDX) |

About the usage of axes in a SELECT statement.

## Description

This section highlights some notable usages of the axes of a SELECT statement (e.g., auto-exists, ...).

# Examples

Non-existing tuples are not removed from an axis cross-product.  
  
In our example, even though only the tuple ([Geography].[Geo].[France], [Geography].[Economy].[France] )  
exists, all other countries are shown in the result.

**Query**

SELECT

{ [Geography].[Economy].[Country].members } on 0,

{ [Geography].[Geo].[France] } on 1

FROM

[Sales]

**Result**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | **France** | **Spain** | **Canada** | | **Mexico** | **United States** | | **Venezuela** | **Switzerland** |
| **France** | | 4 |  |  | |  |  | |  |  |
| Calculated Members | | | | (standard MDX) | | |

Calculated measures allows for defining new measures.

Description

A powerful feature of MDX is defining new measures, known as calculated measures (or calculated members).  
  
They are defined before the SELECT statement :

WITH  
 MEMBER [Measures].[Diff with United States] AS '[Measures].[Amount] - ([Geography].[United States],[Measures].[Amount])'  
 MEMBER [Geography].[Geo].[Europe I] AS '[Geography].[Geo].[Europe] - [Geography].[Geo].[Switzerland]'  
 SELECT  
 ...

In this example, the new measure allows for defining a value based on a formula. This formula may override any member from any hierarchy as defined by the evaluation context of the formula (e.g., axes, slicer, etc...).  
  
Similarly, calculated members can be declared (and removed) at schema level using the CREATE / DROP statements.

DROP MEMBER [Measures].[Benefit]  
 CREATE CALCULATED MEMBER [Measures].[Benefit] AS [Measures].[Sales] - [Measures].[Cost]

|  |  |
| --- | --- |
| Cell Properties | (standard MDX) |

Cell properties are defining some meta-information attached to measures and calculated members/measures.

Description

In MDX a cell can define not only a value (e.g., string, numerical value) but a list of predefined properties.  
  
Two steps are needed :  
  
1) Define the property in a calculated member and/or regular measure :  
  
Note that cell property definitions are not limited to MDX literals (e.g., string, numerical value). Indeed any valid MDX expression can be used as well:

WITH MEMBER [value] AS [Measures].[Amount],  
 BACK\_COLOR = IIF( currentCellValue() < 0, RGB(255,0,0), RGB(0,0,0) )

VALUE, FORMATTED\_VALUE, CELL\_ORDINAL properties are already defined and it's not possible to override the default  
behavior.  
  
2) Define the list of cell properties to be calculated in the MDX query :  
  
After defining a property the cell property value can be retrieved by using 'CELL PROPERTIES' statement in the mdx statement :

SELECT  
 [MyAxe] on 0  
 FROM [MyCube]  
 CELL PROPERTIES VALUE, FORMATTED\_VALUE, FORMAT, FONT\_NAME

If CELL PROPERTIES is not defined VALUE, FORMATTED\_VALUE, CELL\_ORDINAL properties will be retrieved. Note that CELL\_ORDINAL is always included even though it is not in the list of requested cell properties.  
  
  
Standard MDX properties:

* ACTION\_TYPE :
* BACK\_COLOR :
* CELL\_ORDINAL \* :
* FONT\_FLAGS :
* FONT\_NAME :
* FONT\_SIZE :
* FORE\_COLOR :
* [FORMAT\_STRING](http://www.iccube.com/support/documentation/mdx/cellProperties/FORMAT_STRING.html) : controls how cell values are formatted.
* FORMATTED\_VALUE \* :
* [LANGUAGE](http://www.iccube.com/support/documentation/mdx/cellProperties/LANGUAGE.html) : controls how cell formatted values are localized.
* UPDATEABLE :
* VALUE \* :

|  |  |
| --- | --- |
| Comments | (standard MDX) |

* Comments syntax used in MDX expressions.
* Description
* MDX allows for adding comments in a query. Valid comments:  
    
  a) // This is a comment ending with the line  
    
  b) /\* This is a comment that can span  
  different lines as in C++ \*/  
    
  c) -- This is another comment ending with the line
* Characters (possibly spanning several lines) in between /\* \*/ are considered as comment.
* **Query**
* SELECT
* [Measures]./\* I'm a comment \*/ DefaultMember ON 0,
* [Geography].[Geo].[Continent].Members ON 1
* FROM
* [Sales]
* **Result**

|  |  |  |
| --- | --- | --- |
|  | | **Amount** |
| **America** | | 768 |
| **Europe** | | 255 |
| Create (Function) | |  |

Creates a function.

Description

icCube allows for declaring a function at schema level. This function is then available as any other regular MDX function (e.g., topCount). The syntax for declaring a function is very much similar to the declaration of a calculated member.  
  
CREATE [ CONST ] FUNCTION name( arg, ... ) AS MDX-expression  
  
A function can be declared as const using the CONST keyword. See the "Declared Functions" link later in this page for more details.

|  |  |
| --- | --- |
| Declared Functions | (icCube MDX extension) |

Declaration of functions written in MDX.

Description

icCube allows for declaring a function within the scope of a SELECT query. The syntax is very much similar to the declaration of a calculated member.  
  
WITH FUNCTION fun(a,b) AS a + b  
SELECT ...  
  
A function can be declared as const using the CONST keyword. In that case, the evaluation of the function is considered as a constant and is evaluated once outside the current context except for the slicer and the sub-select. Several limitations apply to const functions as described in the following examples.  
  
WITH CONST FUNCTION fun() AS ..  
SELECT ...  
  
Similarly, functions can be declared (and removed) at schema level using the CREATE / DROP statements.  
  
DROP FUNCTION fun  
CREATE FUNCTION fun(a,b) AS a+b

In this example, we are defining the factorial of a number. You may notice the usage of a recursive notation.

**Query**

CREATE FUNCTION \_factorial(n) AS Iif( n = 0, 1, n \* \_factorial(n-1) )

WITH

MEMBER f0 AS \_factorial(0)

MEMBER f3 AS \_factorial(3)

SELECT { f0, f3 } ON 0 FROM [Sales]

**Result**

|  |  |
| --- | --- |
| **f0** | **f3** |
| 1 | 6 |

**Assertion : MDX Equals**

WITH

MEMBER f0 AS 1

MEMBER f3 AS 6

SELECT { f0, f3 } ON 0 FROM [Sales]

|  |  |
| --- | --- |
| Create (Member) | (standard MDX) |

Creates a calculated member/measure.

Description

The following statement allows for adding calculated member to schema dimensions. It is very similar to the WITH MEMBER statement:  
  
CREATE [ CALCULATED ] MEMBER unique-name AS MDX-expression  
  
Note that for defining a CALCULATED measure you'll have to specify the cube the calculated measure declaration applies to (then the [Measures] dimension is optional):  
  
CREATE CALCULATED MEMBER [Sales].[Measures].[Diff with United States] AS '[Measures].[Amount] - ([Geography].[United States],[Measures].[Amount])'  
  
Please, check working examples for further explanation.

Example: Market Share:

Note the usage of the cube name (i.e., [Sales]) in the CREATE statement.

**Query**

CREATE CALCULATED MEMBER [Sales].[Measures].[Market Share - schema] AS ([Measures].[Amount]) / ( [Product].currentMember.parent ,[Measures].[Amount] ), FORMAT\_STRING = "Percent"

WITH

MEMBER [Measures].[Market Share] AS [Measures].[Market Share - schema] -- using the schema definition

SELECT

[Measures].[Market Share] ON 0,

[Product].[icCube].children ON 1

FROM

[Sales]

WHERE

[Geography].[Geo].[Europe]

**Result**

|  |  |
| --- | --- |
|  | **Market Share** |
| **Corporate** | 56.47% |
| **Partnership** | 37.65% |
| **Personal** | 5.88% |
| **Startup** |  |

**Assertion : MDX Equals**

WITH

MEMBER [Measures].[Market Share - select] AS ([Measures].[Amount]) / ( [Product].currentMember.parent ,[Measures].[Amount] ), FORMAT\_STRING = "Percent"

MEMBER [Measures].[Market Share] AS [Measures].[Market Share - select] -- using the query definition

SELECT

[Measures].[Market Share] ON 0,

[Product].[icCube].children ON 1

FROM

[Sales]

WHERE

[Geography].[Geo].[Europe]

|  |  |
| --- | --- |
| Create (Set) | (standard MDX) |

Creates a set.

Description

The following statement allows for adding a set to a schema. It is very similar to the WITH SET statement:  
  
CREATE [ DYNAMIC | STATIC ] SET name AS MDX-expression  
  
DYNAMIC SETs are evaluated once per MDX query. The context of evaluation is the cube defined in the query.  
  
Example : CREATE DYNAMIC SET [Top 3 Countries / Amount] as TopCount( [Geography].[Geo].[City].Members, 3, [Measures].[Amount] )")  
  
STATIC SETs, the default modifier, are evaluated only once. The context of evaluation, cube, needs to be defined in the statement. Therefore, STATIC SETs can be used for example to cache costly evaluations.  
  
Example : CREATE STATIC SET [Sales].[Top 3 Countries / Amount] as TopCount( [Geography].[Geo].[City].Members, 3, [Measures].[Amount] )")

Top Count( Static Set)

In this example, we're caching at schema level the top 3 cities for the [Measures].[Amount]. Note the usage of the cube name (i.e., [Sales]) in the CREATE statement.

**Query**

CREATE SET [Sales].[Top 3 Countries / Amount] as TopCount( [Geography].[Geo].[City].Members, 3, [Measures].[Amount] )

SELECT

[Measures].[Amount] ON 0,

[Top 3 Countries / Amount] ON 1

FROM

[Sales]

**Result**

|  |  |
| --- | --- |
|  | **Amount** |
| **New York** | 768 |
| **Geneva** | 128 |
| **Zurich** | 64 |

**Assertion : MDX Equals**

SELECT

[Measures].[Amount] ON 0,

TopCount( [Geography].[Geo].[City].Members, 3, [Measures].[Amount] ) ON 1

FROM

[Sales]

Declared Functions

[return value as any OLAP object](javascript:%7b%7d)

In this example, we are returning a member to demonstrate that functions are not limited to scalar value.

**Query**

WITH

FUNCTION f() AS [Measures].defaultMember

SELECT

f() ON 0,

[Geography].[Geo].[America].children ON 1

FROM

[Sales]

**Result**

|  |  |
| --- | --- |
|  | **Amount** |
| **Canada** |  |
| **Mexico** |  |
| **United States** | 768 |
| **Venezuela** |  |

**Assertion : Cell Equals**

SELECT

[Measures].defaultMember ON 0,

[Geography].[Geo].[America].children ON 1

FROM

[Sales]

|  |  |
| --- | --- |
| Drillthrough | (standard MDX) |

Allows for retrieving underlying table rows used to calculated cube cell(s).

Description

The DRILLTHROUGH statement is used to retrieve the rows of the data source used to compute one or more cube cells. In addition to supporting multiple cells, note that drilling through calculated members is also supported.

DRILLTHROUGH [MAXROWS integer-value]  
 SELECT-STATEMENT  
 [RETURN return-expression,return-expression,... ]

The optional MAXROWS limits the number of returned rows. See the icCube.xml file for the defaulted value (configuration property: icCube.drillthroughDefaultMaxRows) when not specified.  
  
The optional RETURN statement allows for specifying which information/column is going to be retrieved. When not specified the following columns are returned: RowId(), Cell(), CalcMemberName(), CalcMemberTuple(), Key and Name for each dimension linked to the measure(s) being used during the cell(s) calculation.  
  
The following RETURN expressions are available:  
  
RowId() : the row id in the fact table.  
  
Cell() : the tuple of the cell the rows are attached to.  
  
CalcMemberName() : the name of the calculated member being computed.  
  
CalcMemberUniqueName() : the unique name of the calculated member being computed.  
  
CalcMemberTuple() : the tuple of the calculated member being computed.  
  
CalcMemberInstr() : some information identifying the current MDX expression being computed in the context of a calculated member calculation.  
  
CalcMember() : a shortcut for CalcMemberName() and CalcMemberTuple().  
  
Key( [mdx-dimension-name], n ) : the key of the member of the specified dimension (the optional 0-based n value allows for retrieving composite keys).  
  
Name( [mdx-dimension-name] ) : the name of the member of the specified dimension.  
  
UniqueName( [mdx-dimension-name] ) : the unique name of the member of the specified dimension.  
  
[mdx-dimension-name] : the key and the name of the member of the dimension.

|  |  |
| --- | --- |
| Drop (Member) | (standard MDX) |

Delete a calculated member (measure).

Description

Delete a calculated member (measure) previously created using a create statement. Note that the DROP statement applied to a non existing calculated member does nothing.  
  
DROP MEMBER unique\_name [, unique\_name]\*  
  
Note the usage of the cube name (in front of the unique name) required to drop a calculated measure:  
  
DROP MEMBER [Sales].[Geography].[Geo].[Europe I]

|  |  |
| --- | --- |
| Existing | (standard MDX) |

EXISTING Operator.

Description

The EXISTING operator allows to evaluate a set in the context of the current coordinates (e.g., slicer, sub-select, iteration, etc...).  
  
EXISTING performs an exist on the following set using the current context (e.g. slicer); it works as a filter.

[Slicer - set](javascript:%7b%7d)

The set is evaluated taking into account the slicer definition.  
  
In our example, [Cities] set is filtering all [Economy] countries using the slicer [Switzerland]

**Query**

WITH

SET [Cities] AS EXISTING [Geography].[Economy].[Country].members

SELECT

[Cities] ON 0

FROM

[Sales]

WHERE

[Geography].[Geo].[Switzerland]

**Result**

|  |
| --- |
| **Switzerland** |
| 248 |

**Assertion : MDX Equals**

WITH

SET [Cities] AS {[Geography].[Economy].[Switzerland]}

SELECT

[Cities] ON 0

FROM

[Sales]

|  |  |
| --- | --- |
| Implicit Conversions | (standard MDX) |

MDX automatically converts several types (aka. implicit conversions).

## Description

MDX converts implicitly (automatically) some types if required by the context.  
  
As an example, if a function is expecting a member as first parameter and receives a hierarchy MDX is going to implicitly convert the hierarchy into a member using the hierarchy's default member.  
  
For readability, it is not advised to use implicit conversions as the actual value used is hidden to somebody not aware of the conversion happening in the background.

# Examples

[Implicit conversion from member to set](javascript:%7b%7d)

If required a member is automatically converted to a set. The resulting set contains the member as single item.

**Query**

[Geography].[Geo].[Switzerland]

**Result**

{

[Geography].[Geo].[Country].[Switzerland]

}

**Assertion : Cell Equals**

{[Geography].[Geo].[Switzerland]}

|  |  |
| --- | --- |
| Measure Group | (standard MDX) |

Measure groups allow to create a cube made of multiple fact tables.

## Description

A measure group contains the measures associated to the underlying fact tables. Different measure groups may transparently combined within a single MDX select statement (see the following examples for more details).

# Examples

[Simple example](javascript:%7b%7d)

Measure groups allow to combine on a same request [Measure] belonging to different fact tables.  
  
In our example, we are taking the FX for the [EUR] to apply it to the amounts.

**Query**

WITH

MEMBER amountUSD AS [Measures].[Amount] \* 1.0, FORMAT\_STRING='fixed'

MEMBER amountEUR AS [Measures].[Amount] \* ([Currency].[European Union Currency], [Measures].[FX], [Time].[May 2010]), FORMAT\_STRING='fixed'

SELECT

{amountUSD} + {amountEUR} on 0,

{Descendants([Geography].[Geo].[Europe])} on 1

FROM [Sales]

**Result**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | | | **amountUSD** | | **amountEUR** |
| **Europe** | | | 255.00 | | 275.91 |
| **France** | | | 4.00 | | 4.33 |
| **Paris** | | | 4.00 | | 4.33 |
| **Spain** | | | 3.00 | | 3.25 |
| **Barcelona** | | | 2.00 | | 2.16 |
| **Madrid** | | | 1.00 | | 1.08 |
| **Valencia** | | |  | |  |
| **Switzerland** | | | 248.00 | | 268.34 |
| **Geneva** | | | 128.00 | | 138.50 |
| **Lausanne** | | | 56.00 | | 60.59 |
| **Zurich** | | | 64.00 | | 69.25 |
| Range | (standard MDX) | |

MDX range operator (:).

## Description

Returns a set made of the members between the both specified members (specified members included).  
  
Both parameters must specify members within the same level of a given dimension. If the first member is NULL then the first member of the level (as specified by the second member) is used. If the second member is NULL then the last member of the level (as specified by the first member) is used.

# Examples

[member:member](javascript:%7b%7d)

All the months between [Time].[Month].[Jan 2011] and [Time].[Month].[Apr 2011] are returned.

**Query**

[Time].[Month].[Jan 2011] : [Time].[Month].[Apr 2011]

**Result**

{

[Time].[Month].[Jan 2011],

[Time].[Month].[Feb 2011],

[Time].[Month].[Mar 2011],

[Time].[Month].[Apr 2011]

}

**Assertion : Cell Equals**

{

[Time].[Month].[Jan 2011],

[Time].[Month].[Feb 2011],

[Time].[Month].[Mar 2011],

[Time].[Month].[Apr 2011]

}

|  |  |
| --- | --- |
| SubQueries | (standard MDX) |

MDX subQueries (aka. subSelect).

Description

SubQueries (or subSelects) are one of the latest addition to the MDX language. It is a powerful feature that allows for querying an MDX query instead of a cube.  
  
As opposed to the WHERE clause, subQueries also change the hierarchies structure. Hierarchies defined in a subQuery are filtered (accepting all descendants and ascendants members defined in the subQuery).  
  
See example 'SubQueries in Axis - Branch' for more information.

SELECT  
 [Measures].members ON 0  
 FROM [NON VISUAL]  
 ( SELECT [Geography].[Switzerland] ON 0 FROM [SALES] )

The 'NON VISUAL' flags control whether aggregation is using all members (NON VISUAL) or only the filtered members.

|  |  |
| --- | --- |
| Update Cube | (standard MDX) |

Updating cell values (Write back, What-if scenarios)

Description

In MDX it is possible to change values of existing cells. It's main use is generating and checking what-if scenarios for data changes. It's as easy to update a single cell that it is to change a whole sub-cube.  
  
The syntax of an write back is :

UPDATE CUBE Cube SET Tuple = NEW\_VALUE [WRITE\_BACK\_METHOD] [BY [Weight]]  
  
 UPDATE CUBE [Sales] SET ([Geography].[Geo].[Europe], [Measures].[Amount]) = 20

This query redefines the value for a whole sub-cube defined by ([Geography].[Geo].[Europe], [Measures].[Amount]) to a new value, 20. Note that for missing hierarchies/dimensions the default member will be used and that calculated members are not allowed.  
  
How is the query performing the change of value ? For each existing cell of the sub-cube a new value is assigned so the aggregation is equal to the entered amount. The user has at his disposal different ways of row value assignment.  
  
Note, and this is a difference from MSAS, that only non empty fact rows are updated. Fact rows can be seen as rows of the underlying fact table and will match a cell if the table has one fact row per cell or the flag 'aggregate facts' has been activated in the cube definition. Pay attention that in highly sparse cube with a lot of dimensions updating a tuple may represent very quickly millions of cells.  
  
There are different methods to allocate values to the sub-cube rows :

USE\_EQUAL\_ALLOCATION : row\_tuple\_value = new\_value / num\_rows\_in\_tuple ( note the former value is not used )  
  
 USE\_EQUAL\_INCREMENT : row\_tuple\_value = row\_tuple\_old\_value + (new\_value - old\_value) / num\_rows\_in\_tuple  
  
 USE\_WEIGHTED\_ALLOCATION :  
 without weight : row\_tuple\_old\_value \* new\_value / old\_value ( note this is equivalent to a percentual change on all cells )  
 with weight : new\_value \* weight \* num\_rows\_in\_tuple  
  
 USE\_WEIGHTED\_INCREMENT :  
 without weight : row\_tuple\_old\_value \* new\_value / old\_value ( same as USE\_WEIGHTED\_ALLOCATION without weight )  
 with weight : row\_tuple\_old\_value + (new\_value - old\_value) \* weight

where:

row\_tuple\_value = the new value of the fact row  
 row\_tuple\_old\_value = the former value of the fact row  
 new\_value = the value defined in the UPDATE CUBE MDX statement  
 old\_value = the former value of the tuple defined in the UPDATE CUBE statement  
 num\_rows\_in\_tuple = number of fact rows with a no empty value in the tuple

The UPDATE CUBE statement is performed in the scope of a transaction. A commit of this transaction will update the fact values with the modified values for all users. Note that this commit may change the type of the column to a Float/Double if the former type was an integer number (This is another difference with MSAS).

|  |  |
| --- | --- |
| Where Clause | (standard MDX) |

MDX where clause (aka. slicer).

Description

The MDX where clause even though sharing the same name as the SQL statement is fundamentally different.  
  
A MDX WHERE clause reduces the scope of the query. The WHERE clause reduces the cube into a new 'sliced' cube, the slice being defined by the WHERE clause.

SELECT  
 [Measures].Members ON 0,  
 [Geography].[Geo].Members ON 1  
 FROM  
 [Sales]  
 WHERE  
 ([Time].[Calendar].[Year].[2010])

In the previous query, only that section of the cube where the year is 2010 is used; the rest of the cube is ignored.  
  
Tuples and sets can be used in a WHERE clause :

SELECT  
 ...  
 WHERE  
 {([Geography].[Geo].[Spain],[Product].[icCube].[Personal]),([Geography].[Geo].[Switzerland],[Product].[icCube].[Corporate])}

See 'Where clause on a set of tuples' example for further explanation.  
  
The where clause does not allow to use a hierarchy being used in one of the axis. SubQueries, which are similar, offer greater flexibility. The cost maybe slower performance.

|  |  |
| --- | --- |
| Where Clause Exists | (standard MDX) |

More about the slicer : i.e., behaviors related to tuple exists mechanism.

## Description

This section highlights several usage of the slicer related to tuple completion and exists mechanisms (i.e., tuple with several members of the same dimension).

## See Also

[Where Clause](http://www.iccube.com/support/documentation/mdx/Where%20Clause.html) [Where Set Clause Exists](http://www.iccube.com/support/documentation/mdx/Where%20Set%20Clause%20Exists.html)

# Examples

[Calculated Member : tuple completion](javascript:%7b%7d)

Calculated members are completed with the slicer content. Note, that hierarchies of the same dimension are also completed.  
  
In our example, the slicer member [Geo].[France] is completing the [Economy] hierarchy members defined in the calculated member [sliced].

**Query**

WITH

MEMBER [Measures].[sliced] AS Sum( [Geography].[Economy].[Country].members, [Measures].[Amount] )

SELECT

{ [Measures].[amount], [Measures].[sliced] } on 0

FROM

[Sales]

WHERE

( [Geography].[Geo].[France] )

**Result**

|  |  |
| --- | --- |
| **Amount** | **sliced** |
| 4 | 4 |

**Assertion : Cell Equals**

WITH

MEMBER [Measures].[france] AS ( [Geography].[Economy].[France], [Measures].[Amount] )

SELECT

{ [Measures].[france], [Measures].[france] } on 0

FROM

[Sales]

|  |  |
| --- | --- |
| Where Clause | (standard MDX) |

MDX where clause (aka. slicer).

Description

The MDX where clause even though sharing the same name as the SQL statement is fundamentally different.  
  
A MDX WHERE clause reduces the scope of the query. The WHERE clause reduces the cube into a new 'sliced' cube, the slice being defined by the WHERE clause.

SELECT  
 [Measures].Members ON 0,  
 [Geography].[Geo].Members ON 1  
 FROM  
 [Sales]  
 WHERE  
 ([Time].[Calendar].[Year].[2010])