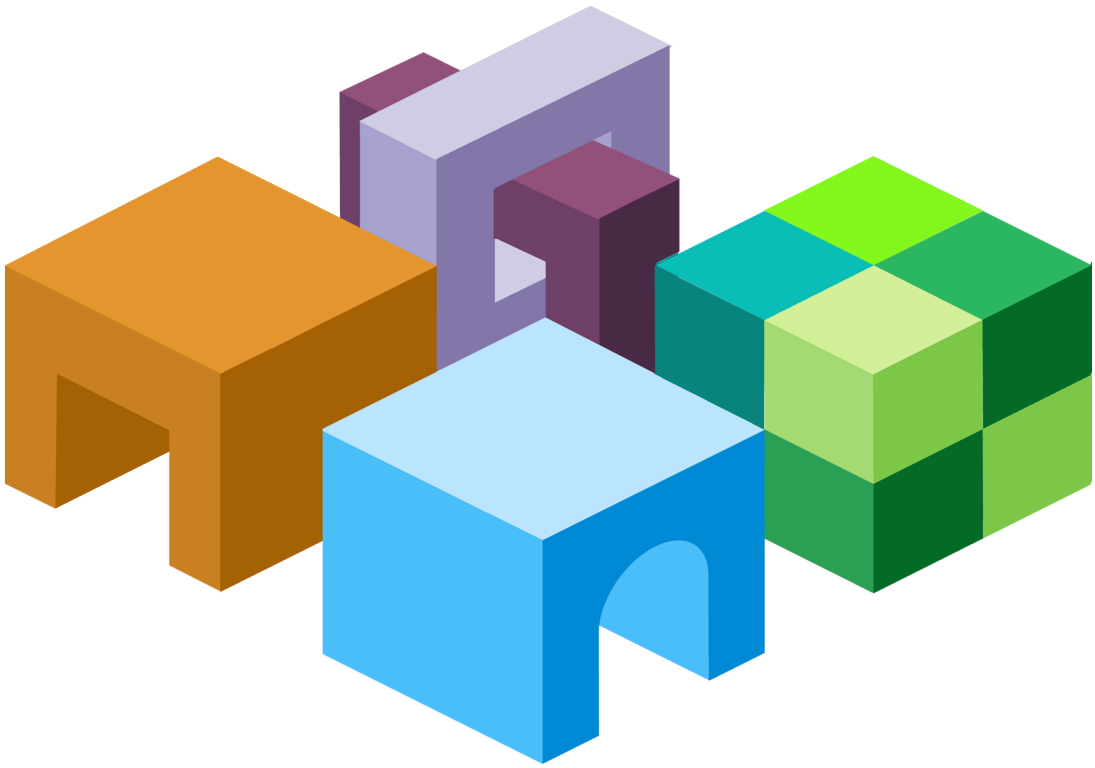

HYPERION® INTERACTIVE REPORTING

RELEASE 11.1.2

OBJECT MODEL AND DASHBOARD DEVELOPMENT SERVICES DEVELOPER'S GUIDE

VOLUME III: OBJECT MODEL GUIDE TO METHODS

ORACLE
ENTERPRISE PERFORMANCE MANAGEMENT SYSTEM

Interactive Reporting Object Model and Dashboard Development Services Developer's Guide, 11.1.2

Authors: EPM Information Development Team

Contents

1 Methods and the Object Model

Welcome to *Volume III: Object Model Guide to Methods*. This guide describes the methods (actions) associated with the Interactive Reporting Object Model (a hierarchical representation of Interactive Reporting).

Typically, methods are actions that return values and may require arguments. Methods can create, activate, open, close, save, add, copy, remove, process, export, recalculate, and so on. Properties are associated with what objects are; methods are associated with what objects do.

This guide provides examples and code that you can use to create scripts for Dashboard sections.

2 Methods

In This Chapter

Activate (Method)

Applies To:

ChartSection object, (CubeQuery)QuerySection object, DataModelSection object, Document object, DashboardSection object, OLAPQuerySection object, PivotSection object, QuerySection object, ReportSection object, ResultsSection object, Sections collection, TableSection object, WebClientDocument object

Description:

Changes the focus of an Oracle's Hyperion® Interactive Reporting document file or section.

Syntax:

```
Expression.Activate()
```

Expression Required:

An expression that returns an object for these sections:

- ChartSection
- DataModelSection
- DashboardSection
- OLAPQuerySection
- CubeQuerySection
- PivotSection
- QuerySection
- ReportSection
- ResultsSection
- Sections
- TableSection
- WebClientDocument

Example:

This example shows how to display and activate a section:

```
var MySection = ActiveDocument.Sections["Results"]
MySection.Visible = true
MySection.Activate()
```

Add (Method)

Applies To:

AggregateLimits collection, AppendQueries collection, Association collection, CategoryItems collection, ChartSection object, ColorRanges collection, Columns collection, ControlsDropDown object, ControlsListBox object, ColorRanges collection, Documents collection, Fact collection, Images collection, Joins collection, Limits collection, LimitValues collection, LocalJoins collection, LocalResults collection, OLAPFilter collection, OLAPLabels collection, OLAPMeasures collection, OLAPSlicers collection, Parentheses collection, PivotLabel object, PivotLabelTotals collection, QueryLabel object, Requests collection, Sections collection, SortItems collection, TargetFact collection, TopLabels collection, Topics collection

Description:

Common to most collections, the Add (Method) adds an object to a collection and returns a corresponding reference.

Note: Do not use Session.Form.Add(), Session.URL.Add() and Session.Cookies.Add() inInteractive Reporting files deployed in the Oracle Enterprise Performance Management Workspace, Fusion Edition. Section.Add is not available for the Dashboard section.

Note: The Add method for CubeQuery "download to results" dependent reporting sections (Chart, Pivot, Table, Report) does not accept the CubeQuery "Query" section name as the dependent section argument. This is different from use of the Add method with a relational Query section. With CubeQuery, you must use the downloaded "Results" section as the dependent section argument.

The Add() method works as follows:

- With AvailableValues (Collection): Nothing happens because the values are obtained from the database.
- With CustomValues (Collection): Values added to the list.
- With SelectedValues (Collection): Value added to a selected list.

For more information see, the *Object Model Guide to Objects and Collections*.

Syntax:

```
Expression.Add(ItemName As String)
```

Syntax (CubeQuery only):

```
Expression.Add(MemberLocation as String, SelectorType as BQOLAPSelector, IncludeSelf as
Boolean, SkipValidation as Boolean)
```

The Add (Method) adds a new filter to the end of the collection. If there is a filter defined for the given dimension, or the dimension or member do not exist, an exception is thrown. The MemberLocation must be a fully qualified path to a member, including its dimension and all intermediate members, delimited by periods ".". If the dimension name or any member name contains a period it must be escaped using a backslash. The optional Variable and Ignore arguments can be used to set the initial values of the OLAPFilter Variable and Ignore properties. They both default to false. If the given member location cannot be mapped to a valid dimension and member an exception is thrown.

If an alias table is in use, the member must be specified using the aliases, for example "Year.Quarter1.January", not "Year.Qtr1.Jan". The MemberLocation argument must include the dimension name as the first part of the value. Wildcards may not be used. The FindAndAdd (Property) can be used to add members using wildcards. The SelectorType argument specifies what type of MemberSelector will be created and equates to the right-click menu in the Member Selection dialog. The bqOlapSelector enumeration contains values for Member, Children, Descendants, Bottom, Siblings, Same Level, and Same Generation. The IncludeSelf argument means the member given in the Member argument is to included as well as any members generated by the SelectorType. For example, if "Qtr1" is added on the Year label with a SelectorType of bqOlapSelectorChildren, if IncludeSelf is true four members will be added (Qtr1, Jan, Feb, Mar) but only three members are added if IncludeSelf is false (Jan, Feb, Mar). The IncludeSelf argument only affects the result when SelectorType is bqOlapSelectorChildren, bqOlapSelectorDescendants, or bqOlapSelectorBottom. The optional SkipValidation argument can be used to change the function behavior. The default value is false. When the default value is false, the function behaves as described above. When the value is set to true, the Physical member name must be used instead of member location, and no exception will be thrown even

if the member does not exist. An ItemNotFound exception will be thrown if the member does not exist at the location specified. The Add (Method) does not support the Select Next/Previous, Subset, and Dynamic Time Series selectors.

Expression Required:

An expression that returns an object for these objects:

- CategoryItems
- ChartSection
- Columns
- ControlsDropDown
- ControlsListBox
- Documents
- Joins
- Limits
- LimitValues
- LocalJoins
- LocalResults
- OLAPLabels
- OLAPMeasures
- OLAPSlicers
- PivotLabels
- Requests
- Sections
- Topics
- TrendLines

Example 1:

This example shows how to create a limit, add limit value, and add the limit to the limit line:

```
var MyLimit =
ActiveDocument.Sections["Query"].Limits.CreateLimit("Stores.Store_Id")
MyLimit.SelectedValues.Add(2)
ActiveDocument.Sections["Query"].Limits.Add(MyLimit)
```

Example 2:

This example shows how to add values to a list box and drop-down list:

```
ActiveDocument.Sections["Dashboard2"].Shapes["DropDown1"].Add(20)
ActiveDocument.Sections["Dashboard2"].Shapes["ListBox1"].Add(1)
```

Example 3:

This example shows how to add new topics to a Data Model and join the topics:

```
var Topic1 =
ActiveDocument.Sections["Query"].DataModel.Catalog.CatalogItems["sales_fact"]
ActiveDocument.Sections["Query"].DataModel.Topics.Add(Topic1)
var Topic2 = ActiveDocument.Sections["Query"].DataModel.Catalog.CatalogItems["Store_ID"]
ActiveDocument.Sections["Query"].DataModel.Topics.Add(Topic2)
var TopicItem1 =
ActiveDocument.Sections["Query"].DataModel.Topics
["SalesFact"].TopicItems["Store_Id"]
var TopicItem2 =
ActiveDocument.Sections["Query"].DataModel.Topics
["Stores"].TopicItems["Store_Id"]
ActiveDocument.Sections["Query"].DataModel.Joins.Add(TopicItem1,TopicItem2,
bqJoinSimpleEqual)
```

Example 4:

This example illustrates how to add a Pivot section type to a Results section:

ActiveDocument.Sections.Add(bqPivot,"Results")

Note: Unlike a Query, Dashboard, or Report section type, you must associate Chart, Pivot, and Table section types with a parent section.

Example 5:

This example shows how to add different types of members:

```
// Add the Qtr1.Jan member, left off IncludeSelf argument because it
// is ignored for this selector type
Sections["Query"].Rows["Year"].Add("Qtr1.Jan", bqOlapSelectorMember);

// Add Jan, Feb, Mar members
Sections["Query"].Rows["Year"].Add("Qtr1", bqOlapSelectorChildren, false);

// Add Qtr1, Jan, Feb, Mar members
Sections["Query"].Rows["Year"].Add("Qtr1", bqOlapSelectorChildren, true);

Sections["Query"].Rows["Year"].Add("Year", bqOlapSelectorBottom, true);
```

AddAll (Method)

Applies To:

SelectedValues collection (instantiate from the LimitValues Collection)

Description:

Enables you to select all values from the AvailableValues (Collection) or CustomValues (Collection) depending on the selection.

Use this method with the LimitValueType (Property) to determine in advance which limit value set is selected. The value associated with this property is a member of the BqLimitValueType constant group. Two possible values of BqLimitValueType are: bqLimitValueTypeAvailable and bqLimitValueTypeCustom.

Note: Select single values using the "Add (Method)" on page 13 of the SelectedValues (Collection). To do so, you must already know all the values. This way of selecting a value can become tedious when there are a lot of values.

Syntax:

```
Expression.SelectedValues.AddAll();
```

Expression Required:

An expression that returns a limit object

Example:

In this example, a *Quarter* limit is created, added to the limit line in the Query section, and all available values in the Limit dialog box added:

```
//Adds a limit to the limit line of the Query section
mylimit =ActiveDocument.Sections["Query"].Limits.CreateLimit("Periods.Quarter")
mylimit.Operator=bqLimitOperatorEqual
ActiveDocument.Sections["Query"].Limits.Add(mylimit)
//Selects ALL Available values in the Limits dialog
ActiveDocument.Sections["Query"].Limits[1].SelectedValues.AddAll()
```

AddAllTopics (Method)

Applies To:

DefinedJoinPath collection

Description:

If you programmatically define join paths, use the AddAllTopics() method of the DefinedJoinPaths (Collection) to select all topics based on items on the Request and Limit lines. This is like using all topics in the Define Join Path dialog to populate the Topics in Join Path list.

Syntax:

```
ActiveDocument.Sections["Query"].DataModel. JoinsOptions.DefinedJoinPath["MyJoinPath"].
AddAllTopics()
```

Example:

In this example all topics are added to the *MyJoinPath* join path.

```
ActiveDocument.Sections["Query"].DataModel.JoinsOptions.DefinedJoinPath[“MyJoinPath”].Ad
dAllTopics()
```

AddComputed (Method)

Applies To:

Columns collection

Description:

Creates a computed column in a Table or Results section.

Syntax:

```
Expression.AddComputed(Name As String,
Expression As String) As Column
```

Expression Required:

An expression that returns an object for **Columns**

Example:

This example illustrates how to create a computed column that concatenates the “`Manager =`” string with the value in the *Store_Manager* column:

```
var ComputedExpression = " \"Manager =\" + Store_Manager"
ActiveDocument.Sections["Results"].Columns.AddComputed("MyComputed", ComputedExpression)
```

AddComputedItem (Method)

Applies To:

Chart Facts, Pivot Facts, Request

Description:

Creates a computed item and returns an object that represents the item.

This method enables you to specify the name, expression, and index for the computed item.

Calculated items created in the Chart section are facts and placed in the Y-Facts pane of the Chart Outliner.

The *name* is the computed item name that appears in the Y-Fact pane of the Chart or Pivot Outliner and the Chart legend.

The expression that you specify must be a valid Oracle's Hyperion® Interactive Reporting Studio and that displays in the Computed Items dialog box.

The index determines the position of a computed item in a pane. For example, an index of two is the second item in the Y-Fact pane.

If you apply the AddComputedItem method to a Query Request object, use the BqDataType constant to confirm or change the data type of an item. This preserves the precision of a mixed-data type computations, or changes how data items are handled (e.g. interpreting numbers as strings).

Pay attention to data types when computing items in the Query section. In this case, computations are performed on the database server, and computed items may be handed to Interactive Reporting with an unanticipated data type. To ensure that data is handled correctly on server computations, set the data type when performing mixed-data type computations.

Syntax:

```
Chart: Expression.AddComputedItem(Name As String, Expression As String,[optional Index As Number])

PivotLabels: Expression.AddComputedItem(Name As String, Expression As String, [optional Index As Number]) As PivotLabel

Requests: Expression.AddComputedItem(Name As String, Expression As String, Type As BqDataType) As Request
```

Expression Required:

An expression that returns a **Chart fact**, **Pivot fact** or **Query request** object

Example:

This example shows how to create a computed column titled *Double Sales*, which doubles the amount in the Unit Sales column:

```
ActiveDocument.Sections["Chart"].Facts.AddComputedItem
('Double_Sales', 'Unit_Sales *2',2)
```

AddDrillThroughValue (Method)

Applies To:

OLAPQuery object

Description:

Adds a label name as a drill-through value.

Syntax:

```
Expression.AddDrillThroughValue(string LabelName)
```

Expression Required:

An expression that returns an add drill-through label value

Example:

This example shows how to add the label value *Store Name* as a drill-through value:

```
ActiveDocument.Sections["OLAPQuery"].AddDrillThroughValue("Store Name")
```

AddExportSection (Method)

Applies To:

ChartSection, Document object, PivotSection, QuerySection, Section, TableSection

Description:

Exports documents to HTML, that enables you to distribute data to many users using corporate intranets or Web sites. This scripting method:

- Executes a high-fidelity series of XHTML pages that match the original Interactive Reporting Studio reports closely.
- Creates a set of .htm, .css, gif,, and .jpg (if charts or Dashboard sections are in the export set) files.

The resulting file set is a frame-based HTML display that includes a report navigation frame, a report display area, and navigational report hyperlinks

When exporting selected sections, specify the section name in the AddExportSection (Method). You must specify a call to AddExportSection Method) to export a section. After specifying all sections to export, the Document level "Export (Method)" on page 52 is called, you use to specify the export file format.

Exported documents preserve the original fixed section order of an Interactive Reporting document file, regardless of the order of the AddExportSection (Method) calls. Invalid AddExportSection (Method) calls, caused by an invalid section type or name, are ignored.

When sections are export, the Export (Method) clears the export buffer. If sections do not export, use the "RemoveExportSection (Method)" on page 113 to flush the export buffer and remove sections set for export. For instance, if you specify a Report, Pivot, and Chart section to export using the AddExportSection (Method), a call to the RemoveExportSections (Method) nullifies the section for export. Calls to the Export (Method) assume that you want to export all, instead of individual, sections.

Exported documents reside in the default export directory where the brioqry.exe file resides. You can modify the export directory by specifying a path for the filename argument in the Export() method, such as *c:\\temp\\myfile.htm* . HTM extensions denote HTML files, as shown in this example:

```
Documents["MyDocument.bqy"].Export('C:\\Temp\\MyExportFile.htm',BqExportFileHTML)
```

Note: You cannot export the Query, OLAPQuery, and DataModel sections.

Syntax:

```
Expression.AddExportSection(SectionName As String)
```

Expression Required:

An expression that returns:

- ChartSections
- PivotSections
- TableSections
- Sections

Example 1:

This example shows how to export selected sections of Interactive Reporting document file:

```
//Export SELECTED Sections of .bqy document
ActiveDocument.AddExportSection('Report')
ActiveDocument.AddExportSection('Report2')
ActiveDocument.AddExportSection('Results')
ActiveDocument.AddExportSection('Table')
ActiveDocument.AddExportSection('Pivot')
ActiveDocument.AddExportSection('Pivot2')
ActiveDocument.AddExportSection('Pivot3')
ActiveDocument.AddExportSection('Chart')
ActiveDocument.AddExportSection('Chart2')
ActiveDocument.AddExportSection('OLAPQuery')
ActiveDocument.Export('C;\\Temp\\MyExportFile.htm', bqExportFormatHTML)
```

Example 2:

In this example, selected sections are set to be exported, then cleared from the export buffer. The Export method in the last section of script enables all sections to be exported:

```
//Export SELECTED Sections of .bqy document
Documents["MyDocument.bqy"].AddExportSection('Report')
Documents["MyDocument.bqy"].AddExportSection('Report2')
Documents["MyDocument.bqy"].AddExportSection('Results')
Documents["MyDocument.bqy"].AddExportSection('Table')
Documents["MyDocument.bqy"].AddExportSection('Pivot')
Documents["MyDocument.bqy"].AddExportSection('Pivot2')
Documents["MyDocument.bqy"].AddExportSection('Pivot3')
Documents["MyDocument.bqy"].AddExportSection('Chart')
Documents["MyDocument.bqy"].AddExportSection('Chart2')
Documents["MyDocument.bqy"].AddExportSection('OLAPQuery')
Documents["MyDocument.bqy"].Export('C;\\Temp\\MyExportFile.htm', bqExportFormatHTML)
ActiveDocument.RemoveExportSections();
//Export ALL sections of .bqy document since Export buffer was flushed
ActiveDocument.Export('C;\\Temp\\MyExportFile.htm', bqExportFormatHTML)
```

AddFilter (Method)

Applies To:

OLAPLabel object

Description:

Adds a new filter value and returns an object that represents the new item.

The AddFilter (Method) uses all operator types for side labels and measures for all supported OLAP engines.

The "AddFilterValue (Method) " on page 23 supports only the Select MembersFromDatabase operator type.

Syntax:

```
Expression.AddFilter(BqOperatorType operatorType, BqOperator dataOperator, String
value1, String value2, [optional] Boolean IsVariable]
```

Expression Required:

An expression that returns an object for **an OLAPLabel object**

Constants:

The AddFilter (Method) uses the BqOperatorType and BqOperator constant groups.

The BqOperatorType constant group consists of these values:

- bqOperatorTypeBottomN
- bqOperatorTypeBottomNPercent
- bqOperatorTypeBottomSum
- bqOperatorTypeMeasure
- bqOperatorTypeSelectByMeasure
- bqOperatorTypeSelectMembers
- bqOperatorTypeSelectMembersFromDB
- bqOperatorTypeSelectMembersFromFile
- bqOperatorTypeSubstitutionVariables
- bqOperatorTypeTopN
- bqOperatorTypeTopNPercent
- bqOperatorTypeTopSum
- bqOperatorTypeUDA
- bqOperatorTypeUndefined

The BqOperator constant group consists of these values:

- bqOperatorEqual
- bqOperatorGreaterThan
- bqOperatorGreaterThanOrEqual
- bqOperatorLessThan
- bqOperatorLessThanOrEqual
- bqOperatorMatchMember
- bqOperatorNotEqual
- bqOperatorUndefined

Example 1:

This example shows how to add a filter to the Product Family top label equal to the "Drink" database member. The filter value was not made a variable:

```
ActiveDocument.Sections["OLAPQuery"].TopLabels["Product
Family"].AddFilter(bqOperatorTypeSelectMembersFromDB,  bqOperatorEqual, "Drink", false)
```

Example 2:

This example shows how to add a Top N filter to the Product Family side label. The filter value retrieves only one Product Family member with the top value of the Store Invoice measure.

```
ActiveDocument.Sections["OLAPQuery"].SideLabels["Product
Family"].AddFilter(bqOperatorTypeTopN, bqOperatorUndefined, "1","Store Invoice", false)
```

AddFilterValue (Method)

Applies To:

OLAPLabel, OLAPMeasures

Description:

Adds a new filter value and returns an object that represents the new item.

The AddFilterValue (Method) supports only the Select MembersFromDatabase operator type. The "AddFilter (Method)" on page 22 uses all operator types for side labels and measures for all supported OLAP engines.

Note: You can only use this method to apply a filter to a measure value against an Oracle Essbase database. You cannot use an alias.

Syntax:

```
OLAPLabel.AddFilterValue(MemberName As String, Operator As BqOperator)
```

```
OLAPMeasure.AddFilterValue(ColumnIndex As String, Operator As BqOperator, MeasureValue
As String)
```

Expression Required:

An expression that returns an **OLAPLabel** or **OLAPMeasure** object.

Constants:

The AddFilterValue (Method) uses the BqOperator constant group, which consists of these values:

- bqOperatorEqual
- bqOperatorGreaterThan
- bqOperatorGreaterThanOrEqual
- bqOperatorLessThan
- bqOperatorLessThanOrEqual
- bqOperatorMatchMember
- bqOperatorNotEqual
- bqOperatorUndefined

Example 1:

This example shows how to add the new filter *AZ* item to the side label:

```
OQPath = ActiveDocument.Sections["OLAPQuery"]
OQPath.SideLabels[1].AddFilterValue('AZ',bqOperatorEqual)
OQPath.Process()
OQPath.Activate()
```

Example 2:

This example shows how to add a filter value to a *Profit* measure. The operator equals 13,438.

```
ActiveDocument.Sections["OLAPQuery"].Measures["Profit"].AddFilterValue
('1',bqOperatorEqual,'13438')
```

Example 3:

This example shows how to add a filter value to a *Name* measure:

ActiveDocument.Sections["OLAPQuery2"].TopLabels["Name"].AddFilterValue("[Eastern Winds]", bqOperatorEqual)

Example 4:

This example shows how to add a filter value to the *Name* measure. This example places the filter value in brackets to use a URL.

ActiveDocument.Sections["OLAPQuery2"].TopLabels["Name"].AddFilterValue("[www.easternwinds.com]", bqOperatorEqual)

AddTopic (Method)

Applies To:

DefinedJoinPath collection

Description:

If you define your own join paths, use the AddTopics() method of the DefinedJoinPaths (Collection) to select a topic based on an item on the Request or Limit lines. This method corresponds to selecting an available topic on the Define Join Path dialog box and adding it to the Topics in Join Path list.

Syntax:

```
ActiveDocument.Sections["Query"]. DataModel.JoinsOptions.DefinedJoinPath["MyJoinPath"].
AddTopic(String)
```

Example:

In this example, the topic *Products* is added to the *MyJoinPath* join path:

```
ActiveDocument.Sections["Query"].DataModel.JoinsOptions.DefinedJoinPath["MyJoinPath"].Ad
dTopic("Products")
```

AddTotal (Method)

Applies To:

OLAPLabel object (TopLabels and SideLabels object)

Description:

Creates an additional column containing the totals for a top or side label.

Syntax:

```
Expression.AddTotals()
```

Expression Required:

An expression that returns a PivotLabel object

Example:

This example shows how to total the side label columns called *Year*:

```
ActiveDocument.Sections["OLAPQuery"].TopLabels["Year"].AddTotal()
```

AddTotals (Method)

Applies To:

PivotLabels (TopLabels and SideLabels collections)

Description:

Creates an additional row or column containing the totals for all columns or rows of the pivot.

Syntax:

```
Expression.AddTotals()
```

Expression Required:

An expression that returns a PivotLabel object

Example 1:

This example shows how to total the top label columns called *Product*:

```
ActiveDocument.Sections["Pivot"].TopLabels["Product Id"].AddTotals()
```

Example 2:

This example shows how to add a total to the side label rows called *Quarter*:

```
ActiveDocument.Sections["Pivot"].SideLabels["Quarter"].AddTotals()
```

Alert (Method)

Applies To:

Application object

Description:

Displays a simple dialog box on which you can display three buttons with custom names. When users select a button, an integer is returned that corresponds to the button number. For example, if users click button #1, number 1 is returned.

Note: If an Alert is used after the Printout (Method) in an Interactive Reporting document file deployed in the EPM Workspace, the Printout (Method) does not execute.

Syntax:

```
Expression.Alert(Prompt As String, [Title As String], [Button1Text As String],
[Button2Text As String], [Button3Text As String]) As Integer.
```

Expression Required:

An expression that returns an **Application object**

Example:

This example shows how to display an Alert dialog and process a user response:

```
var ReturnVal =0
ReturnVal = Alert("Please press a button","Alert Title","One","Two","Three")
switch (ReturnVal)
{
     case 1:
     Alert("The user pressed the One button")
     break;
case 2:
     Alert("The user pressed the Two button")
     break;
case 3:
     Alert("The user pressed the Three button")
     break;
default:
     Alert("An error occurred!")
}
```

AliasTable (Method)

Applies To:

DBSpecific object (Essbase and DB2 OLAP only)

Description:

Enables you to use aliases to assign custom names to database physical members or generation/level names. Essbase stores aliases in an Alias Table in the cube. Since cubes can have multiple alias tables, you can select the alias table and modify the query based on the value you enter.

The AliasTable (Method) prompts users to process queries. The PromptOption argument determines if a dialog displays from which users specify a value of 1 (OK) or 2 (Cancel).

The PromptDialog determines if the dialog box displays. The two arguments can work together or separately. If both arguments are specified, the dialog box displays with the prompt option as the default selection (user can then change the option when prompted with the dialog box). If only the PromptOption argument is specified, no dialog displays and the prompt option is executed. If only the PromptDialog argument is specified, the dialog displays with the default prompt of *OK*. If neither option is specified, then no dialog appears and the default option of *OK* is assumed.

The AliasTable (Method) prompts a user to use the process command.

Syntax:

```
Expression.AliasTable(String AliasTableName,Number PromptOption, Boolean PromptDialog)
```

Expression Required:

An expression that returns a database **specific** object

Example:

This example calls the Beep function of the Kernal32.dll for 4 seconds with 5000Hz:

```
ActiveDocument.Sections["OLAPQuery"].DBSpecific.AliasTable("default", 2, false)
```

ApplyColors (Method)

Applies To:

ColorSet object

Description

Applies the color changes made to a color element associated with a custom chart scheme. This method is used in conjunction with the ColorSet (Object). The ApplyColors (Method) does not allow you to add or remove color elements.

Example:

This example shows you how to set the chart color scheme to a custom color scheme, changes the first color in the scheme from number 1 to number 10 (from blue to black), and applies the color changes.

```
ActiveDocument.Sections["Chart3"].ColorScheme = bqChartColorSchemeCustom
ActiveDocument.Sections["Chart3"].Colors.Color1 = 10
ActiveDocument.Sections["Chart3"].Colors.ApplyColors()
```

AuditSQL (Method)

Applies To:

Query object, DataModel object

Description:

Enables you to define SQL Statements that execute when audit events are triggered. This record show Interactive Reporting Studio, a database server, or network resources are used. When triggered, SQL statements update an audit log table, which administrators can query to monitor data.

Syntax:

```
Expression.AuditSQL(EventType As BqAuditEventType, SQLStatement As String)
```

Expression Required:

An expression that returns a Query Object

Constants:

The BqAuditEventType constant group consists of these values:

- bqAuditDataModelRefresh
- bqAuditDetail View
- bqAuditLimitShowValues
- bqAuditLogoff
- bqAuditLogon
- bqAuditNewDataModel
- bqAuditPostProcess
- bqAuditPreProcess

Example 1:

In this example, an audit event is triggered when the user logs on:

```
ActiveDocument.Sections["Query"].DataModel.AuditSQl(bqAuditLogon,"Select username from
all_users")
```

Example 2:

In this example, an audit event is triggered when the user logs off:

```
ActiveDocument.Sections["Query"].DataModel.AuditSQl(bqAuditLogoff,"Select username from
all_users")
```

Example 3:

In this example, an audit event is triggered when *Process* is selected, but before the SQL query statement executes:

```
ActiveDocument.Sections["Query"].DataModel.AuditSQl(bqAuditPreProcess,"Select username
from all_users")
```

Example 4:

In this example, an audit event is triggered when the final row in the Results set retrieves to the client workstation:

```
ActiveDocument.Sections["Query"].DataModel.AuditSQl(bqAuditPostProcess,"Select username
from all_users")
```

AutoArrangeLabel (Method)

Applies To:

PieChart object

Description:

Repositions a label automatically with the slice when the slice is rearranged.

Syntax:

```
Expression.AutoArrangeLabel(Value ItemNameOrIndex)
```

Expression Required:

An expression that returns a PieChart object

Example:

This example shows how to enable the automatically arrange labels:

```
ActiveDocument.Sections["UnitProdLineChart"].PieChart.AutoArrangeLabel(1)
```

AutoSizeHeight (Method)

Applies To:

PivotLabel collection, PivotFact object, OLAPLabel object, OLAPMeasure object, QueryLabel object

Description:

By default, Interactive Reporting Studio truncates Pivot fact columns evenly, regardless of the length or height of data values. Numeric data that does not fit is replaced with pound signs (#). Use the AutoSizeHeight method to automatically size the height of a Pivot fact columns to ensure that all values display.

Syntax:

```
Expression.AutoSizeHeight()
```

Expression Required:

An expression that autosizes the height of a Pivot fact column

Example:

This example shows how auto size the height and the width of the *Unit Sales* fact column:

```
ActiveDocument.Sections["Pivot"].Facts["Unit Sales"].AutoSizeHeight()
ActiveDocument.Sections["Pivot"].Facts["Unit Sales"].AutoSizeWidth()
```

AutoSizeWidth (Method)

Applies To:

PivotLabel collection, PivotFact object, OLAPLabel object, OLAPMeasure object, QueryLabel object

Description:

By default Interactive Reporting truncates Pivot fact columns evenly, regardless of the length or height of data values. Numeric data that does not fit, is replaced with pound signs (#). Use the AutoSizeWidth method to automatically size the width of a Pivot fact column to ensure that all values are displayed.

Syntax:

```
Expression.AutoSizeWidth()
```

Expression Required:

An expression that autosizes the width of a Pivot fact column

Example:

This example shows how to auto size the height and width of the *Unit Sales* fact column:

```
ActiveDocument.Sections["Pivot"].Facts["Unit Sales"].AutoSizeWidth()
ActiveDocument.Sections["Pivot"].Facts["Unit Sales"].AutoSizeHeight()
```

Call (Method)

Applies To:

SharedLibrary object

Description:

Invokes functions in external DLLs.

Syntax:

```
Expression.Call(sFunctionName As String, sArgumentType As String, [arg1], [arg2],
[arg3], [arg4], [arg5], [arg6], [arg7], [arg8])
```

Expression Required:

An expression that returns a **SharedLibrary** object

Example:

This example calls the Beep function of the Kernal32.dll for four seconds with 5000Hz:

```
var oLibrary;
```

```
oLibrary = LoadSharedLibrary("kernel32.dll");
oLibrary.Call("Beep", "UI,UI", 5000, 4000);
```

ChartThisPivot (Method)

Applies To:

PivotSection object

Description:

Creates a Chart section using criteria defined in a Pivot section.

Syntax:

```
Expression.ChartThisPivot()
```

Expression Required:

An expression that returns an object for the **Chart section**

Example:

This example shows how to chart a Pivot section and change the chart display settings:

```
MyChart = ActiveDocument.Sections["Pivot"].ChartThisPivot()
MyChart.Title = "Chart Created from Pivot"
```

Close (Method)

Applies To:

Document object,

Description:

Closes the Interactive Reporting document file. This method is equivalent to selecting File > Close.

Note: The Document.Close() object model syntax is not supported in an Interactive Reporting document file deployed in the EPM Workspace or in Oracle's Hyperion® Interactive Reporting Web Client.

Syntax:

```
Expression.Close([SaveChanges As Boolean])
```

Expression Required:

An expression that returns a **Document or WebClientDocument** object

Example:

This example shows how to close all the open documents in the application:

```
var OpenDocs = Documents.Count
for (j = 1 ; j <= OpenDocs ; j++)
    Documents[j].Close()
```

Connect (Method)

Applies To:

Connection object, MetaDataConnection object

Description:

Establishes a connection to the database using the criteria set in the connection object. (The Connection object represents either an Interactive Reporting database connection file (OCE) or the connection to a database that is associated ultimately with a specific query section, or less commonly, a specific data model section.)

The Connect method optionally relies on the GetCredentials argument that precedes Release 8.2 credential information.

GetCredentials Argument

The GetCredentials argument provides backward compatibility for users who need to make connections using a script created before Release 8.2. The GetCredentials argument takes a Boolean value.

When the GetCredentials argument is used in the Designer/Explorer application, you are prompted to supply the user ID and password, and optionally a database name. The prompt is the same one used when the user interface requests credentials, such as when you click the Process button and the connection status is *disconnected*.

If the GetCredentials argument is false when the Connect () method is invoked, whatever credentials have been supplied with the Username property together with a call to the Connection's SetPassword() method are used to establish the connection. If no credentials have been explicitly supplied by these means, you get a standard database error. When the GetCredentials parameter is used in the Web-based client applications, the credentials for the user are obtained following the rules established by the Foundation and by the publishers of the Interactive Reporting database connection file (OCE) and Interactive Reporting document file (BQY) content. For example, if the Interactive Reporting database connection file associated with the query is set to prompt the user when the Interactive Reporting document file was published, you are prompted; if at publishing time the credentials were supplied *(Specify Now)*, then those credentials are used, and so on.

The GetCredentials parameter by default is false. This parameter is persisted as part of a script; it is not saved with the Interactive Reporting document file, and when anInteractive Reporting document file is opened, it is set to its default value. The user must explicitly change the setting of GetCredentials if need be before calling the "Connect (Method)" on page 33.

Syntax:

```
Expression.Connect([optional] Boolean GetCredentials)
```

Expression Required:

An expression that returns a **Connection** object

Example 1:

This example shows how to establish a connection with a database using the connection object:

```
MyConnection = ActiveDocument.Sections["Query"].DataModel.Connection
MyConnection.Open("c:\\OCEs\\SampleDB.oce")
MyConnection.Username = "hyperion"
MyConnection.SetPassword("hyperion")
MyConnection.Connect(true)
```

Example 2:

This example shows how to use the "Disconnect (Method) " on page 40 to disconnect the current connection, and connect to another database. The Disconnect (Method) is available for Designer only:

```
// < disconnects the current connection
if (ActiveDocument.Sections["Query"].DataModel.Connection.Connected == true)
{
ActiveDocument.Sections["Query"].DataModel.Connection.Disconnect()
}
else
{
// < connect to another database
MyConnection = ActiveDocument.Sections["Query"].DataModel.Connection
MyConnection.Open("c:\\OCEs\\myNewSalesOCE.oce")
MyConnection.Connect(true)
}
```

Copy (Method)

Applies To:

ChartSection object, (CubeQuery) QuerySection object, PivotSection object, Section object, TableSection object, OLAPQuerySection object

Description:

Copies a section and places it on the clipboard.

Syntax:

```
Expression.Copy()
```

Expression Required:

An expression that returns an object for any of these sections:

- ChartSection
- PivotSection
- OLAPQuerySection
- CubeQuerySection
- Section
- TableSection
- ResultsSection object
- Imported Data File object

The Copy (Method) is unavailable for these objects:

- DashboardSection
- QuerySection
- ReportSection
- DataModelSection

The Copy (Method) does not work in an Interactive Reporting document file to be deployed in the EPM Workspace.

Example:

This example shows how to copy an entire Results section to the Clipboard:

```
ActiveDocument.Sections["Results"].Copy()
```

CreateConnection (Method)

Applies To:

Application

Description:

Use this method to create an Interactive Reporting database connection file for a relational database, which is not automatically associated with a Data Model. CreateConnection() returns an Interactive Reporting database connection file. Refer to the Connection (Object) for a complete list of its methods and properties.

This method does not allow you to create an Interactive Reporting database connection file for a multi-dimensional database (OLAPQuery or CubeQuery).

Note: The Application.CreateConnection() object model syntax is not supported in an Interactive Reporting document file deployed in the EPM Workspace.

Syntax:

```
Expression.CreateConnection() As Connection
```

Expression Required:

An expression that returns an Application object

Example:

This example shows how to create a connection from scratch, save it as an. Interactive Reporting database connection file and use it as the current connection. In this example, the host name uses the ODBC data source name *Bookmart*:

```
var myCon = CreateConnection()
myCon.Api = bqApiODBC
myCon.Database = bqDatabaseODBC
myCon.HostName = "Bookmart"
myCon.SaveAs("c:\\temp\\bookmart.oce")
var MyQuery = ActiveDocument.Sections.Add(bqQuery)
MyQuery.DataModel.Connection.Open("c:\\temp\\bookmart.oce")
MyQuery.DataModel.Connection.Connect()
```

CreateDateGroup (Method)

Applies To:

Column

Description:

Creates a date group from a Results or Table column. The data in the column must be a date.

Syntax:

```
Expression.CreateDateGroup()
```

Expression Required:

An expression that returns a **Column** object

Example:

This example searches through a Results set for a date column and creates a date group:

```
ColCount = ActiveDocument.Sections["Results"].Columns.Count
for (i = 1; i <= ColCount ; i++)
{
if (ActiveDocument.Sections["Results"].Columns[i].DataType ==bqDataTypeDate)
    ActiveDocument.Sections["Results"].Columns[i].CreateDateGroup()
}
```

CreateLimit (Method)

Applies To:

AggregateLimits collection, Limits collection

Description:

Creates a stand-alone limit object. After creating the limit, complete its properties before adding it to the limits collection.

Syntax:

```
Expression.CreateLimit(limitItem As String) As Limit
```

Note: The argument for CreateLimit method varies for regular, computed item, and aggregate limits. For regular limits the argument is a reference to the table topic and the topic item (e.g. CreateLimit("Sales_Facts.Amount_Sales")). For computed item and aggregate limits, the argument is a reference to the item Display Name on the request line (e.g. CreateLimit("Request.Amount Sales")).

Expression Required:

An expression that returns a **Limits** object

Example 1:

This example shows how to create a Results limit. When you create a local (Results) limit, the value for the LimitItem parameter needs to be the name of the column the limit is being applied:

```
MyLimit = ActiveDocument.Sections["Results"].Limits.CreateLimit("State")
MyLimit.Operator = bqLimitOperatorEqual
MyLimit.CustomValues.Add("CA")
MyLimit.SelectedValues.Add("CA")
ActiveDocument.Sections["Results"].Limits.Add(MyLimit)
ActiveDocument.Sections["Results"].Limits[1].DisplayName = "State" Expression Required:
```

Example 2:

This example sets the FROM clause and the WHERE clause, processes the query, and then restores the original SQL statement:

```
//Set the FROM clause, Set the WHERE clause, PROCESS, and then RESET
SQLActiveDocument.Sections["Query"].CustomSQLFrom("FROM From.Sales_Fact, From.Periods,
From.Products") ActiveDocument.Sections["Query"].CustomSQLWhere("WHERE
(Periods.Day_Id=Sales_Fact.Day_Id AND
Products.Product_Id=Sales_Fact.Product_Id)  AND (Periods.Quarter='Q1')")
ActiveDocument.Sections["Query"].Process()
ActiveDocument.Sections["Query"].ResetCustomSQL();
```

CreateShape (Method)

Applies To:

Shapes collection

Description:

CreateShape(bqShapeType, [sectionName])

To programmatically create a shape on a dashboard, use the CreateShape function. The first parameter specifies the sort of shape or control to be created and must be one of the values of the bqShapeType enumeration.

If the shape being created is an embedded section the section parameter must be the name of the section to be embedded. This must be a chart, pivot, result set, or table. If the name of any other type of section is supplied the function fails and an exception is thrown.

When creating shapes using COM, you must supply a value for the second parameter. You can supply the empty string "" for this value for shapes and controls other than embedded sections.

Shapes are created at the top-left of the dashboard section, with a default size based on type of shape or control. The default size is the same size used when the same type of shape is dragged onto the dashboard in Design mode.

A reference to the newly created shape is returned from this function. You can use the reference to set the Placement of the shape using the Placement properties or functions, and you can set the name of the shape using the Name property.

Note: Using more than 1,021 shapes is not supported in a Dashboard section.

Example:

This example creates a rectangular shape as a goal, called LeftGoal, sets the placement properties, and colors the rectangle black.

```
var objRect = ActiveSection.Shapes.CreateShape(bqRectangle)
objRect.Name = "LeftGoal"
objRect.Placement.XOffset = Field.Placement.XOffset - 10
objRect.Placement.YOffset = Field.Placement.YOffset - 10
objRect.Placement.Width = Ball.Placement.Width - 60
objRect.Placement.Height = Field.Placement.Height + 20
objRect.Fill.Color = bqBlack
```

CustomSQLFrom (Method)

Applies To:

QuerySection object

Description:

Sets the FROM clause of a SQL statement before processing.

The FROM clause indicates which tables are to be referenced when the SELECT statement is processed.

The FROM clause is appended to your custom SQL each time the CustomSQLFrom method is activated. To clear any clauses appended to the Custom SQL statement, use the "ResetCustomerSQL (Method)" on page 117.

CustomSQLFrom (Method), "CustomSQLWhere (Method)" on page 40, and "ResetCustomerSQL (Method)" on page 117 correspond to the edit SQL functionality in the Custom SQL dialog box. However, no Custom SQL dialog is displayed when this method is executed.

Note: To use the Custom SQL feature, the query data model must have at least one table. If no table exists, then the Console Window displays this message: "Script(x):uncaught exception:Invalid String".

Note: You can use the CustomSQLFrom (Method) to define all of the SQL (including the WHERE clause) in the Custom SQL window. If a query includes temporary tables and correlated subqueries, it is recommended that you use CustomSQLFrom method to define all of the SQL.

Syntax:

```
Expression.CustomSQLFrom(String CustomSQLStr)
```

Expression Required:

An expression that returns a query object

Example:

This example sets the FROM clause and the WHERE clause, next processes the query and then restores the original SQL statement:

```
//Set the FROM clause, Set the WHERE clause, and PROCESS, and then RESET SQL
ActiveDocument.Sections["Query"].CustomSQLFrom('FROM From.Sales_Fact, From.Periods,
From.Products')
ActiveDocument.Sections["Query"].CustomSQLWhere('WHERE
(Periods.Day_Id=Sales_Fact.Day_Id AND Products.Product_Id=Sales_Fact.Product_Id)  AND
(Periods.Quarter='Q1')')
ActiveDocument.Sections["Query"].Process()
ActiveDocument.Sections["Query"].ResetCustomSQL();
```

CustomSQLWhere (Method)

Applies To:

QuerySection object

Description

Sets the WHERE clause of an SQL statement before processing, overwriting any SQL from the initial WHERE clause to the end of the SQL statement.

This is a useful method when you want to create a query which references temporary tables and you need to write the SQL WHERE clause to derive effective database indices.

The WHERE clause identifies which rows to use in a table based on selected criteria. The CustomSQLWhere (Method), the "CustomSQLFrom (Method) " on page 38, and the "ResetCustomerSQL (Method)" on page 117 correspond to the edit SQL functionality in the Custom SQL dialog box. However, no Custom SQL dialog will display when this method is executed.

Note: To use the Custom SQL feature, the query data model must have at least one table. If no table exists, then the Console Window displays: "Script(x):uncaught exception:Invalid String".

Syntax:

```
Expression.CustomSQLWhere(CustomSQLStr As String)
```

Expression Required:

An expression that returns a query object

Example:

This example sets the FROM clause and the WHERE clause, processes the query, and then restores the original SQL statement:

```
//Set the FROM clause, Set the WHERE clause, PROCESS, and then RESET
SQLActiveDocument.Sections["Query"].CustomSQLFrom("FROM From.Sales_Fact, From.Periods,
From.Products") ActiveDocument.Sections["Query"].CustomSQLWhere("WHERE
(Periods.Day_Id=Sales_Fact.Day_Id AND
Products.Product_Id=Sales_Fact.Product_Id)  AND (Periods.Quarter='Q1')")
ActiveDocument.Sections["Query"].Process()
ActiveDocument.Sections["Query"].ResetCustomSQL();
```

Disconnect (Method)

Applies To:

Connection object, MetaDataConnection object

Description:

Drops the connection between the Interactive Reporting database connection file object and the data source.

Syntax:

```
Expression.Disconnect()
```

Expression Required:

An expression that returns a **Connection** object

Example:

This example shows how to disconnect from the database:

```
if (ActiveDocument.Sections["Query"].DataModel.Connection.Connected == true)
 ActiveDocument.Sections["Query"].DataModel.Connection.Disconnect()
```

DoEvents (Method)

Applies To:

Application

Description:

Halts a script from executing and switches control to the operating-environment kernel so that the application can respond to pending or queued events. This method is typically placed at the end of a *for* statement. It is usually included in a script that runs continuously and displays live data.

Note: The Application.DoEvents() object model syntax is not supported in an Interactive Reporting document file deployed on the EPM Workspace.

Syntax:

```
Application.DoEvents()
```

Example:

This script processes a query five times with limits. A DoEvents method is included to display the applied limits each time the query is processed:

```
function Wait(ms)
{
var oStart = new Date();
var oNow = new Date();
     while (oNow.getTime() - oStart.getTime() < ms)
          {
               oNow = new Date() ;
```

```
                DoEvents();
            }
}
for (i=1;i<=5 ;i++)
{
      // do something
      if(ActiveDocument.Sections["Query"].Limits[2].Ignore ==false)
            ActiveDocument.Sections["Query"].Limits[2].Ignore=true;
      else
            ActiveDocument.Sections["Query"].Limits[2].Ignore=false;
      Console.Write("processing number:  "+i+"\n")
      ActiveDocument.Sections["Query"].Process()
         Wait(9000)
}
```

DownloadToResults (Method)

Applies To:

(CubeQuery) QuerySection object

Description:

Makes the CubeQuery data available by way of a table section called OLAPResults. If a section called OLAPResults already exists, the new section will be called OLAPResults2, etc. Only one table section will be created. Second and subsequent invocations of this method activates the table section created the first time this method was called.

Note: Shared members can be excluded from the OLAP query by way of Query Options, however there are some cases where customers might want to include Shared Members in the query and result set, but not in the totals. If you want to include shared members in the results set, the parent context needs to exist in the query. In other words, if a shared member's parent does not exist in the query, Download to Results does not recognize that it is a shared member. When downloading to results a query that has a "ragged" member selection, where some parent members are not expanded to details, the warning message appears: "The validity of the flattened results is not guaranteed if the grid is not fully expanded and symmetric. This is necessary for "parent context" of shared members in a results set – shared member parents need to exist in the query in order to determine that they are shared for the results set:

Note: The Add method for CubeQuery "download to results" dependent reporting sections (Chart, Pivot, Table, Report) does not accept the CubeQuery "Query" section name as the dependent section argument. This is different from use of the Add method with a relational Query section. With CubeQuery, you must use the downloaded "Results" section as the dependent section argument.

Syntax:

```
Expression.DownloadToResults()
```

Example:

This example shows how to download the CubeQuery Section to results:

```
ActiveDocument.Sections["Query"].DownloadToResults()
```

DownloadToResults (Method)

Applies To:

OLAPQuery object

Description:

Enables you to download an OLAPQuery data set to an OLAPResults section within the Interactive Reporting document file. When an OLAPResults section is created, it is refreshed automatically with data from its associated OLAPQuery section at each process of the section. When downloaded, the OLAPQuery section can be integrated with the Chart, Table and reporting sections.

If you expect the query to retrieve a small to medium sized data set, it is recommended that you use the automatic download feature from the OLAP, then Tools, then Options, then Program Options, then OLAP tab, then Auto Generate Results When Processing OLAP Query. If you choose this option, the Results set is not created for the current OLAP query, but only for new OLAP Query sections. Also note that in some circumstances when querying large amounts of data, the automatic creation of an OLAPResults section may result in a slight reduction in the query performance. Using the manual download feature either through the menu command or the DownloadToResults method is the preferred method in this case.

Note: If you create dependent OLAP sections from an OLAPQuery section without database totals and later attempt to enable the database totals function, the local break totals and grand totals may not accurately reflect the correct totals.

Syntax:

```
Expression.DownloadToResults()
```

Example:

The following script was associated with a OnClick event. When the user clicks a command button, an OLAPResults section is generated and displayed in the Catalog pane:

```
ActiveDocument.Sections["OLAPQuery"].DownloadToResults()
```

DrillDown (Method)

Applies To:

TopLabels collection, SideLabels collection, Measures collection

Note: DrillDown and DrillUp Measures are specific to Essbase and DB2 for OLAP.

Description:

Executes the drill-down value set up through the "AddExportSection (Method)" on page 20.

Syntax:

```
Expression.DrillDown()
```

Expression Required:

An expression that executes a drill-down value

Example:

This example shows how to execute the drill down-value *State* added through the AddDrillValue method:

```
ActiveDocument.Sections["OLAPQuery"].SideLabels.AddDrillValue("State")
ActiveDocument.Sections["OLAPQuery"].SideLabels.DrillDown()
```

DrillThrough (Method)

Applies To:

TopLabels collection, SideLabels collection, Measures collection

Description:

Executes the drill-through from a multi-dimensional database to a relational database. The DrillThrough() method assumes that a connection to relational and OLAP queries exists. Each method uses the BqDrillThroughPrompt constant group.

Note: The dialog prompt for DrillThrough() is not a requirement in the EPM Workspace. The Prompt Dialog is ignored and the default Prompt Option is taken. If a Prompt Option is not specified, the default of New Pivot is assumed.

Syntax:

```
Expression.DrillThrough([optional] Number [promptOption, [optional] String pivotName,
[optional] Boolean promptDialog]
```

Also note these optional arguments:

- promptOption (Number)—Specific one of the following prompt options: select 1 for an UpdatePivot, 3 for a NewPivot, or 2 for a Cancel. If you do not specify a prompt option, the default option is 1 (NewPivot).
- pivotName (String)—Specify the name of either the existing pivot to update if UpdatePivot is selected as the PromptOption. If the name is required but not specified, the default name used is the first pivot in the drop-down list (which is also the first in Catalog pane).
- promptDialog (Boolean)—Specify whether the dialog prompts will display. If this option is enabled, all related prompts display. If this option is not enabled, no prompts display and default values are assumed.

Expression Required:

An expression that executes a drill through command

Constants:

The DrillThrough (Method) uses the BqDrillThroughPrompt constant group, which consists of these values:

bqDrillThroughPromptCancel = 2

bqDrillThroughPromptNew = 3

bqDrillThroughPromptUpdate = 1

Example:

This example shows how to drill up a drilled-down value:

ActiveDocument.Sections["OLAPQuery"].DrillThrough(2,"Pivot",true)

DrillUp (Method)

Applies To:

TopLabels collection, SideLabels collection, Measures collection

Description:

Drills up the value drilled down using the "AddExportSection (Method)" on page 20.

Note: DrillDown and DrillUp Measures are specific to Oracle Essbase and DB2 for OLAP.

Syntax:

```
Expression.DrillUp
```

Expression Required:

An expression that executes a drill-up on a label or measure value

Example:

This example shows how to drill-up a side label:

```
ActiveDocument.Sections["OLAPQuery"].SideLabels.DrillUp()
```

Duplicate (Method)

Applies To:

ChartSection object, (CubeQuery) QuerySection object, DataModelSection object, DashboardSection object, OLAPQuerySection object, PivotSection object, QuerySection object, ResultsSection object, TableSection object, ReportSection object

Description:

Creates a copy of a section.

Syntax:

```
Expression.Duplicate()
```

Expression Required:

An expression that returns a reference to a newly created section including:

- QuerySection
- (CubeQuery) QuerySection
- (OLAP) QuerySection
- ChartSection
- DataModelSection
- DashboardSection
- PivotSection
- ReportSection

Example:

This example duplicates the Chart section. The new section label is based on the original section label name, but displays the section label number. For example, if you duplicate the chart three times, the section pane shows: Chart, Chart2, and Chart3:

```
ActiveDocument.Sections["Chart"].Duplicate()
```

EnableMenuItem (Method)

Applies To:

ChartSection object, (CubeQuery) QuerySection object, OLAPQuerySection object, PivotSection object, QuerySection object, ResultsSection object, Section object, TableSection object,

Description:

Returns if a shortcut menu items is enabled for an active embedded section object. This method returns a Boolean value (true or false) and references the BqMenuItem constant group which corresponds to the eligible shortcut menu items in Interactive Reporting Studio,Interactive Reporting Web Client and EPM Workspace. Some shortcut menu items can only be enabled for active embedded section objects in the EPM Workspace and are denoted by "Workspace". Some shortcut menu items can only be enabled in Interactive Reporting Web Client and Interactive Reporting Studio. These shortcut menu items are denoted by "Desktop". Other options can be enabled for both.

Note: When the shortcut menu changes by way of object model for embedded section objects of the same dashboard, the entire dashboard needs to be refreshed for changes to take effect. To refresh entire dashboard, switch to another section and come back to the same section.

For active *Results/Table* embedded section objects, the following shortcut menu items are available when a column or columns is selected:

- Sort Ascending (Desktop and Workspace)
- Sort Descending (Desktop and Workspace)
- Background (Desktop)
- Auto-Size Column (Workspace)

For active *Pivot* embedded section objects, the following shortcut menu items are available by selecting the top or side labels, fact column, fact column label, column title or the pivot handles of the top and side labels. Individual or multiple selections of pivot components is available.

Depending on the selected component selected, the shortcut menu displays the following options:

- Drill Anywhere (Desktop and Workspace)
- Drill Up (Desktop and Workspace)
- Focus on Items (Desktop and Workpace)
- Hide Items (Desktop and Workspace)
- Show Hidden Items (Desktop and Workspace)
- Show All Items (Desktop and Workspace)

- Auto-Size Column Width (Workspace)
- Swing
  - Horizontal (Workspace)
  - Vertical (Workspace)
  - Up (Workspace)
  - Down (Workspace)
  - Left (Workspace)
  - Right (Workspace)
  - Before (Workspace)
  - After (Workspace)
- Sort Ascending (Workspace)
- Sort Descending (Workspace)
- Refresh Pivot (Workspace)

For active *Chart* embedded section objects, shortcut menus can be launched by selecting bars, pie slices, lines, slice labels, or X and Z axis labels. Individual or multiple selections of Chart components is available. Multiple component selection can be performed by selecting Ctrl + Click. This command allows the selection of multiple like components, one at a time (it also allows for the deselection of components one at a time) Depending on the selected component(s) selected, the shortcut menu displays the following options:

The list of available speed menu options is context sensitive and depends on what component of the report is selected (label, bar, slice) and the state when the component selection is made (if items have been hidden, if additional columns are available for drill down, etc).

- Drill Anywhere (Desktop and Workspace)
- Drill Up (Desktop and Workspace)
- Focus on Items (Desktop and Workspace)
- Hide Items (Desktop and Workspace)
- Show Hidden Items (Desktop and Workspace)
- Show All Items (Desktop and Workspace)
- (Un)Group (Workspace)
- Show Negative Values (Desktop)
- Show Pie Outline (Desktop)
- Show Bar Border (Desktop)
- Show Marker Border (Desktop)
- Show Label (Desktop and Workspace)
- Sort Ascending (Workspace)
- Sort Descending (Workspace)

- Zoom (Workspace)
  - In (Workspace)
  - Out (Workspace)
  - Return to original (Workspace)
- Refresh Chart (Workspace)

For active *OLAP Query/CubeQuery* embedded section object shortcut menu items are launched by either selecting the top or side labels, fact column, fact column label or column title or by selecting the handles of the top and side labels. Individual selections of these components can be performed as well as the selection of multiple components. The list of available shortcut menu options is context sensitive and depends on what component of the report is selected (label, bar, slice) and the state when the component selection is made (if items have been hidden, if additional columns are available for drill down, etc).

. Depending on the selected component selected, the shortcut menu displays the following options:

- Keep Only (Desktop and Workspace)
- Remove Only (Desktop and Workspace)
- Drill (Desktop and Workspace)
  - Down (Desktop and Workspace)
  - Up (Desktop and Workspace)
  - Next (Desktop and Workspace)
  - Bottom (Desktop and Workspace)
  - All Descendants (Desktop and Workspace)
  - Siblings (Desktop and Workspace)
  - Same Level (Desktop and Workspace)
  - Same Generation (Desktop and Workspace)
- Suppress (Desktop and Workspace)
  - Missing rows (Desktop and Workspace)
  - Missing columns (Desktop and Workspace)
  - Missing columns (Desktop and Workspace)
  - Zero rows (Desktop and Workspace)
  - Zero columns (Desktop and Workspace)
- Drill-Through (Desktop and Workspace)
- Column width ... (Desktop and Workspace)
- Row height (Desktop and Workspace)

Syntax:

Expression.EnableMenuItem(BeMenuItem as ItemId, Boolean as enable)

Constants:

The BqMenuItem constant group consists of these values:

- bqMenuItem_AddRefLine = 28
- bqMenuItem_AddTrendLine = 27
- bqMenuItem_AutoSizeCol = 5
- bqMenuItem_Background = 4
- bqMenuItem_Drill = 6
- bqMenuItem_DrillAllDesc = 13
- bqMenuItem_DrillAnywhere = 7
- bqMenuItem_DrillBottom = 12
- bqMenuItem_DrillDown = 9
- bqMenuItem_DrillNext = 11
- bqMenuItem_DrillSameGen = 16
- bqMenuItem_DrillSameLevel = 15
- bqMenuItem_DrillSiblings = 14
- bqMenuItem_DrillThrough = 8
- bqMenuItem_DrillUp = 10
- bqMenuItem_FocusOnItem = 17
- bqMenuItem_HideItems = 18
- bqMenuItem_KeepOnly = 38
- bqMenuItem_Refresh = 45
- bqMenuItem_RemoveOnly = 39
- bqMenuItem_SetColWidth = 50
- bqMenuItem_SetRowHeight = 51
- bqMenuItem_ShowAll = 20
- bqMenuItem_ShowBarBorder = 24
- bqMenuItem_ShowHidden = 19
- bqMenuItem_ShowLabel = 26
- bqMenuItem_ShowMarkerBorder = 25
- bqMenuItem_ShowNegative = 22
- bqMenuItem_ShowPieBorder = 23
- bqMenuItem_Sort = 1
- bqMenuItem_SortAsc = 2
- bqMenuItem_SortDesc = 3
- bqMenuItem_SupMissCols = 42

- bqMenuItem_SupMissRows = 41
- bqMenuItem_Suppress = 40
- bqMenuItem_SupZeroCols = 44
- bqMenuItem_SupZeroRows = 43
- bqMenuItem_Swing = 29
- bqMenuItem_SwingAfter = 37
- bqMenuItem_SwingBefore = 36
- bqMenuItem_SwingDow = 33
- bqMenuItem_SwingHorz = 30
- bqMenuItem_SwingLeft = 34
- bqMenuItem_SwingRight = 35
- bqMenuItem_SwingUp = 32
- bqMenuItem_SwingVert = 32
- bqMenuItem_ToggleGroup = 21
- bqMenuItem_Zoom = 48
- bqMenuItem_ZoomIn = 46
- bqMenuItem_ZoomOriginal = 49
- bqMenuItem_ZoomOut =47

Example:

This example show how to enable all shortcut menu items for a Results embedded section object::

```
Results1.EnableMenuItem(bqMenuItem_SortAsc,true)
Results1.EnableMenuItem(bqMenuItem_SortDesc,true)
Results1.EnableMenuItem(bqMenuItem_Background,true)
Results1.EnableMenuItem(bqMenuItem_AutoSizeCol,true)
```

ExecuteBScript (Method)

Applies To:

Application

Description:

Executes Interactive Reporting old scripting language commands. By default, all old scripts are wrapped by this function when they are converted from an old Interactive Reporting document fjle.

Syntax:

```
Expression.ExecuteBScript(Script As String)
```

Expression Required:

An expression that returns an **Application** object

Example:

This example shows a translated 5.x script:

Commands can be separated by (;) semicolons or placed on individual lines:

```
ExecuteBScript("set logon root, 'OCENAME', 'test.oce'")
ExecuteBScript("connect logon root; show doc root, 'sectiontab'; hide doc root,
'requestline'")
```

Export (Method)

Applies To:

ChartSection object, (CubeQuery) QuerySection object, DataModelSection object, Document object, DashboardSection object, OLAPQuerySection object, PivotSection object, QuerySection object, ResultsSection object, Section object, TableSection object, ReportSection object

Description:

Creates a file with the information from a section object. Files can be created using the standard data formats from the BqExportFileFormat constant group.

Note: For UNIX users of the Export and ExportToStream methods,Interactive Reporting Web Client and Interactive Reporting Web Client reference the umask command of the user running the Interactive Reporting Service to determine the three-digit octal code. This code defines the read-write-execute permissions to be turned off. Typically, umask is used in .login or .profile and in the Bourne and C shells. It is a built-in command.

Note: The use of the Export (Method) or ExportToStream (Method) does not display a prompt dialog in the EPM Workspace or Jobs/Scheduler.

Syntax:

```
Expression.Export(Filename As String, [optional] FileFormat As BqExportFileFormat,
[optional]IncludeHeaders As Boolean], [optional] Boolean Prompt), [optional] BqEncoding
Encoding)
```

Expression Required:

An expression that returns an object for any of these sections:

- ChartSection
- DataModelSection
- DashboardSection

- OLAPQuerySection
- PivotSection
- QuerySection
- Section
- TableSection

Constants:

The BqExportFileFormat constant group consists of these values:

- bqExportFormatCSV = 6
- bqExportFormatExcel2 = 3
- bqExportFormatExcel5 = 4
- bqExportFormatExcel8 = 11
- bqExportFormatHTML = 1
- bqExportFormatJPEG = 2
- bqExportFFormatLotus123 = 7
- bqExportFormatOfficeHTML = 9
- bqExportFormatOfficeMHTML = 10
- bqExportFormatPDF = 8
- bqExportFormatText = 5

The BqEncoding constant group values include:

- bqEnc_Arabic_ISO
- bqEnc_Arabic_Windows
- bqEnc_Baltic_ISO
- bqEnc_Baltic_Windows
- bqEnc_CentralEuropean_ISO
- bqEnc_CentralEuropean_Windows
- bqEnc_Chinese_BIG5
- bqEnc_Chinese_GB2312
- bqEnc_Chinese_HZ
- bqEnc_Cyrillic_DOS
- bqEnc_Cyrillic_ISO
- bqEnc_Cyrillic_KOI8R
- bqEnc_Cyrillic_KOI8U
- bqEnc_Greek_ISO
- bqEnc_Greek_Windows

- bqEnc_Hebrew_Windows
- bqEnc_Japanese_EUC
- bqEnc_Japanese_JIS
- bqEnc_Korean
- bqEnc_Thai_Windows
- bqEnc_Turkish_ISO
- bqEnc_Turkish_Windows
- bqEnc_Unicode
- bqEnc_Unicode_UTF16
- bqEnc_Unicode_UTF16_BigEndian
- bqEnc_Unicode_UTF16_LittleEndian
- bqEnc_Unicode_UTF32
- bqEnc_Unicode_UTF32_BigEndian
- bqEnc_Unicode_UTF32_LittleEndian
- bqEnc_Unicode_UTF8
- bqEnc_Vietnamese_Windows
- bqEnc_WesternEuropean_ISO
- bqEnc_WesternEuropean_Windows

For information on exporting an Interactive Reporting document file to HTML format statically, see HTMLExportBreakRowCount (Property) and HTMLVerticalPageBreakUnits (Property) in the *Object Model Guide to Properties and Constants*.

For information on exporting an Interactive Reporting document file to the HTML format dynamically (EPM Workspace), see DHTMLExportBreakRowCount (Property), and DHTMLVerticalPageBreakUnits (Property) in the *Volume 4: Object Model Guide to Properties and Constants*.

Example:

This example shows how to export a Results section to HTML. The first part of the script creates a computed column that displays the contents of the *URL* columns as HTML HREFs:

```
//Call the JavaScript link() method to convert the string to HREFs
var ComputedExpression = "URL.link()"
ActiveDocument.Sections["Results"].Columns.AddComputed("Clickable
URLS",ComputedExpression)
ActiveDocument.Sections["Results"].Export("C:\\HTML\\MyResults.htm",
bqExportFormatHTML,false)
```

ExportToStream (Method)

Applies To:

ChartSection object, (CubeQuery)QuerySection object,, DataModelSection ojbect, Document object, DashboardSection object, OLAPQuerySection object, PivotSection object, QuerySection object, ResultsSection object, Section object, TableSection object, ReportSection object

Description:

Enables you to use data streaming in EPM Workspace. Streamed data displays before the entire file is exported. This feature improves performance.

If the file name and associated path information are specified when this method, this information is ignored when streaming is enabled. In the case of the full clients (Interactive Reporting Studio and Interactive Reporting Web Client), the file name and associated path information are used for writing data to disk with no additional errors cited when streaming is enabled (as in the case of export without streaming).

Note: For UNIX users of the Export and ExportToStream methods, Interactive Reporting references the umask command of the user running the Interactive Reporting Service to determine the three-digit octal code. This code defines the read-write-execute permissions to be turned off. Typically, umask is used in .login or .profile and in the Bourne and C shells. It is a built-in command.

Note: The use of the Export (Method) or ExportToStream (Method) does not display a prompt dialog in the EPM Workspace or Jobs/Scheduler.

Syntax:

```
Expression.String Filename,

BqExportFileFormat FileFormat,

[optional]Boolean IncludeHeaders,

Boolean DataStreaming,

[optional]Boolean Prompt )

[optional]Boolean Prompt, [optional] BqEncoding Encoding)
```

Expression Required:

An expression that returns an object for any of these sections:

- ChartSection
- DataModelSection
- DashboardSection
- OLAPQuerySection

- CubeQuerySection
- PivotSection
- QuerySection
- Section
- TableSection

Constants:

The ExportToStream (Method) uses the BqExportFileFormat constant group, which consists of these values:

- bqExportFormatCSV = 6
- bqExportFormatExcel2 = 3
- bqExportFormatExcel5 = 4
- bqExportFormatExcel8 = 11
- bqExportFormatHTML = 1
- bqExportFormatJPEG = 2
- bqExportFFormatLotus123 = 7
- bqExportFormatOfficeHTML = 9
- bqExportFormatOfficeMHTML = 10
- bqExportFormatPDF = 8
- bqExportFormatText = 5

The BqEncoding constant group values include:

- bqEnc_Arabic_ISO
- bqEnc_Arabic_Windows
- bqEnc_Baltic_ISO
- bqEnc_Baltic_Windows
- bqEnc_CentralEuropean_ISO
- bqEnc_CentralEuropean_Windows
- bqEnc_Chinese_BIG5
- bqEnc_Chinese_GB2312
- bqEnc_Chinese_HZ
- bqEnc_Cyrillic_DOS
- bqEnc_Cyrillic_ISO
- bqEnc_Cyrillic_KOI8R
- bqEnc_Cyrillic_KOI8U
- bqEnc_Greek_ISO

- bqEnc_Greek_Windows
- bqEnc_Hebrew_Windows
- bqEnc_Japanese_EUC
- bqEnc_Japanese_JIS
- bqEnc_Korean
- bqEnc_Thai_Windows
- bqEnc_Turkish_ISO
- bqEnc_Turkish_Windows
- bqEnc_Unicode
- bqEnc_Unicode_UTF16
- bqEnc_Unicode_UTF16_BigEndian
- bqEnc_Unicode_UTF16_LittleEndian
- bqEnc_Unicode_UTF32
- bqEnc_Unicode_UTF32_BigEndian
- bqEnc_Unicode_UTF32_LittleEndian
- bqEnc_Unicode_UTF8
- bqEnc_Vietnamese_Windows
- bqEnc_WesternEuropean_ISO
- bqEnc_WesternEuropean_Windows

Example:

This example shows how to export a Results section in a data stream to HTML. Because the streaming is enabled, no file name and associated path information have been specified:

```
ActiveDocument.Sections["Results"].ExportToStream("", bqExportFormatHTML, false, true,
false)
```

Find (Method)

Applies To:

QueryLabel object

Description:

Finds and returns a collection of members matching the Member argument.

Syntax:

```
Expression.Find(MemberName as String, IncludeSharedMembers as Boolean)
```

The "*" character may be used as a wild-card in the MemberName argument.

If an alias table is in use, the member may be specified using the aliases, for example, "Cola" instead of "100-10".

The IncludeSharedMembers argument specifies whether to include shared members in the results of the search or not.

Example 1:

This example shows how to determine which returned members to add using the Find (Method) and Add (Method):

```
var prodLabel = Sections["Query"].Rows["Product"];

// Find and add all instances of members starting with 100- to the Product label
// Results are returned as an anonymous collection
var results = prodLabel.Find("100-*, true);
for (var i = 1; i <= results.Count; i++) {
    var m = results[i].Name;
    prodLabel.Add(m, bqOlapSelectorMember);
}
```

Example 2:

This example shows how to use the Find (Method) with various control objects:

```
//Clear List Box
ActiveDocument.Sections["QueryLabel - Find"].Shapes["ListBox1"].RemoveAll()

//Get Dimension to search
var DimSel = ActiveDocument.Sections["QueryLabel -
Find"].Shapes["DropDown1"].SelectedIndex
var strDimension = ActiveDocument.Sections["Query"].Catalog.Dimensions.Item(DimSel).Name

//Get Include Shared Members Value
var strIncludeSharedMembers = false
if (ActiveDocument.Sections["QueryLabel - Find"].Shapes["RadioButton1"].Checked)
    strIncludeSharedMembers = true
else if (ActiveDocument.Sections["QueryLabel - Find"].Shapes["RadioButton2"].Checked)
    strIncludeSharedMembers = false

//Get Text to search for
var strSearchMember = ActiveDocument.Sections["QueryLabel -
Find"].Shapes["TextBox5"].Text

//Find using user entered search string and write to Results set
var Results = ActiveDocument.Sections["Query"].Rows[strDimension].Find(strSearchMember,
strIncludeSharedMembers)

//Get the hits from QueryLabel.FindResults collection
var intNumOfHits = Results.Count

//Set Counter
var intCount = 1

//Loop throught Items() and populate listbox
do
{
```

```
    if (ActiveDocument.Sections["QueryLabel - Find"].Shapes["RadioButton3"].Checked)
        ActiveDocument.Sections["QueryLabel -
Find"].Shapes["ListBox1"].Add(Results[intCount].Name)
    else if (ActiveDocument.Sections["QueryLabel -
Find"].Shapes["RadioButton4"].Checked)
        ActiveDocument.Sections["QueryLabel -
Find"].Shapes["ListBox1"].Add(Results[intCount].UniqueName)
    intCount ++
}
while (intCount <= intNumOfHits );
```

FindAndAdd (Method)

Applies To:

QueryLabel object

Description:

Adds the specified member to the label.

Syntax:

```
Expression.FindAndAdd(MemberName as String, AddAll as Boolean)
```

If the member does not have a unique name, then the first member with the given name is added if AddAll is false, otherwise all instances of the member are added.

The "*" character may be used as a wild-card in the MemberName argument.

If an alias table is in use, the member may be specified using the aliases, for example "Cola" instead of "100-10".

An ItemNotFound exception will be thrown if the supplied member name does not exist.

Example:

This example shows various methods for finding and adding members to the Product label.

```
// Add the first instance of member 100-20 to the Product label.
Sections["Query"].Columns["Product"].FindAndAdd("100-20");

// Add all instances of member 100-20 to the Product label.
Sections["Query"].Columns["Product"].FindAndAdd("100-20", true);

// Add all instances of members starting with 100- to the Product label.
Sections["Query"].Columns["Product"].FindAndAdd("100-*", true);

// Assuming an alias table is in use, add all instances of members
// starting with "Cola" to the Product label.
Sections["Query"].Columns["Product"].FindAndAdd("Cola*", true);
```

FindItems

Applies To:

Images collection

Description:

Determines a group of images with the same name in the Resource Manager. The return value of this methods is a pseudo-array containing all images in the Resource Manager with display name equal to selected. The first item in the array is at ordinal 1.

Example

This example shows how to find a group of images with the same name in the Resource Manager. The return value is captured by the Count property and displayed in a text box.

```
//Clear all TextBoxes except TextBox5
TextBox1.Text = ""
TextBox2.Text = ""
TextBox3.Text = ""

var ActionName = "RM FindItems()"

TextBox1.Text ="Start " + ActionName

strDisplayName = TextBox5.Text

if (TextBox5.Text == "")
{

try
{

arrFindRM= ActiveDocument.ResourceManager.Images.FindItems("Waves")
//arrFindRM= ActiveDocument.ResourceManager.Images.FindItems("")

TextBox3.Text = "Count for this image is: " + arrFindRM.Count

ActiveSection.Shapes["ListBox2"].RemoveAll()

for (i = 1; i <= arrFindRM.Count; i++)
{
Console.Writeln(arrFindRM[i].Name)
ActiveSection.Shapes["ListBox2"].Add(arrFindRM[i].Name)
} //end for

} //end try
catch(e)
{
TextBox2.Text = "Caught: " + e.toString()
}//end catch

} //end if
else
{
```

```
try
{
arrFindRM= ActiveDocument.ResourceManager.Images.FindItems(TextBox5.Text)

TextBox3.Text = "Count for this image is: " + arrFindRM.Count

ActiveSection.Shapes["ListBox2"].RemoveAll()

for (i = 1; i <= arrFindRM.Count; i++)
{
Console.Writeln(arrFindRM[i].Name)
ActiveSection.Shapes["ListBox2"].Add(arrFindRM[i].Name)
} //end for

} //end try
catch(e)
{
TextBox2.Text = "Caught: " + e.toString()
}//end catch


}

TextBox1.Text ="End " + ActionName
```

FocusSelection (Method)

Applies To:

AxisLabels (XLabels, YLabels, and ZLabels)

Description:

Enables you to single out selected label value items, providing you with the ability to concentrate your view to particular items of interest.

Note: You must specify the label value item in an array before using the FocusSelection (Method).

Syntax:

```
Expression.FocusSelection(ItemArray As Value)
```

Expression Required:

An expression that focuses a **LabelValues** item

Example:

This example shows how to include LabelValues items 1 and 3 in an array and then focus them in the Chart:

```
var NewArray = new Array()
NewArray[0]=ActiveDocument.Sections["AllChart"].XLabels.LabelValues.Item(1)
NewArray[1]=ActiveDocument.Sections["AllChart"].XLabels.LabelValues.Item(2)
ActiveDocument.Sections["AllChart"].XLabels.FocusSelection(NewArray)
```

GetCell (Method)

Applies To:

Column, ResultsSection object, TableSection

Description:

Returns the value of a cell in a Results or Table section.

Syntax:

```
Expression.GetCell(nRow As Long) as variant
```

```
Expression.GetCell(nRow As Long, nCol as Long)
```

Expression Required:

An expression that returns a **Column or a TableSection** object

Example:

This example shows how to populate a list box using Results section values:

```
var MyList = ActiveDocument.Sections["Dashboard"].Controls["ListBox"]
var RowCount = ActiveDocument.Sections["Results"].RowCount
var MyCol  = ActiveDocument.Sections["Results"].Columns["State"]
for (j = 1 ; j <= RowCount ; j = j+1)
{
  var Temp = MyCol.GetCell(j)
  MyList.Add(Temp)
}
```

GetLabelText (Method)

Applies To:

TrendLine object, ReferenceLine object

Description:

Returns the label text for the specified z layer (by default, 1).

Syntax:

```
Expression.GetLabelText([optional] Number z)
```

Example:

This example shows how to return the label text from trend lines 1 and 2, and writes them to the Console window:

```
Console.Write(ActiveDocument.Sections["Chart"].TrendLines[1].GetLabelText())
Console.Write(ActiveDocument.Sections["Chart"].TrendLines[2].GetLabelText())
```

GetRSquared (Method)

Applies To:

TrendLine object

Description:

Returns the calculated goodness of fit (R-squared) value for the selected z layer (1 by default).

Syntax:

```
Expression.GetRSquared([optional] Number z)
```

Example:

This example shows how to return the goodness of fit value from trend lines 2, and write it to the Console window:

```
Console.Write(ActiveDocument.Sections["Chart6"].TrendLines[2].GetRSquared())
```

GetValue (Method)

Applies To:

ReferenceLine object

Description:

Returns the reference line value for the selected z layer.

Syntax:

```
Expression.GetValue[optional] Number z)
```

Example:

This example show to return the reference line value in the Console window:

```
Console.Write(ActiveDocument.Sections["Chart8"].ReferenceLines[1].GetValue())
```

Hide (Method)

Applies To:

Chart Fact object

Description:

Enables you to conceal a Chart fact object. When executed for a Chart object, the selected item is removed from the Y-Facts area of the Chart Outliner.

Syntax:

```
Expression.Hide()
```

Expression Required:

An expression that hides a **Chart fact object**

Example:

This example shows how to conceal the *Amount Sales* fact:

```
ActiveDocument.Sections["Chart"].Facts["Amount Sales"].Hide()
```

HideSelection (Method)

Applies To:

AxisLabels (XLabels, Ylabels and ZLabels)

Description:

Enables you to conceal label value items. Use this method to concentrate your view on selected item(s). To hide a Chart YFact object, use the YFact. Although XLabels and ZLabels behavior is simple (labels on the X and Z axis, i.e. Q1, Q2 ...), the behavior of YLabels is more complex. YLabels are the layers in the Y direction. For example, a stack bar could have 3 layers in the Y direction although there are two items in the YFact outliner.

Syntax:

```
Expression.HideSelection(ItemArray As Value)
```

Expression Required:

An expression that hides a **LabelValues** item

Example:

This example shows how to include LabelValues items 1 and 3 in an array but hide them in the Chart:

```
var NewArray = new Array()
```

```
NewArray[0]=ActiveDocument.Sections["AllChart"].XLabels.LabelValues.Item(1)
NewArray[1]=ActiveDocument.Sections["AllChart"].XLabels.LabelValues.Item(2)
ActiveDocument.Sections["AllChart"].XLabels.HideSelection(NewArray)
```

ImportDataFile (Method)

Applies To:

Sections collection, WebClientDocument object

Description:

Imports a data file into a Query section.

Syntax:

```
Expression.Import(Filename As String, FileType As BqImportDataFileFormat [optional],
BqEncoding Encoding)
```

Expression Required:

An expression that returns a **Sections** object

Constants:

The BqImportDataFileFormat constant group contains these values:

- bqImportFormatCommaText
- bqImportFormatExcel
- bqImportFormatTabText

The BqEncoding group values include:

- bqEnc_Arabic_ISO
- bqEnc_Arabic_Windows
- bqEnc_Baltic_ISO
- bqEnc_Baltic_Windows
- bqEnc_CentralEuropean_ISO
- bqEnc_CentralEuropean_Windows
- bqEnc_Chinese_BIG5
- bqEnc_Chinese_GB2312
- bqEnc_Chinese_HZ
- bqEnc_Cyrillic_DOS
- bqEnc_Cyrillic_ISO
- bqEnc_Cyrillic_KOI8R
- bqEnc_Cyrillic_KOI8U

- bqEnc_Greek_ISO
- bqEnc_Greek_Windows
- bqEnc_Hebrew_Windows
- bqEnc_Japanese_EUC
- bqEnc_Japanese_JIS
- bqEnc_Korean
- bqEnc_Thai_Windows
- bqEnc_Turkish_ISO
- bqEnc_Turkish_Windows
- bqEnc_Unicode
- bqEnc_Unicode_UTF16
- bqEnc_Unicode_UTF16_BigEndian
- bqEnc_Unicode_UTF16_LittleEndian
- bqEnc_Unicode_UTF32
- bqEnc_Unicode_UTF32_BigEndian
- bqEnc_Unicode_UTF32_LittleEndian
- bqEnc_Unicode_UTF8
- bqEnc_Vietnamese_Windows
- bqEnc_WesternEuropean_ISO
- bqEnc_WesternEuropean_Windows

Example:

This example shows how to import a comma separated data file:

```
var Filename = "C:\\Imports\\SalesData.csv"
var MySection = ActiveDocument.Sections.ImportDataFile(Filename,
bqImportFormatCommaText)
```

ImportSQLFile (Method)

Applies To:

QuerySection

Description:

Imports a complete SQL statement from a text file to a query, and retrieves the data set from the database server. Files you import are scanned to determine the number of columns returned by the SQL. The request line is populated with indicators for each column. This enables you to use existing SQL statements.

Before using this method, perform these tasks:

- Connect to the database server.
- Ensure that the Query section to which you import SQL does not have tables.
- Ensure that the SQL file begins with a SELECT statement
- You know how many columns to display in the Results section.

When the SQL file imports, you can drag items from the table to the Request line, use the Custom SQL feature, or display its properties. You cannot edit the imported SQL file, but you can specify a user-friendly Request line item name and identify the data type.

Syntax:

```
Expression.ImportSQLFile(Filename As String,numColumns As Number, [optional] BqEncoding
Encoding)
```

Expression Required:

An expression that returns a Query object

Constants:

The ImportSQLFile (Method) uses the BqEncoding constant group, which includes:

- bqEnc_Arabic_ISO
- bqEnc_Arabic_Windows
- bqEnc_Baltic_ISO
- bqEnc_Baltic_Windows
- bqEnc_CentralEuropean_ISO
- bqEnc_CentralEuropean_Windows
- bqEnc_Chinese_BIG5
- bqEnc_Chinese_GB2312
- bqEnc_Chinese_HZ
- bqEnc_Cyrillic_DOS
- bqEnc_Cyrillic_ISO
- bqEnc_Cyrillic_KOI8R
- bqEnc_Cyrillic_KOI8U
- bqEnc_Greek_ISO
- bqEnc_Greek_Windows
- bqEnc_Hebrew_Windows
- bqEnc_Japanese_EUC
- bqEnc_Japanese_JIS
- bqEnc_Korean

- bqEnc_Thai_Windows
- bqEnc_Turkish_ISO
- bqEnc_Turkish_Windows
- bqEnc_Unicode
- bqEnc_Unicode_UTF16
- bqEnc_Unicode_UTF16_BigEndian
- bqEnc_Unicode_UTF16_LittleEndian
- bqEnc_Unicode_UTF32
- bqEnc_Unicode_UTF32_BigEndian
- bqEnc_Unicode_UTF32_LittleEndian
- bqEnc_Unicode_UTF8
- bqEnc_Vietnamese_Windows
- bqEnc_WesternEuropean_ISO
- bqEnc_WesternEuropean_Windows

Example:

This example shows how to set the imported SQL file name, and process the query:

```
var Filename = "C:\\Program Files\\Hyperion\\BIPlus\\docs\Samples\\SQLLoad\
\SalesData.sql"
var MySection = ActiveDocument.Sections["Query"].ImportSQLFile(Filename, 2)
ActiveDocument.Sections["Query"].Process()
```

InterruptQueryProcess (Method)

Applies To:

Document object

Description:

Interactive Reporting document file level function that stops the processing sequence and should be used only with the "OnPreProcess (Method)" on page 96 event. This method takes no arguments.

Syntax:

```
Expression.InterruptQueryProcess()
```

Expression Required:

Interactive Reporting document file

Example:

This example displays the InterruptQueryProcess method for an active Interactive Reporting document file:

```
ActiveDocument.InterruptQueryProcess()
```

Item (Method)

Applies To:

AggregateLimits collection, AppendQueries collection, AliasTableName collection, Association collection, ColorRanges collection, Columns collection, Controls collection, ControlsDropDown object, ControlsListBox object, ColorRanges collection, DMCatalogItems collection, DMResults collection, Documents collection, Fact collection, Joins collection, Limits collection, LimitValues collection, ListSelection object, OLAPCatalog object, OLAPCatalogNew object, OLAPLevelOrHierarchyNew collection, Parentheses collection, PivotLabel object, PivotLabelTotals collection, QueryLabel object, RecentFiles collection, Requests collection, Results collection, Sections collection, SortItems collection, Shapes collection, Themes collection, TargetFact collection, Toolbars collection, TopicItems collection, TopLabels collection, Topics collection,, Values collection

Description:

This is the accessor function for all collections. Item is the default method used by all collections. It returns the value of an item in a collection referred to by the name or index.

Note: The Session.Form.Item (), Session.URL.Item(), and Session.Cookies.Item() object model syntax is *not* supported in an Interactive Reporting document file deployed in the EPM Workspace.

Syntax:

```
Expression.Item(NameOrIndex) As Object
```

Expression Required:

An expression that returns an object for any of these objects:

- Column
- Control
- ControlsDropDown
- ControlsListBox
- DerivableQueries
- DMCatalogItem
- DMResults

- Document
- Images
- Join
- LabelValues
- Limit
- LimitValues
- ListSelection
- LocalJoins
- LocalResults
- OLAPLabel
- OLAPMeasure
- OLAPSlicer
- PivotLabel
- PivotLabelValue
- RecentFiles
- Request
- Section
- Shape
- SortItems
- Toolbar
- TopicItem
- Topic
- TrendLines

Example:

This example shows how to return the third section named *Query* in the current Interactive Reporting document file:

```
var MySection = ActiveDocument.Sections.Item(3)
or
var MySection = ActiveDocument.Sections[3]
or
var MySection = ActiveDocument.Sections.Item("Query")
or
var MySection = ActiveDocument.Sections["Query"]
```

ItemIndex (Method)

Applies To:

List box object

Description:

Sets the index value of each value in a list box.

Syntax:

```
Expression.ItemIndex(Number nIndex)
```

Example:

This example shows how to return the third index value:

```
ListBox1.SelectedList.ItemIndex()
```

Layer (Method)

Applies To:

Field object, Table object, ReportPivot collection, ReportChart collection, Shapes collection

Description:

Sets the value of the layer value for an object in the Report section. A single object can be layered (stacked) in relative position to other objects. The layer options include four rearrangement options: Send to Front, Send to Back, Bring Forward, and Send Backward.

Send to Front—brings the object all the way front and puts the object at the front of the stack.

Send to Back —sends the object all the way back and puts the object on the bottom of the stack. For example, if there are a square on the bottom, a triangle on top of the square and a circle on top of the triangle, and you apply Send to Back to the circle, it places the circle at the bottom of the stack. The new order of the objects from bottom to top is: circle, square, and triangle.

Bring Forward—brings an object forward one layer. For example, if there are a square on the bottom, a triangle on top of the square and a circle on top of the triangle, and you apply *Bring Forward* to the triangle, it places the triangle at the front of the stack. The new order of the objects from top to bottom is: triangle, circle, and square.

Send Backward – sends the object back one layer. Given the same initial placement of triangle, square, and circle layered from bottom to top, applying *Send Backward* to the circle places the circle one layer down. The new order of the objects from bottom to top is: square, circle, and triangle.

Syntax:

```
Expression.Spring(Name as String)
```

Expression Required:

An expression that layers a report object

Constants:

The Layer method uses the BqLayer constant group. This group consists of these values:

- bqLayerBack
- bqLayerBackward
- bqLayerForward
- bqLayerFront

Example:

This example shows how to reposition the Pivot object one position forward:

```
ActiveDocument.Sections["Report"].Body.Pivots["Pivot"].Layer(bqLayerForward)
```

LoadFromFile (Method)

Applies To:

Limit object

Description:

Loads a list of values into a limit from a file.

Syntax:

```
Expression.LoadFromFile(Filename As String) As Boolean
```

Expression Required:

An expression that returns a **Limit** object

Example:

This example loads a list of values from a file named *limits.txt* into a query limit on the *Store_Id* topic item:

```
var Filename = "d:\\LimitData.txt"
ActiveDocument.Sections["Query"].Limits["Store_Id"].LoadFromFile(Filename)
```

LoadSharedLibrary (Method)

Applies To:

Application

Description:

Initializes the communication between Interactive Reporting and an external shared library (.DLL). It also returns a SharedLibrary object that can invoke functions of the shared library.

Note: The Application.LoadSharedLibrary() object model syntax is not supported in an Interactive Reporting document file deployed in the EPM Workspace.

Syntax:

```
Expression.LoadSharedLibrary(Name As String) As SharedLibrary
```

Expression Required:

An expression that returns an **Application** object

Example:

This example calls the Beep function of the Kernal32.dll for 4 seconds with 5000Hz:

```
var oLibrary;
oLibrary = LoadSharedLibrary("kernel32.dll");
oLibrary.Call("Beep", "UI,UI", 5000, 4000);
```

MenuItemEnabled (Method)

Applies To:

ChartSection object, (CubeQuery) QuerySection object, OLAPQuerySection object, PivotSection object, QuerySection object, ResultsSection object, Section object, TableSection object,

Description:

Returns if a shortcut menu items is enabled for an active embedded section object. This method returns a Boolean value (true or false) and references the BqMenuItem constant group which corresponds to the eligible shortcut menu items in Interactive Reporting Studio,Interactive Reporting Web Client and EPM Workspace. Some shortcut menu items can only be enabled for active embedded section objects in the EPM Workspace and are denoted by “Workspace”. Some shortcut menu items can only be enabled in Interactive Reporting Web Client and Interactive Reporting Studio. These shortcut menu items are denoted by “Desktop”. Other options can be enabled for both.

Note: When the shortcut menu changes by way of object model for embedded section objects of the same dashboard, the entire dashboard needs to be refreshed for changes to take effect. To refresh entire dashboard, switch to another section and come back to the same section.

For active *Results/Table* embedded section objects, the following shortcut menu items are available when a column or columns is selected:

- Sort Ascending (Desktop and Workspace)
- Sort Descending (Desktop and Workspace)
- Background (Desktop)
- Auto-Size Column (Workspace)

For active *Pivot* embedded section objects, the following shortcut menu items are available by selecting the top or side labels, fact column, fact column label, column title or the pivot handles of the top and side labels. Individual or multiple selections of pivot components is available.

Depending on the selected component selected, the shortcut menu displays the following options:

- Drill Anywhere (Desktop and Workspace)
- Drill Up (Desktop and Workspace)
- Focus on Items (Desktop and Workpace)
- Hide Items (Desktop and Workspace)
- Show Hidden Items (Desktop and Workspace)
- Show All Items (Desktop and Workspace)
- Auto-Size Column Width (Workspace)
- Swing
    - Horizontal (Workspace)
    - Vertical (Workspace)
    - Up (Workspace)
    - Down (Workspace)
    - Left (Workspace)
    - Right (Workspace)
    - Before (Workspace)
    - After (Workspace)
- Sort Ascending (Workspace)
- Sort Descending (Workspace)
- Refresh Pivot (Workspace)

For active *Chart* embedded section objects, shortcut menus can be launched by selecting bars, pie slices, lines, slice labels, or X and Z axis labels. Individual or multiple selections of Chart components is available. Multiple component selection can be performed by selecting Ctrl + Click. This command allows the selection of multiple like components, one at a time (it also

allows for the deselection of components one at a time) Depending on the selected component(s) selected, the shortcut menu displays the following options:

The list of available speed menu options is context sensitive and depends on what component of the report is selected (label, bar, slice) and the state when the component selection is made (if items have been hidden, if additional columns are available for drill down, etc).

- Drill Anywhere (Desktop and Workspace)
- Drill Up (Desktop and Workspace)
- Focus on Items (Desktop and Workspace)
- Hide Items (Desktop and Workspace)
- Show Hidden Items (Desktop and Workspace)
- Show All Items (Desktop and Workspace)
- (Un)Group (Workspace)
- Show Negative Values (Desktop)
- Show Pie Outline (Desktop)
- Show Bar Border (Desktop)
- Show Marker Border (Desktop)
- Show Label (Desktop and Workspace)
- Sort Ascending (Workspace)
- Sort Descending (Workspace)
- Zoom (Workspace)
    - In (Workspace)
    - Out (Workspace)
    - Return to original (Workspace)
- Refresh Chart (Workspace)

For active *OLAP Query/CubeQuery* embedded section object shortcut menu items are launched by either selecting the top or side labels, fact column, fact column label or column title or by selecting the handles of the top and side labels. Individual selections of these components can be performed as well as the selection of multiple components. The list of available shortcut menu options is context sensitive and depends on what component of the report is selected (label, bar, slice) and the state when the component selection is made (if items have been hidden, if additional columns are available for drill down, etc).

. Depending on the selected component selected, the shortcut menu displays the following options:

- Keep Only (Desktop and Workspace)
- Remove Only (Desktop and Workspace)
- Drill (Desktop and Workspace)
    - Down (Desktop and Workspace)

    - Up (Desktop and Workspace)
    - Next (Desktop and Workspace)
    - Bottom (Desktop and Workspace)
    - All Descendants (Desktop and Workspace)
    - Siblings (Desktop and Workspace)
    - Same Level (Desktop and Workspace)
    - Same Generation (Desktop and Workspace)
- Suppress (Desktop and Workspace)
    - Missing rows (Desktop and Workspace)
    - Missing columns (Desktop and Workspace)
    - Missing columns (Desktop and Workspace)
    - Zero rows (Desktop and Workspace)
    - Zero columns (Desktop and Workspace)
- Drill-Through (Desktop and Workspace)
- Column width ... (Desktop and Workspace)
- Row height (Desktop and Workspace)

Syntax:

Expression.EnableMenuItem(BeMenuItem as ItemId, Boolean as enables)

Constants:

The BqMenuItem constant group consists of these values:

- bqMenuItem_AddRefLine = 28
- bqMenuItem_AddTrendLine = 27
- bqMenuItem_AutoSizeCol = 5
- bqMenuItem_Background = 4
- bqMenuItem_Drill = 6
- bqMenuItem_DrillAllDesc = 13
- bqMenuItem_DrillAnywhere = 7
- bqMenuItem_DrillBottom = 12
- bqMenuItem_DrillDown = 9
- bqMenuItem_DrillNext = 11
- bqMenuItem_DrillSameGen = 16
- bqMenuItem_DrillSameLevel = 15
- bqMenuItem_DrillSiblings = 14
- bqMenuItem_DrillThrough = 8

- bqMenuItem_DrillUp = 10
- bqMenuItem_FocusOnItem = 17
- bqMenuItem_HideItems = 18
- bqMenuItem_KeepOnly = 38
- bqMenuItem_Refresh = 45
- bqMenuItem_RemoveOnly = 39
- bqMenuItem_SetColWidth = 50
- bqMenuItem_SetRowHeight = 51
- bqMenuItem_ShowAll = 20
- bqMenuItem_ShowBarBorder = 24
- bqMenuItem_ShowHidden = 19
- bqMenuItem_ShowLabel = 26
- bqMenuItem_ShowMarkerBorder = 25
- bqMenuItem_ShowNegative = 22
- bqMenuItem_ShowPieBorder = 23
- bqMenuItem_Sort = 1
- bqMenuItem_SortAsc = 2
- bqMenuItem_SortDesc = 3
- bqMenuItem_SupMissCols = 42
- bqMenuItem_SupMissRows = 41
- bqMenuItem_Suppress = 40
- bqMenuItem_SupZeroCols = 44
- bqMenuItem_SupZeroRows = 43
- bqMenuItem_Swing = 29
- bqMenuItem_SwingAfter = 37
- bqMenuItem_SwingBefore = 36
- bqMenuItem_SwingDow = 33
- bqMenuItem_SwingHorz = 30
- bqMenuItem_SwingLeft = 34
- bqMenuItem_SwingRight = 35
- bqMenuItem_SwingUp = 32
- bqMenuItem_SwingVert = 32
- bqMenuItem_ToggleGroup = 21
- bqMenuItem_Zoom = 48
- bqMenuItem_ZoomIn = 46

- bqMenuItem_ZoomOriginal = 49
- bqMenuItem_ZoomOut = 47

Example:

This example show how to read which shortcut menu items for the Results embedded section object have been enabled, and writes the values to the Console window and a text box:

```
Console.Writeln("Start")
TextBox3.Text = ""

if (Results1.MenuItemEnabled(bqMenuItem_SortAsc) == true)
{
Console.Writeln("bqMenuItem_SortAsc")
TextBox3.Text += "bqMenuItem_SortAsc"
}

if (Results1.MenuItemEnabled(bqMenuItem_SortDesc) == true)
{
Console.Writeln("bqMenuItem_SortDesc")
TextBox3.Text += "\r\nbqMenuItem_SortDesc"
}

if (Results1.MenuItemEnabled(bqMenuItem_Background) == true)
{
Console.Writeln("bqMenuItem_Background")
TextBox3.Text += "\r\nbqMenuItem_Background"
}

if (Results1.MenuItemEnabled(bqMenuItem_AutoSizeCol) == true)
{
Console.Writeln("bqMenuItem_AutoSizeCol")
TextBox3.Text += "\r\nbqMenuItem_AutoSizeCol"
}

Console.Writeln("End")
```

ModifyComputed (Method)

Applies To:

Columns collection

Description:

Enables you to reference an existing column and change its expression while still maintaining the column name. That is, you can recreate the column which might be used by other columns without having to delete it.

Syntax:

```
Expression.ModifyComputed(NameOrIndex As Value, Expression As String)
```

Expression Required:

An expression that returns a Columns object

Example:

The first part of the script adds four undefined computed columns. The second part of the script resolves the errors in the computed column:

```
//This expression causes the four computed items to become undefined
ActiveDocument.Sections["Results"].Columns.AddComputed("Twice","Unit_Sales * 2");
ActiveDocument.Sections["Results"].Columns.AddComputed("Fours","Twice * 2")
ActiveDocument.Sections["Results"].Columns["Twice"].Remove()
ActiveDocument.Sections["Query"].Process()
ActiveDocument.Sections["Results"].Columns.AddComputed("Twice","Unit_Sales * 3");
//This expression resolves the problem
ActiveDocument.Sections["Results"].Columns.AddComputed("Twice","Unit_Sales * 2");
ActiveDocument.Sections["Results"].Columns.AddComputed("Fours","Twice * 2")
ActiveDocument.Sections["Query"].Process()
ActiveDocument.Sections["Results"].Columns.ModifyComputed("Twice", "Unit_Sales *3";
```

ModifyRepositoryFileAnalyzer (Method)

Applies To:

EmbeddedBrowser object, HyperLink object

Description:

Enables you to modify a Hyperion Web Analysis repository object for an embedded browser object or hyperlink object. When this method executed, the corresponding smartcut is regenerated and the embedded browser or hyperlink is refreshed.

Syntax:

```
Expression.ModifyRepositoryFileAnalyzer(String ServerNameAndPortNumber, String Folder
AndPathAndDocumentName, [optional] BqRepositoryToolbarType RepositoryToolbarType,
[optional] String SmartcutParameters
```

Note: A forward slash (/, or also known as a solidus or virgule) is not recognized as a valid character in an Interactive Reporting document file name argument for this method.

Expression Required:

Parameters:

DocumentName—Required; String; Maps to the name of the object selected from the repository.

SmartcutParams—Optional; String; Maps to the *Smartcut Parameters* control used for any additional parameters appended to the URL.

Example:

This example shows how to add additional parameters to Smartcut for the Hyperion Web Analysis file.

```
ModifyRepositoryFileReports("http://myserver:1800","/MyFolder/MyDocument")
```

ModifyRepositoryFileOther (Method)

Applies To:

EmbeddedBrowser object, HyperLink object

Description:

Enables you to modify an Interactive Reporting document file repository object for an embedded browser object or hyperlink object.

Note: A forward slash (/, or also known as a solidus or virgule) is not recognized as a valid character in an Interactive Reporting document file name argument for this method.

Syntax:

```
Expression.ModifyRepositoryFileOther(DocumentName, [optional]String SmartcutParameters)
```

Expression Required:

Parameters:

DocumentName—Required; String; Maps to the name of the object selected from the repository.

SmartcutParams—Optional; String; Maps to the *Smartcut Parameters* control used for any additional parameters appended to the URL.

Example:

See the example in "ModifyRepositoryFileBQY (Method)" on page 80.

ModifyRepositoryFileBQY (Method)

Applies To:

EmbeddedBrowser object, HyperLink object

Description:

Enables you to modify an Interactive Reporting document file repository object for an embedded browser object or hyperlink object. When this method executed, the corresponding Smartcut is regenerated and the embedded browser or hyperlink is refreshed.

Note: A forward slash (/, or also known as a solidus or virgule) is not recognized as a valid character in an Interactive Reporting document file name argument for this method.

Syntax:

```
Expression. ModifyRepositoryFileBQY(StringDocumentName,[BQYSectionName], [optional]
BqRepositoryToolbarType as RepositoryToolbarType, [optional] String SmartcutParameters)
```

Expression Required:

Parameters:

DocumentName—Required; String; Maps to the name of the object selected from the repository.

BQYSectionName – Optional; String; Maps to the Interactive Reporting document file section name set for the repository object (corresponds to the Toolbar field on the Document Options dialog).

ToolbarType—Optional; RepositoryToolbarType; Maps to the toolbar type set for the Interactive Reporting document file repository object (corresponds to the Toolbar field on the Document Options dialog). Uses the bqRepositoryToolbarType constant.

Expression Required:

Parameters:

SmartcutParams—Optional; String; Maps to the *Other Parameters* control used for any additional parameters appended to the URL.

Constants:

The bqReposiotryToolbarType consists of these values:

- bqRepositoryToolbarNavigation = 3
- bqRepositoryToolbarNone = 0
- bqRepositoryToolbarPaging = 2
- bqRepositoryToolbarPagingNavigation = 4
- bqRepositoryToolbarStandard = 1

Example:

The following example shows how to change a repository Interactive Reporting document file section and toolbar. This script presumes that the existences of TextBox1 through TextBox4. The Console.Writeln statements are written to the log file in the EPM Workspace:

```
var ActionName = "ModifyRepositoryFileBQY"


Console.Writeln("Start " + ActionName)
//Set document argument value
Console.Writeln("Step 1")
//If the textbox is empty, use the default value, otherwise use the value in the textbox
if (TextBox1.Text == "") {
try
```

```
{
//This is the default name of the folder and document as they appear when published var
sDocumentName = "\MyDocuments\MyDocumentName.bqy" }
catch(e)
{
Console.Writeln("Caught 1a: " + e.toString())
}
}
else
{
try
{
var sDocumentName = TextBox1.Text
}
catch(e)
{
Console.Writeln("Caught 1b: " + e.toString())
}
}
//Set section name argument value
Console.Writeln("Step 2")
//If the textbox is empty, use the default value, otherwise use the value in the textbox
if (TextBox2.Text == "") {
try
{
//This is the default name of the section to which the document will open var
sBQYSectionName = "Dashboard" }
catch(e)
{
Console.Writeln("Caught 2a: " + e.toString())
}
}
else
{
try
{
var sBQYSectionName = TextBox2.Text
}
catch(e)
{
Console.Writeln("Caught 2b: " + e.toString())
}
}
//Set Toolbar argument value
Console.Writeln("Step 3")
//If the textbox is empty, use the default value, otherwise use the value in the textbox
if (TextBox3.Text == "") {
try
{
//This sets the document to a standard toolbar
var cToolbarType = bqRepositoryBQYToolbarStandard
}
catch(e)
{
Console.Writeln("Caught 3a: " + e.toString())
}
}
```

```
else
{
try
{
//Use an eval statement here to treat the TextBox contents like a constant var
cToolbarType = eval(TextBox3.Text) }
catch(e)
{
Console.Writeln("Caught 3b: " + e.toString())
}
}
//Set parameter argument value
Console.Writeln("Step 4")
//If the textbox is empty, use the default value, otherwise use the value in the textbox
if (TextBox4.Text == "") {
try
{
//This sets no document parameters
var sDocParams = ""
}
catch(e)
{
Console.Writeln("Caught 4a: " + e.toString())
}
}
else
{
try
{
var sDocParams = TextBox5.Text
}
catch(e)
{
Console.Writeln("Caught 4b: " + e.toString())
}
}
//Use the argument values in the method
Console.Writeln("Step 5")
try
{ActiveSection.Shapes["EmbeddedBrowser1"].ModifyRepositoryFileBQY(sDocumentName,
sBQYSectionName, cToolbarType, sDocParams) }
catch(e)
{
Console.Writeln("Caught 5: " + e.toString())
}
}


 Console.Writeln("End " + ActionName
```

ModifyRepositoryFileBQYJob (Method)

Applies To:

EmbeddedBrowser object, HyperLink object

Description:

Enables you to modify an Interactive Reporting job repository object for an embedded browser object or hyperlink object. When this method executed, the corresponding Smartcut is regenerated and the embedded browser or hyperlink is refreshed.

Syntax:

```
Expression.ModifyRepositoryFileBQYJob(DocumentName, [optional] String BQYSectionName,
[optional] BqRepositoryToolbarType RepositoryToolbarType, [optional] String
SmartcutParameters
```

Note: A forward slash (/, or also known as a solidus or virgule) is not recognized as a valid character in an Interactive Reporting document file name argument for this method.

Expression Required:

Parameters:

DocumentName—Required; String; Maps to the name of the job object selected from the repository.

BQYSectionName—Optional; String; Maps to the Interactive Reporting document file section name set for the repository object (corresponds to the Section field on the Document Options dialog).

RepositoryToolbarType—Optional; BqRepositoryToolbarType; Maps to the toolbar type set for a Interactive Reporting document file repository object (corresponds to the Toolbar field on the Document Options dialog box).

Smartcut Parameters—Optional; String; Maps to the "Smartcut Parameters" control used for any additional parameters appended to the URL.

Constants:

The bqRepositoryToolbarType consists of these values:

- bqRepositoryToolbarNavigation = 3
- bqRepositoryToolbarNone = 0
- bqRepositoryToolbarPaging = 2
- bqRepositoryToolbarPagingNavigation = 4
- bqRepositoryToolbarStandard = 1

Example:

See the example in "ModifyRepositoryFileBQY (Method)" on page 80.

ModifyRepositoryFileSQRJob (Method)

Applies To:

EmbeddedBrowser object, HyperLink object

Description:

Enables you to modify a SQR job repository object for an embedded browser object or hyperlink object. When this method executed, the corresponding smartcut is regenerated and the embedded browser or hyperlink is refreshed.

Syntax:

```
Expression.ModifyRepositoryFileSQRJob(DocumentName, String SmartcutParameters,
[optional] Boolean JobRun])
```

Note: A forward slash (/, or also known as a solidus or virgule) is not recognized as a valid character in an Interactive Reporting document file name argument for this method.

Expression Required:

Parameters:

DocumentName—Required; String; Maps to the name of the object selected from the repository.

JobFileName—Required; String; Maps the name of the job file object displayed from an Interactive Reporting job output repository object

SmartcutParameters—Optional; String; Maps to the "Smartcut Parameters" control used for any additional parameters appended to the URL

JobRun—Optional; Boolean; Maps to the Run Job control.

Example:

See the example in "ModifyRepositoryFileBQY (Method)" on page 80.

ModifyRepositoryFileReports (Method)

Applies To:

EmbeddedBrowser object, HyperLink object

Description:

Enables you to modify a repository object for an embedded browser object or hyperlink object. When this method executes, the corresponding Smartcut is regenerated and the embedded browser or hyperlink is refreshed.

ModifyRepositoryFileReports(String ServerNameAndPortNumber, String FolderAndPathAndDocumentName, [Optional] bqExportFileFormatDisplayFormat, [Optional] bqRepositoryToolbarType, [Optional] String RepositorySmartcutParams)

Note: A forward slash (/, or also known as a solidus or virgule) is not recognized as a valid character in an Interactive Reporting document file name argument for this method.

Expression Required:

Parameters:

ServerNameAndPortNumberName—Required; String; Maps to the server name and port number of the Smartcut for the Interactive Reporting document file.

FolderAndPathAndDocumentName—Required; String; Maps to the folder, path and Interactive Reporting document file name of the Smartcut for the Interactive Reporting document file.

DisplayFormat—Returns the format type in which the Reports document is displayed using the bqExportFileFormat. Valid options are the bqExportFormatHTML and bqExportFormatPDF values.

RepositoryReportsToolbarType—Returns the type of toolbar displayed with a Hyperion Reports document embedded or hyperlinked in the Dashboards section. Valid options are standard or none.

The first and second Reports toolbars (Help/Preferences/LogOff/Help and Repository->MyReport) are hidden for Reports populating an Interactive Reporting document file external content control. The third Reports toolbar, Point of View (POV), is an optional Dashboard, and hidden by default. To display HTML/PDF radio buttons in Reports, show the third Reports toolbar.

RepositorySmartcutParams—Returns any additional parameters appended to the Smartcut url. This is a string property that corresponds to any parameters in the "Smartcut Parameters" edit box on the Options tab.

Example:

This example shows how to add additional parameters to a Smartcut for the Hyperion Reports document

```
ModifyRepositoryFileReports("http://myserver:1800", "/MyFolder/MyDocument",
bqReportsTypeBook)
```

Move (Method)

Applies To:

Groupitems object, ReportGroup object, TableFacts object

Description:

Moves an object in the ReportGroup collection. For example, use this method to reverse the order of items in the Facts pane of the Table Outliner.

Syntax:

```
Expression.Move(LabelNameBefore as String)
```

Expression Required:

An expression that returns an object for the following:

- Group Items object
- ReportGroup object
- TableFacts object

Example:

This example shows how to place the object *Unit Sales* before *Amount Sales* in the TableFacts collection:

```
//State is Report Group 1, City is Report Group2.
//This script should move City on top of State.
//Description:  void Move(String LabelNameBefore)
try
  {
ActiveDocument.Sections["Report"].Groups["Report Group2"].Move("Report Group1")
  }
catch(e)
  {
    Console.Writeln(e.toString())
  }
```

New (Method)

Applies To:

Documents collection

Description:

Creates a blank Interactive Reporting document file.

Syntax:

```
Expression.New([Name As String]) As Document
```

Expression Required:

An expression that returns a **Documents** object

Example:

This example shows how to create an Interactive Reporting document file:

```
var MyName = "JavaScript Test"
var MyDoc = Documents.New(MyName)
MyDoc.Save()
```

OnActivate (Method)

Applies To:

Dashboard Section object, EventScript object

Description:

Interactive Reportingsection level function available regardless of the application state, and that you can access through scripting. The OnActivate() method executes a script stored under the OnActivate event trigger. The method takes no arguments. Any scripts associated with the OnActivate method are executed when you enter a Dashboard section.

Syntax:

```
Expression. OnActivate()
```

Expression Required:

An expression that returns an object for any of the following items:

- ControlsCheckBox
- CommandButton
- List Box
- Radio ButtonGraphicsLine
- Hz Line
- Vt Line
- Rectangle
- Round Rectangle
- Oval
- Text Label
- Picture
- Embedded Section Objects
- Query
- Results
- Pivot

- Chart
- Table
- OLAPQuery
- Dashboard

Example:

This example displays the OnActivate method for an active Dashboard section:

```
ActiveDocument.Sections["Dashboard"].OnActivate()
```

OnCellDoubleClick (Method)

Applies To:

Dashboard Pivot Embedded Section Object (ESO) in Active Mode only, EventScript object

Description:

Setting the OnCellDoubleClick (Method) is equivalent to repeating the action of clicking the last cell clicked in the user interface. This method performs an action which does not persist with (dirty) the Interactive Reporting document file or application. This method does not allow the user to select and individual cell. If the user single clicks cells, the selected row are highlighted in black. If the user double clicks cells, the selected row are highlighted in white.

The OnCellDoubleClick (Method) is only available for Interactive Reporting Studio and Interactive Reporting Web Client. It is not supported in the EPM Workspace and Jobs/Scheduler.

Syntax:

```
No arguments
```

Example:

This example shows how to execute an OnCellDoubleClick (Method) when a command button is clicked. The script includes a try-catch statement if the action is tried and then fails:

```
var ActionName = "OnCellDoubleClick"

TextBox1.Text ="Start " + ActionName

try

{

ActiveSection.Shapes["Pivot1"].OnCellDoubleClick()

}

catch(e)

{

TextBox2.Text = "Caught: " + e.toString()
```

```
}

TextBox1.Text ="End " + ActionName
```

OnChange (Method)

Applies To:

Dashboard Section object, EventScript object

Description:

An Interactive Reporting Dashboard Object level function that is only available when a Dashboard section containing a text box is included in the Interactive Reporting document. The OnChange() method executes a script stored in a Dashboard section text box under the OnChange event trigger. The method takes no arguments. The ActiveDocument.Sections[“Dashboard”].Shapes[“TextBox1”]. OnChange()object model syntax is not supported in an Interactive Reporting document file deployed in the EPM Workspace.

Syntax:

```
Expression.OnChange()
```

Expression Required:

An expression that returns a text box object

Example:

This example shows how to associate an OnChange method in a text box:

```
TextBox1.OnChange()
```

OnClick (Method)

Applies To:

ControlsCheckBox object, ControlsCommandButton object, ControlsDropDown, ControlsOptionsButton, ControlsTextBox, EventScript object, Shape object

Description:

Simulates a user click event. This method exhibits the same behavior as simply clicking on a control. Any scripts associated with an OnClick event are triggered.

The OnClick handler supports the optional bqoEvent parameter, which provides access to the mouse cursor position relative to the Dashboard control. The bqoEvent parameter contains two properties: ClickX and ClickY. When the event occurs, the event handler has access to the event related information needed to process it. For example, the event might require the position of

the mouse cursor for a picture OnClick event, or information about which table column was clicked for a table embedded section object OnClick event.

Syntax:

```
Expression.OnClick([optional] BQEvent as bqoEvent)
```

Expression Required:

An expression that returns an object for any of the following:

- ControlsCheckBox
- ControlsCommandButton
- ControlsDropDown
- ControlsOptionsButton
- ControlsTextBox
- Shape

Example:

This example shows how to invoke a command button event handler:

```
MyDashboard = ActiveDocument.Sections["Dashboard"]
MyDashboard.Controls["CommandButton1"].OnClick()
```

OnClientClick (Method)

Applies To:

ControlsCheckBox, ControlsCommandButton, ControlsDropDown, ControlsListBox, ControlsTextBox, ControlsRadioButton, EventScript object

Description:

Launches a client-side JavaScript OnClientClick event when a user clicks on a control or enters text in a text box in the EPM Workspace. Any client-side JavaScript associated with an OnClientClick event gets triggered. For more information on client-side JavaScript, see Client-Side Events in *Volume 1: Dashboard Design Guide*.

Syntax:

```
Expression.OnClientClick()
```

Expression Required:

An expression that returns an object for any of the following:

- ControlsCheckBox
- ControlsCommandButton

- ControlsDropDown
- ControlsListBox
- ControlsRadioButton
- ControlsTextBox

Example:

This example shows how to invoke a command button event handler:

```
MyDashboard = ActiveDocument.Sections["Dashboard"]
MyDashboard.Controls["CommandButton1"].OnClientClick()
```

OnClientEnter (Method)

Applies To:

ControlsTextBox object, EventScript object

Description:

Launches a client-side JavaScript OnClientEnter event when a user enters a text box in the EPM Workspace. That is, any client-side JavaScript associated with an OnClientEnter event gets triggered. This method is only available when a Dashboard section contains a text box.

Note: For more information on client-side JavaScript, see Client-Side Events in the Dashboard Design Guide.

Syntax:

```
Expression. OnClientEnter()
```

Expression Required:

An expression that returns a Textbox object

Example:

This example shows how to activate a text box:

```
ActiveDocument.Sections["Dashboard2"].Shapes["Textbox1"].OnClientEnter()
```

OnClientExit (Method)

Applies to:

ControlsTextBox object, EventScript object

Description:

Launches a client-side JavaScript OnClientExit event when a user exits a text box in the EPM Workspace. That is, any client-side JavaScript associated with an OnClientExit event gets triggered. This method is only available when a Dashboard section contains a text box.

Note: For more information on client-side JavaScript, see Client-Side Events in the Dashboard Design Guide.

Syntax:

```
Expression. OnClientExit()
```

Expression Required:

An expression that returns a Textbox object

Example:

This example shows how to activate a text box:

```
ActiveDocument.Sections["Dashboard2"].Shapes["Textbox1"].OnClientExit()
```

OnDeactivate (Method)

Applies To:

Dashboard Section, EventScript object

Description:

A Dashboard section level event is available regardless of the application state, and is accessible through scripting. The OnDeactivate() method executes a script stored under the OnDeactivate event trigger. The method takes no arguments. Any scripts associated with the OnDeactivate method are executed when you exit a Dashboard section.

Syntax:

```
Expression. OnDeactivate()
```

Expression Required:

An expression that returns an object for any of the following:

- Controls—CheckBox, CommandButton, List Box, Radio Button
- Graphics—Line, Hz Line, Vt Line, Rectangle, Round Rectangle, Oval, Text Label, Picture
- Embedded Section Objects—Results, Pivot, Chart, Table, OLAPQuery
- Dashboard section script
- Customized script

Example:

This example displays the OnDeActivate method for an active Dashboard section:

```
ActiveDocument.Sections["Dashboard"].OnDeactivate()
```

OnDoubleClick (Method)

Applies To:

Dashboard Section, EventScript object

Description:

An Interactive Reporting Dashboard object level function that is only available when a Dashboard section containing a list box is included in an Interactive Reporting document file. The OnDoubleClick() method executes a script stored in a Dashboard section list box under the OnDoubleClick event trigger. This method takes no arguments.

Syntax:

```
Expression. OnDoubleClick()
```

Expression Required:

An expression that returns a list box object

Example:

This example shows how to associate an OnDoubleClick method with a list box:

```
ListBox1.OnDoubleClick()
```

OnEnter (Method)

Applies To:

Dashboard Section, EventScript object

Description:

The OnEnter() method is an Interactive Reporting Dashboard object level function. The OnEnter event is used to invoke specific processing to execute when a control becomes active.

Note: The ActiveDocument.Sections[“Dashboard”].Shapes[“TextBox1”]. OnEnter()object model syntax is not supported in an Interactive Reporting document deployed on the EPM Workspace.

Syntax:

```
Expression. OnEnter()
```

Expression Required:

An expression that returns a Textbox object

Example:

This example shows how to activate a text box:

```
ActiveDocument.Sections["Dashboard2"].Shapes["Textbox1"].OnEnter()
```

OnExit (Method)

Applies To:

Dashboard Section, EventScript object

Description:

An Interactive Reporting Dashboard object level function triggered when a text box object loses focus. This method is only available when a Dashboard section that contains a text box is included in theInteractive Reporting document file.

Syntax:

```
Expression.OnExit()
```

Expression Required:

An expression that returns a Textbox object

Example:

This example shows how to exit a text box:

```
ActiveDocument.Sections["Dashboard2"].Shapes["Textbox1"].OnExit()
```

OnPostProcess (Method)

Applies To:

Document object, EventScript object

Description:

An Interactive Reporting document file level function that is available for running application through scripting. The OnPostProcess method executes a script stored under the OnPostProcess event trigger. This method takes no arguments.

Note: Calling the "Process (Method)" on page 105 from the "OnPreProcess (Method)" on page 96 or OnPostProcess (Method) events can result in an infinite loop.

Syntax:

```
Expression.OnPostProcess()
```

Expression Required:

An expression that returns an Interactive Reporting document file object

Example:

This example displays the OnPostProcess method for the active Interactive Reporting document file:

```
ActiveDocument.OnPostProcess()
```

OnPreProcess (Method)

Applies To:

Document object, EventScript object

Description:

An Interactive Reporting document level function that executes a script stored under the OnPreProcess event trigger. The method takes no arguments.

Note: Calling the "Process (Method)" on page 105 from the OnPreProcess (Method) or "OnPostProcess (Method)" on page 95 events can result in an infinite loop.

Syntax:

```
Expression.OnPreProcess()
```

Expression Required:

An expression that returns an Interactive Reporting document file object

Example:

This example displays the OnPreProcess method for the active Interactive Reporting document file:

```
ActiveDocument.OnPreProcess()
```

OnRowDoubleClick (Method)

Applies To:

Dashboard Section, EventScript object

Description:

An Interactive Reporting Dashboard object level function that is executed when you double-click a row from an active embedded Results/Table section within a Dashboard section.

Syntax:

```
Expression. OnRowDoubleClick()
```

Expression Required:

An expression that returns a Results/Table section object

Example:

This example shows how to associate the OnRowDoubleClick method with an active Table in the Dashboard section:

```
Table1.OnRowDoubleClick()
```

OnSelection (Method)

Applies To:

Dashboard Section, EventScript object

Description:

Interactive Reporting Dashboard object level function that is triggered when the drop-down box selections are made. This method is only available when a Dashboard section is included in the Interactive Reporting document file and the Dashboard section contains a drop down-box.

Syntax:

```
Expression.OnSelection()
```

Expression Required:

Drop-down box

Example:

This example shows how to change the selection in a text box based on the *OnSelection* event:

```
TextLabel.Text = "DropDown OnSelection"
```

OnShutdown (Method)

Applies To:

Document object, EventScript object, Slier object

Description:

An Interactive Reporting document file level function available for running applications through scripting. The OnShutdown method executes a script stored under the OnShutdown event trigger. This method takes no arguments. Any OnShutDown events are executed before you are prompted to save or discard changes made to a document in the Save dialog box.

Syntax:

```
Expression. OnShutdown()
```

Expression Required:

An expression that returns an Interactive Reporting document object

Example:

This example shows how to use the OnShutdown() method to exit an Interactive Reporting document file and save the document

```
Documents["Untitled"].OnShutdown()
Documents["SaleQuery"].Save()
Documents["SalesQuery"].Save()
```

OnStartup (Method)

Applies To:

Document object, EventScript object

Description:

An Interactive Reporting document level function that executes when an Interactive Reporting document file is opened and that can initialize the Interactive Reporting document file and application for users. This method is available regardless of the state of the application. As long as the application is running, this method is available through scripting. The OnStartup method executes a script stored under the Nostratic event trigger. This method takes no arguments.

Syntax:

```
Expression. OnStartup()
```

Expression Required:

An expression that returns an Interactive Reporting Document object

Example:

This example displays the Nostratic method for an active Interactive Reporting document file:

```
ActiveDocument.OnStartup()
```

Open (Method)

Applies To:

Connection object, MetaDataConnection object, Documents collection

Description:

Documents—Opens an existing Interactive Reporting document file (BQY).

Connection—Opens an existing Interactive Reporting database connection file (OCE).

Note: The Documents.Open() object model syntax is not supported in an Interactive Reporting document file to be deployed in EPM Workspace.

Syntax:

```
Expression.Open(Filename As String)
```

Expression Required:

An expression that returns a Connection, or Documents object

Example 1:

This example shows how to open an existing Interactive Reporting document file:

```
var MyFile = "C:\\BQDocs\\JavaTest.bqy"
var MyDoc = Documents.Open(MyFile)
Alert(MyDoc.Name + " is open")
```

Example 2:

This example shows how to open an existing Interactive Reporting file (.oce):

```
var MyOCE = "C:\\BQDocs\\SQL.oce"
ActiveDocument.Sections["Query"].DataModel.Connection.Open(MyOCE)
ActiveDocument.Sections["Query"].DataModel.Connection.Username = "qa"
ActiveDocument.Sections["Query"].DataModel.Connection.SetPassword("qa")
ActiveDocument.Sections["Query"].DataModel.Connection.Connect()
or
var MyOCE = "C:\\BQDocs\\SQL.oce"
var MyCon = ActiveDocument.Sections["Query"].DataModel.Connection
MyCon.Open(MyOCE)
MyCon.Username = "qa"
MyCon.SetPassword("qa")
MyCon.Connect()
```

OpenURL (Method)

Applies To:

Application

Description:

Requests the browser to open an URL specified by the URL parameter. The target parameter refers to the browser window where the URL should be displayed. Target may be the name of a browser frame or a keyword referring to a specific browser window.

You must include the “http” part of the URL when you specify the URL parameter.

Target Description

“_self” The current browser window.

“_new” A new browser window.

In Avalanche, this mapping occurs for the OpenUrl method:

OpenURL ”_parent” target is mapped to “_top”

Open URL “_self” target is mapped to “_new”

Syntax:

```
Expression.OpenURL(URL As String, Target As String)
```

Expression Required:

An expression that returns an **Application** object.

Example 1:

This example shows how to open a Web page in a new window:

```
if(Application.Name != "Hyperion - Designer")
{
    var MyURL = “http://www.Hyperion.com”
    Application.OpenURL(MyURL)
}
```

Example 2:

This example shows how to open a localInteractive Reporting document file in Interactive Reporting Web Client.

```
Application.OpenURL("E:\\Demo.bqy", "_self")
```

or

```
Application.OpenURL("E:\\Demo.bqy", "_new")
```

PivotThisChart (Method)

Applies To:

PivotCollection

Description:

Changes a Chart object into the form of a Pivot report.

Syntax:

```
Expression.PivotThisChart()
```

Expression Required:

An expression that returns a Pivot object

Example:

This example shows how to change the *BooksChart* chart object into the form of a Pivot report:

```
ActiveDocument.Sections["BooksChart"].PivotThisChart()
```

PivotTo (Method)

Applies To:

PivotLabel collection

Description:

Changes the position of a pivot label. By default, calling the PivotTo method moves a pivot label from one label collection to another. PivotTo performs the same action as selecting or deleting a pivot label from one group and reinserting into a different group.

Syntax:

```
Expression.PivotTo([Index As Number])
```

Expression Required:

An expression that returns a **PivotLabel** object

Example:

This example shows how to pivot a label from the top labels collection to the 1st position in the side labels collection. The Index is an optional property, which specifies where the label pivots. If the property is empty, then the pivot places the label at the end of the list:

```
ActiveDocument.Sections["Pivot"].TopLabels["Year"].PivotTo(1)
//To pivot back to its original position use:
```

```
ActiveDocument.Sections["Pivot"].TopLabels["Year"].PivotTo()
```

PlacementModify (Method)

Applies To:

Shapes collection (Placement node)

Description:

Enables the parent shape or control to be moved or resized.

All parameters must be supplied. If YOffset, Width, or Height are not supplied, YOffset has a value of zero, and Width and Height have a value of one.

Using this method is a quick way to set all Placement properties at once.

Note: To keep the current position or size of a shape when using the Placement.Modify method, supply the current values using the Placement properties.

Syntax:

```
Modify(XOffset, YOffset, Width, Height)
```

PreloadContent (Method)

Applies To:

ActiveDocument object

Description:

Enables the pre-loading of embedded browser objects for the currently active section in order to increase performance in EPM Workspace, especially if the section contains large startup scripts. When the Interactive Reporting Service encounters an embedded browser object, it generates and sends the empty Dashboard HTML with the embedded browser object to the browser, which contains only iframes of the embedded browser object. When the HTML is rendered, the browser processes the URLs of the iframes automatically. At the same time, the Interactive Reporting Service resumes the processing of the section and executes any Object Model script if it exists. Finally the browser is refreshed with the actual Dashboard HTML. This method preloads the section named in the argument. By default, if no section name is specified, the active section is preloaded. The method is valid only if the specified section is of a Dashboard type and contains an embedded browser object.

The PreloadContent (Method) should be exercised carefully, following these recommendations:

- Invoke the PreloadContent (Method) only after the point in which URLs of the embedded browsers are determined.

- All embedded browsers of the preloaded Dashboard are displayed even before that Dashboard is activated.
- Invoke the PreloadContent (Method) only for a single Dashboard that users visit. This is recommended to avoid overloading the browser, which could cause negative performance.

Syntax:

```
Expression.PreloadContent([optional] String SectionName]
```

Expression Required:

An expression that returns section object.

Example:

This example shows how to preload the Dashboard section.

```
ActiveDocument.PreloadContent("Dashboard")
```

PrintOut (Method)

Applies To:

ChartSection object, (CubeQuery)QuerySection object, DataModelSection object, OLAPQuerySection object, PivotSection object, QuerySection object, ResultsSection object, Section object, TableSection object, ReportSection object

Description:

Interactive Reporting document file (BQY) and section level function. This method is available regardless of the state of the application. As long as the application is running, this method is available through scripting. The PrintOut() method in the studio or web client either prints the document according to the arguments provided, or opens a standard OS print box.

If the PrintOut (Method) is encountered while executing a script for a EPM Workspace user, when the server finishes the script processing, the data sent back to the client browser includes a PDF file as part of the data stream.

This PDF data opens in a new window using the Adobe Acrobat Reader (version 7.0.1 or later required), and the Adobe Print dialog is displayed. Due to an issue with Adobe Acrobat Reader, the default settings in the Print dialog are set to print only the first page. Users should reset the Print Range options in the dialog before clicking OK to initiate the printing. Note that the script is run in its entirety before the PDF data is sent to the client browser. This may cause some unexpected behavior with existing documents, such as those with a script that displays an Alert dialog saying “Print Complete”, when in fact it has not even started. Script authors should also be aware of these factors when designing the application.

Use of the Alert method after the Printout() method prevents the Printout() method from executing in an Interactive Reporting document file to be deployed in the EPM Workspace. When used in EPM Workspace mode, the optional parameters of the method, for example pages to print and number of copies are ignored. These options can be specified in the Adobe Acrobat

Reader's Print dialog. The Printout() method is not supported in scripts that run as the result of Document events (such as OnStartup); it is also not supported for section-level activation and deactivation events. The Printout() method has no effect on jobs and does not cause a printout to occur. It is not supported when used as a method of Query or Datamodel type section.

The PrintOut (Methods) takes five arguments. The user can define the start page, end page, number of copies, PrintOut filename and whether or not to prompt for a print dialog box. All arguments are optional, but placeholder empty string e.g. (“”,””,””,””, “”) must be used to pass no value. If insufficient arguments are passed to the method, the PrintOut dialog box open. The user cannot programmatically change the screen location where the PrintOut Dialog box opens.

The first argument is a number. This is equivalent to setting the first number to print in a print dialog box. Value provided must be a number representing the first page to print between 1 and the last page in the Interactive Reporting document file. This is an optional value. This value is ignored in EPM Workspace.

The second argument is a number. This is equivalent to setting the last number to print in a print dialog box. Value provided must be a number representing the last page to print between 1 and the last page in the Interactive Reporting document file. This is an optional value. This value is ignored in the EPM Workspace.

The third argument is a number. This is equivalent to setting the number of copies to be printed in a print dialog box. Value provided must be number representing 1 to n copies. This is an optional value. This value is ignored in the EPM Workspace.

The fourth argument is a string. This is equivalent to setting the filename for printing to file. The string should have a printer specific extension for later use, foe example. .prn or .ps . The file is generated by the default printer driver, not BI, and requires that a default printer be available. Populating this value has the effect of checking the print to file checkbox in the print dialog. The Value provided must be a string. representing a properly formatted local or UNC path. URL syntax is not supported. This is an optional value. This value is ignored in the EPM Workspace. The fifth argument is a boolean. This is equivalent to showing the print dialog or not. Value provided must false/0, or true/any number other than 0. The method does not persist with the Interactive Reporting document file/application.

Note: If an Alert is used after the Printout (Method) in an Interactive Reporting document file deployed in the EPM Workspace, the Printout (Method) does not execute.

Syntax:

```
Expression.PrintOut([optional]FromPage as Number, [optional] ToPage As Number], [Copies
As Long], [optional] Filename As String, [optional]Boolean Prompt)
```

Expression Required:

An expression that returns an object for any of these sections:

- ChartSection
- DataModelSection
- OLAPQuerySection

- CubeQuerySection
- PivotSection
- QuerySection
- Section
- TableSection

Example:

This example shows how to print multiple copies of a Pivot section:

```
var StartPage = 1
var EndPage = 1
var NumCopies =2
ActiveDocument.Sections["Pivot"].PrintOut(StartPage,EndPage,NumCopies)
```

Process (Method)

Applies To:

(CubeQuery)QuerySection object, OLAPQuerySection, QuerySection

Description:

Executes a query. This method is equivalent to selecting the Process Current item from the Tools menu.

Syntax:

```
Expression.Process()
```

Expression Required:

An expression that returns an **OLAPQuerySection or a QuerySection** object

Example:

This example shows how to process every query in an Interactive Reporting document file:

```
for (j =1; j <= ActiveDocument.Sections.Count; j++)
{
     if (ActiveDocument.Sections[j].Type == bqQuery)
     {
             var MyCon = ActiveDocument.Sections[j].DataModel.Connection
              MyCon.Username = "Hyperion"
              MyCon.SetPassword("Hyperion")
              MyCon.Connect()
              ActiveDocument.Sections[j].Process()
              Console.Writeln(ActiveDocument.Sections[j].Name + " was processed.")
    }
}
```

ProcessAll (Method)

Applies To:

(CubeQuery)QuerySection object, OLAPQuerySection, QuerySection

Description:

Executes the *Process All* command for a query. If you have defined a query processing order, queries are processed in the order specified on the Query Processing Order dialog or by the ProcessSequenceNum (Property). For example, in an Interactive Reporting document file with three queries, Query4, Query2, and Query1, the queries are processed in that order.

If no query order was defined, queries are processed in the order in which they appear in the Section Catalog. For example in an Interactive Reporting document file with three queries: Query1, Query2, and Query3, the queries are processed in that order.

Syntax:

```
Expression.ProcessAll()
```

Expression Required:

An expression that returns an **OLAPQuerySection or a QuerySection** object

Example:

This example shows how to display the number of queries in the IInteractive Reporting document file in an Alert box, set the processing the *Query* section to the second position in the Query Processing Order dialog box, include the *Query* section in a *Process All* command, and then execute the Process All command for the Interactive Reporting document file:

```
Alert("Number of Query Sections "   +  ActiveDocument.Sections.QueryCount)
ActiveDocument.Sections["Query"].ProcessSequenceNum = 2
ActiveDocument.Sections["Query"].IncludeInProcessAll = true
ActiveDocument.ProcessAll()
```

ProcessStoredProc (Method)

Applies To:

QuerySection

Description:

Enables you to process stored procedures to obtain results.

This method is used with the "SetStoredProcParam (Method)" on page 124.

Syntax:

```
Expression.ProcessStoredProc()
```

Example:

This example shows how to open and process a stored procedure in the Query section:

```
ActiveDocument.Sections["Query"].SetStoredProcParam("Param1",1)
ActiveDocument.Sections["Query"].SetStoredProcParam("Param2",2)
ActiveDocument.Sections["Query"].ProcessStoredProc()
```

ProcessToTable (Method)

Applies To:

QuerySection

Description:

Executes the query and stores the results as a table on the database. Items on the Request line become the column headings of the new table, and you can append new columns to the table and query it as needed.

Tip: The Interactive Reporting database connection file and database to which you are connecting determine whether or not you can use this feature. You must also have *Create* and *Insert* privileges on the database in order to process to a database table.

Syntax:

```
Expression.ProcessToTable (TableName As String, bqProcessType As String, [optional]
Grantee As String).
```

Note: Grantee is the person to whom access is granted—either PUBLIC, a single user id, or list user identifications that are comma delimited. Grantee is optional because it depends on whether a user is creating a table or appending to an existing table.

Expression Required:

An expression that returns a QuerySection object

Constants:

The BqProcessType is constant group contains the bqProcessCreateTable and bqProcessAppendToTable values.

Example 1:

In this example, the results are stored in a new table entitled MyTable:

```
ActiveDocument.Sections["Query"].ProcessToTable('MyTable', bqProcessCreateTable,
'Public')
```

Example 2:

In this example, the results are appended to "MyTable":

```
ActiveDocument.Sections["Query"].ProcessToTable('MyTable', bqProcessAppendToTable,
'Public')
```

Quit (Method)

Applies To:

Application

Description:

Shuts down the Interactive Reporting application. When the Quit (Method) is executed, the Save dialog box is always prompted before the application can be closed.

Note: The Application.Quit() object model syntax is not supported in an Interactive Reporting document to be deployed in EPM Workspace.

Note: The Quit method does not shut down a browser window.

Syntax:

```
Expression.Quit([Silent As Boolean])
```

Expression Required:

An expression that returns an **Application** object

Example:

This example shows how to quit Interactive Reporting silently:

```
Application.Quit(false)
```

Recalculate (Method)

Applies To:

ChartSection object, (CubeQuery)QuerySection object, DataModelSection object, DashboardSection object, OLAPQuerySection object, PivotSection object, QuerySection object, ResultsSection object, Section object, TableSection object, ReportSection object

Description:

Forces a section to recalculate. This method is useful if you use variables in computed columns.

Syntax:

```
Expression.Recalculate()
```

Expression Required:

An expression that returns an object for the Results and Table sections

Example:

This example forces Results sections to recalculate their values:

```
ActiveDocument.Sections["Results"].Recalculate()
```

Refresh (Method)

Applies To:

DMCatalog object

Description:

Refreshes the tables in the Table Catalog.

Syntax:

```
Expression.Refresh()
```

Expression Required:

An expression that returns a DMCatalog object

Example:

This example shows how to refresh the items in the Table Catalog:

```
ActiveDocument.Sections["Query"].Catalog.Refresh()
```

RefreshAvailableValues (Method)

Applies To:

Limit object

Description:

Generates values for a limit. This is like clicking the Show Values button on the Limit dialog box.

Syntax:

```
Expression.RefreshAvailableValues()
```

Expression Required:

An expression that returns a **Limit** object

Example:

This example shows how to update the available values for the *Unit Sales* limit:

```
ActiveDocument.Sections["SalesQuery"].Limits["Unit Sales"]. RefreshAvailableValues()
```

RefreshDataNow (Method)

Applies To:

ChartSection object, PivotSection object

Description:

Immediately refresh a section that was manually refreshed using the Object Model or user interface. This method is used with the RefreshData (Property) if the property value was set to the bqRefreshDataManually constant.

Syntax:

```
Expression.RefreshDataNow()
```

Expression Required:

An expression that returns an object for Pivot or Chart sections.

Example:

In this example a Pivot section is set to refresh manually; immediately after the command executes:

```
ActiveDocument.Sections["Pivot"].RefreshData=bqRefreshDataManually
ActiveDocument.Sections["Pivot"].RefreshDataNow()
```

Remove (Method)

Applies To:

Association collection, CategoryItems collection, ChartSection object, (CubeQuery)QuerySection object,, Column object, ControlsDropDown object, ControlsListBox object, DataModelSection object, DashboardSection object, Fact collection, Images collection, Join object, Limit object, OLAPQuerySection object, PivotLabels collection, PivotSection object, QueryLabel object, QuerySection object, ReportSection, Requests collection, ResultsSection object, Section object, SortItems collection, TableSection object, TargetFact collection, Topic object. TrendLines collection

Description:

Removes an item from a category. You need not specify a name or index to delete an object.

Syntax:

```
Expression.Remove(NameOrIndex) or Expression.Remove()
```

Expression Required:

An expression that returns an object to any of these items:

- CategoryItems
- ChartSection
- Column
- ControlsDropDown
- ControlsListBox
- DataModelSection
- DashboardSection
- Join
- Limit
- LocalJoin
- LocalResult
- OLAPLabel
- OLAPMeasure
- OLAPQuerySection
- OLAPSlicer
- PivotLabel
- PivotSection
- QuerySection
- Request
- Section
- TableSection
- Topic

Example 1:

This example shows how to remove the *Product ID* column from a Results section:

```
ActiveDocument.Sections["Results"].Columns["Product Id"]. Remove()
```

Example 2:

This example shows how remove the Pivot section from an Interactive Reporting document file (BQY):

```
ActiveDocument.Sections["Pivot"].Remove()
```

RemoveAll (Method)

Applies To:

AggregateLimits collection, Association collection, AxisLabels collection, CategoryItems collection, ColorRanges collection, Columns collection, ControlsDropDown object, ControlsListBox object, ColorRanges collection, Fact collection, Images collection, Joins collection, Limits collection, LimitValues collection, Parentheses collection, PivotLabelsTotals collection, QueryLabel object, Requests collection, TargetFact collection, Topics collection, TopLabels collection, TrendLines collection

Description:

Removes all the items from a collection.

Note: In a CubeQuery section, the RemoveAll method removes all member selectors from the label. This method should not be used as an the argument to MemberSelectors.RemoveAll() in an Interactive Reporting document file. It is only meant to be used forInteractive Reporting document files run as jobs.

Syntax:

```
Expression.RemoveAll()
```

Expression Required:

An expression that returns a collection for any of these items:

- Limits
- AxisLabels
- CategoryItems
- Columns
- ControlsDropDown
- ControlsListBox
- Join
- LimitValues
- LocalJoins
- OLAPLabels

- OLAPMeasures
- OLAPSlicers
- Requests
- Topics

Example:

This example shows how to remove every column from a Results or Table section:

```
ActiveDocument.Sections["Results"].Columns.RemoveAll()
```

RemoveAllTopics (Method)

Applies To:

DefinedJoinPath object

Description:

If you define your own join paths, use the RemoveAllTopics (Method) of the DefinedJoinPaths (Collection) to remove all topics based on the items on the Request and Limit lines. This method corresponds to selecting all available topics on the Define Join Path dialog box and removing the values in the Topics in Join Path list.

Syntax:

```
ActiveDocument.Sections["Query"].DataModel.
JoinsOptions.DefinedJoinPath["MyJoinPath"].RemoveAllTopics()
```

Example:

In the following example all topics are removed from the *MyJoinPath* join path:

```
ActiveDocument.Sections["Query"].DataModel.JoinsOptions.DefinedJoinPath["MyJoinPath"].Re
moveAllTopics()
```

RemoveExportSection (Method)

Applies To:

ChartSection object, DataModelSection object, Document object, DashboardSection object, OLAPQuerySection object, PivotSection object, QuerySection object, Section object, TableSection object

Description:

When sections export, the "Export (Method)" on page 52 clears the export buffer. If sections do not export, use this method to flush the export buffer. This removes all sections set to export from the bugger. For instance, if you set a Report, Pivot, and Chart section for export using the

"AddExportSection (Method)" on page 20, a call to RemoveExportSections (Method) nullifies the export. You could then specify the Export (Method) to export the sections.

Syntax:

```
Expression.RemoveExportSections()
```

Example:

In this example sections are set to export using AddExportSection (Method), cleared from the export buffer using RemoveExportSections (Method), and all Interactive Reporting document file sections are exported using Export (Method):

```
//Export SELECTED Sections of .bqy document
ActiveDocument.AddExportSection('Report')
ActiveDocument.AddExportSection('Report2')
ActiveDocument.AddExportSection('Results')
ActiveDocument.AddExportSection('Table')
ActiveDocument.AddExportSection('Pivot')
ActiveDocument.AddExportSection('Pivot2')
ActiveDocument.AddExportSection('Pivot3')
ActiveDocument.AddExportSection('Chart')
ActiveDocument.AddExportSection('Chart2')
ActiveDocument.AddExportSection('OLAPQuery')
//Flushes the Export buffer
ActiveDocument.RemoveExportSections()
//Export ALL sections of .bqy document since Export buffer was flushed
ActiveDocument.Export('C;\\Temp\\MyExportFile.htm', bqExportFormatHTML)
```

RemoveFilterValue (Method)

Applies To:

TopLabel object, SideLabel object, Measure object

Description:

Removes an entire filter value, a single value based on a literal string value in a filter array, or a single value based on the position in a filter array.

Syntax:

Expression to remove a single value based on literal string value in filter array:

```
ActiveDocument.Sections["OLAPQuery"].Measures["Profit"].RemoveFilterValue()
```

Expression to remove a single value based on a literal string value in a filter array:

```
Expression.RemoveFilterValue(string
SearchStringOrMemberNameOrNumberRowsColumnIndexORUserDefinedAttributeOrSubstitutionVaria
ble)
```

Expression to remove a single value based on the position in a filter array:

```
ActiveDocument.Sections["OLAPQuery"].Measures["Profit"].RemoveFilterValue(number index)
```

Example:

The following example shows how to remove the filter value from the *Line Name* top label:

```
ActiveDocument.Sections["OLAPQuery"].TopLabels["Line Name"].RemoveFilterValue()
```

RemoveRange (Method)

Applies To:

ColorRange object

Description:

Removes a color range object, which includes the color, minimum and maximum values and tool tip.

Example:

This example shows how to remove the second color range for a speedometer gauge:

```
Speedometer.ColorRanges[2].RemoveRange()
```

RemoveShape (Method)

Applies To:

Shapes collection

Description:

RemoveShape(shapeName)

Deletes a shape from a dashboard using COM or JavaScript. Pass the name of the shape to delete as shapeName.

Caution! This operation cannot be undone.

Example:

This example removes a shape called LeftGoal.

```
try{
    Shapes.RemoveShape("LeftGoal")

}catch(e){} // no worries it does not exist
```

RemoveTopic (Method)

Applies To:

DefinedJoinPath collection

Description:

If you chose to define your own join paths, use the RemoveTopic() method of the DefinedJoinPaths (Collection) to remove a topic based on an item on the Request and Limit lines. This method corresponds to selecting all available topics on the Define Join Path dialog box and removing the values in the Topics in Join Path list.

Syntax:

```
ActiveDocument.Sections["Query"].DataModel.JoinsOptions.DefinedJoinPath["MyJoinPath"].Re
moveTopics(String DefinedJoinPathName)
```

Example:

In this example, all topics are removed from the *MyJoinPath* join path:

```
ActiveDocument.Sections["Query"].DataModel.JoinsOptions.DefinedJoinPath["MyJoinPath"].Re
moveTopic("Products")
```

RemoveTotal (Method)

Applies To:

OLAPLabels collection

Description:

Removes the totals rows added to a top or side label column.

Syntax:

```
Expression.RemoveTotal()
```

Example:

In this example, the totals row is removed from the side labels column of the OLAPQuery and the "Label Totals Have been Removed" alert message displays:

```
ActiveDocument.Sections["OLAPQuery"].SideLabels["State"].RemoveTotal()
Application.Alert("Label Totals Have been Removed")
```

ResetCustomerSQL (Method)

Applies To:

QuerySection object

Description:

Resets the original SQL statement before processing and has the Custom SQL window open in a query. The "CustomSQLFrom (Method) " on page 38, "CustomSQLFrom (Method) " on page 38, and ResetCustomSQL (Method) correspond to the edit SQL functionality in the Custom SQL dialog. However, no Custom SQL dialog displays when this method executes.

Note: To use the Custom SQL feature, ensure that the query's data model has one table. If no table exists, "Script(x):uncaught exception:Invalid String" displays in the Console.

Syntax:

```
Expression.ResetCustomSQL()
```

Expression Required:

An expression that returns a query object

Example:

This example sets the FROM and WHERE clauses, processes a query, and restores the original SQL statement:

```
//Set the FROM clause, Set the WHERE clause, PROCESS, and then RESET SQL
ActiveDocument.Sections["Query"].CustomSQLFrom("FROM From.Sales_Fact, From.Periods,
From.Products") ActiveDocument.Sections["Query"].CustomSQLWhere("WHERE
(Periods.Day_Id=Sales_Fact .Day_Id AND Products.Product_Id=Sales_Fact.Product_Id) AND
(Periods.Quarter='Q1')")
ActiveDocument.Sections["Query"].Process()
ActiveDocument.Sections["Query"].ResetCustomSQL();
```

ResizeToBestFit (Method)

Applies To:

Column object

Description:

Resizes columns to ensure data is properly displayed.

Syntax:

```
Expression.ResizeToBestFit()
```

Expression Required:

An expression that returns a **Column** object

Example:

This example shows how to resize columns in a result set to best display data:

```
for (j =1; j < = ActiveDocument.Sections["Results"].Columns.Count; j++)
ActiveDocument.Sections["Results"].Columns[j].ResizeToBestFit()
```

RetrieveDimensions (Method)

Applies To:

OLAPCatalog object, OLAPCatalogNew (CubeQuery) object

Description:

Refreshes the dimension values in the Catalog pane. If a query is disconnected, this method attempts to connect silently to the database if the query is disconnected. The password used for the connection in this case must be set using SetPassword or a connection error will occur and an exception thrown. This method serves no useful purpose in the new OLAP section due to differences in how the new section handles the catalog.

Syntax:

```
Expression.RetrieveDimensions()
```

Expression Required:

An expression that returns a **Dimensions** object

Example:

This example shows how to change all columns in a result set to best fit the data:

```
ActiveDocument.Sections["OLAPQuery"].Catalog.Dimensions.RetrieveDimensions()
```

Save (Method)

Applies To:

Connection object, MetaDataConnection object, Document object, WebClientDocument object

Description:

Saves changes to documents or Interactive Reporting database connection files (.oce).

Note: Do not use ActiveDocument.Save()object in Interactive Reporting document file to be deployed in EPM Workspace.

Syntax:

```
Expression.Save()
```

Expression Required:

An expression that returns an object for these items:

- Connection
- Document
- WebClientDocument

Example:

This example shows how to create and save an Interactive Reporting document file:

```
var MyDocs = "c:\\Mydocs"
var MyName = "JavaScript Test"
var MyDoc = Documents.New(MyName)
MyDoc.Save()
```

SaveAs (Method)

Applies To:

Connection object, MetaDataConnection object, Document object, WebClientDocument object

Description:

Saves documents or Interactive Reporting connection files (.oce) under another name or location.

Note: Do not use ActiveDocument.SaveAs() in Interactive Reporting document files to be deployed in the EPM Workspace.

Syntax:

```
Expression.SaveAs(Filename As String)
```

Expression Required:

An expression that returns an object for these items:

- Connection
- Document

- WebClientDocument

Example:

This example shows how to save an Interactive Reporting document file using a different name:

```
var MyDocs = "c:\\Mydocs"
var MyName = "JavaScriptTest.bqy"
var MyFilename = MyDocs + "\\"+ MyName
ActiveDocument.SaveAs(MyFilename)
```

SaveToRepository (Method)

Applies to:

ActiveDocument object

Description:

Note: The SaveToRepository (Method) is only available for an Interactive Reporting document deployed in Interactive Reporting Web Client.

Uploads and saves repository objects (data models, standard queries, and standard queries with reports) in an Interactive Reporting document (BQY) for version-controlled distribution to networked Interactive Reporting users.

If you need to save the current Interactive Reporting document as another document, use the SaveToRepositoryAs (Method).

Syntax:

This method takes no arguments.

Example:

This example shows how to save the current Interactive Reporting document to the repository.

```
ActiveDocument.SaveToRepository()
```

SaveToRepositoryAs (Method)

Applies to:

ActiveDocument object

Description:

Note: The SaveToRepositoryAs(Method) is only available for an Interactive Reporting document deployed in Oracle's Hyperion® Interactive Reporting Web Client.

Uploads and saves as repository objects (data models, standard queries, and standard queries with reports) in an Interactive Reporting document (BQY) for version-controlled distribution to networked Interactive Reporting users.

Syntax:

```
Expression.SaveToRepositoryAs(String Filename, [optional] String Filedesc, [optional]
Boolean bOverride)
```

- **Filename**—(Required)

  Specify the file name consists of two parts: the Interactive Reporting document (BQY) path and name. The path is in the form of "/dir1/dir2/" (full path), where the leading "/" represents the Root directory or "dir1/dir2" (relative path) to the currently open Interactive Reporting document directory. The default path for the Interactive Reporting document path is the current directory. For example, you might specify: "/my directory/sub directory/doc.bqy", "doc.bqy" or "directory/doc.bqy".

- **String Filedesc**—(Optional) Specify a file description for the Interactive Reporting document to be saved to the repository. If the description exists, then it is added to the new document or replace sthe description of the existing document. The Boolean Override argument must be true to specify a file description.

- **Boolean Override**—(Optional) Specify if the Interactive Reporting document file name can be overridden. If the file name exists, the description is added to the new Interactive Reporting document or replaces the description of the existing Interactive Reporting document file name. This argument is only valid if the Boolean Override argument is true. If the override is false, no file description is applied.

Example:

This example shows how to save *mybqy.bqy* to the repository. Note that *myfolder* is in the current open directory.

```
ActiveDocument.SaveToRepositoryAs("myfolder/mybqy.bqy", "salesquery", true)
```

Select (Method)

Applies To:

ControlsDropDown object, ControlsListBox object

Description:

Changes the user selection of items in a control.

Syntax:

```
Expression.Select(Index As Long)
```

Expression Required:

An expression that returns an object for these items:

- ControlsDropDown
- ControlsListBox

Example:

This example shows how to set the selection of one drop-down list based on the index selected in another drop-down list:

```
var MyIndex = DropDown1.SelectedIndex=1
DropDown2.Select(MyIndex)
```

SendSQL (Method)

Applies To:

Application

Description:

Sends SQL strings to a data source. Data is not retrieved from the database.

This method does not send a SQL statement to the database session to which your query is connected.

Note: If your SendSQL string sends data modification commands, the database may require a *commit* statement. The commit behavior of the database may restrict which type of SQL string you can send. Because the SendSQL method requires an Interactive Reporting database connection file (OCE) as an argument, it does not apply to a script written for web client.

Syntax:

```
Expression.SendSQL(Ocename As String, Username As String, Password As String, SQLString
As String)
```

Expression Required:

An expression that returns an **Application** object.

Example:

This example shows how to send a SQL Statement to a database associated with an .Interactive Reporting database connection file (OCE):

```
var SQL = "insert into test (store_id, store) values (2, 'Computer City')"
var OCE = "c:\\OCEs\\Oracle.oce"
var user = "hyperion"
var pass = "hyperion"
Application.SendSQL(OCE,user,pass,SQL)
```

SetDrillThrough (Method)

Applies To:

OLAPQuery object

Description:

Enables you to define the mapping between multi-dimensional and relational databases. You must call the SetDrillThrough method for each map or unmap action.

Syntax:

Expression.SetDrillThrough(MapORUnMap as Boolean, String RelationalQueryName As String, DimensionOrRelationalTopicName as String, FactName as String)

- **MapORUnMap** As Boolean—Specify the boolean argument (true or false), to map or unmap dimensions to topic names. Use a boolean value of true, to map a dimension and a topic. Use a boolean value of false, to unmap a dimension. When maps are created, an internal link is made.
- **RelationalQueryName** As String—Specify the relational query name as a string from which you want to map or unmap a topic.
- **DimensionOrRelationalTopicName As String** —Specify the dimension or relational topic name as a string that you want to map or unmap. The topic name and dimensional level name must have identical names to set up a mapping or unmapping.
- **FactName As String** — Specify the topic name as a string used to map to the corresponding OLAP measure.

Expression Required:

An expression that returns a **Drill Through** object

Example:

This example shows how to use the SetDrillThrough method to map the *Store* dimension and topic:

```
ActiveDocument.Sections["OLAPQuery"].SetDrillThrough( true,"Query2","Store","Sales
Fact")
```

SetODSPassword (Method)

Applies to:

WebClientDocument object

Description:

Sets the OnDemand Server password and can automate OnDemand Server logon. This method is a web-enabled method and does not apply to the Oracle's Hyperion® Interactive Reporting Studio.

SetPassword (Method)

Applies To:

Connection object, MetaDataConnection object

Description:

Sets the password the Interactive Reporting connection file (OCE) uses to connecting to the database.

Syntax:

```
Expression.SetPassword(Password As String)
```

Note: Enclose the password in parentheses. If you do not, the string is created as a variable and you cannot unassign it.

Expression Required:

An expression that returns a **Connection** object

Example:

This example shows how to set the Password from a password field in a Dashboard section. The name of the password field is TextBox1:

```
var MyPass = TextBox1.Text
if (Application.Name != "BrioQuery")
ActiveDocument.Sections["Query"].DataModel.Connection.SetPassword(MyPass)
```

SetStoredProcParam (Method)

Applies To:

QuerySection object

Description:

Enables you to select stored procedures to obtain results.

The optional index parameter specifies the nth position in the stored procedure argument list (with the first parameter being indexed at 1). If no index value is provided, the assumed order is the order in which they are defined (again, beginning at 1). If there is a mix of some method calls with the index value and some without, the order are those with indexes first followed by definition order of those without indexes.

This method is used with the "ProcessStoredProc (Method)" on page 106.

Syntax:

```
Expression.SetStoredProcParm(Parameter As Value, [Optional]ParamIndex As Number)
```

Example 1:

This example shows how to open and process a stored procedure in the Query section:

```
ActiveDocument.Sections["Query"].SetStoredProcParam("Param1",1);
ActiveDocument.Sections["Query"].SetStoredProcParam("Param2",2);
ActiveDocument.Sections["Query"].ProcessStoredProc();
```

Example 2:

This example shows how to call a stored procedure. It assumes you have inserted a Query section and added the procedure.

```
ActiveDocument.Sections["Query"].ProcessStoredProc()
```

Example 3:

This example shows how to pass a single parameter from a Dashboards object. It proceeds the ProcessStoredProc method. The value of 1 in this example matches the position in the argument:

```
ActiveDocument.Sections["Query"].SetStoredProcParam(txtBeginDate.Text,1);
```

SetValue (Method)

Applies To:

ReferenceLine object

Description:

Sets the reference line value.

Syntax:

```
Expression.SetValue(Value as Number)
```

Example:

This example shows how to set the reference line value to 50:

```
ActiveDocument.Sections["Chart8"].ReferenceLines[1].SetValue('50')
```

Shell (Method)

Applies To:

Application object

Description:

Launches an external application and passes a command line argument to the application.

Use this method to open the browser from the Desktop client (e.g. from the Dashboard section). To open an URL from a web-based client (e.g. the web client), use the "OpenURL (Method)" on page 100.

Note: Do not use Application.Shell() in Interactive Reporting document files to be deployed in the EPM Workspace.

Syntax:

```
Expression.Shell(CommandLine As String, [optional]Arguments As String)
```

Expression Required:

An expression that returns an **Application** object

Example:

This example launches Notepad with a text file:

```
var App = "c:\\Winnt\\notepad.exe"
var Args = "C:\\Docs\\Readme.txt"
Application.Shell(App,Args)
```

ShowAll (Method)

Applies To:

AxisLabels collection

Description:

The ShowAll (Method) unhides all hidden axis labels.

Syntax:

```
Expression.ShowAll()
```

Expression Required:

An expression that returns an AxisLabels collection

Example:

This example unhides all the hidden Axis labels for the X and Z labels collections:

```
var MyChart = ActiveDocument.Sections["Chart"]
MyChart.XLabels.ShowAll()
MyChart.ZLabels.ShowAll()
```

ShowAsChart (Method)

Applies To:

(CubeQuery) QuerySection object

Description:

Creates a new Chart section to display the CubeQuery data as a chart. Calls DownloadToResults as part of its processing to create a table section as a parent for the Chart section. If a table or chart section already exists for the CubeQuery section, a new one is not created.

Syntax:

```
Expression.ShowAsChart()
```

Example:

This example show how to create a Chart section showing the data in the CubeQuery grid:

```
Sections["Query"].ShowAsChart();
```

ShowAsChart (Method)

Applies To:

OLAPQuery object

Description:

Charts the OLAPQuery data set, and creates an OLAPChart and OLAPResults section automatically.

Syntax:

```
Expression.ShowAsChart()
```

Example:

The following script was associated with a OnClick event. When the user clicks a command button, the OLAPQuery is charted and an OLAPResults and OLAPChart are created automatically:

```
ActiveDocument.Sections["OLAPQuery"].ShowAsChart()
```

SortByFact (Method)

Applies To:

PivotLabels collection, CategoryItems collection

Description:

Sets a data value (rather than label) criterion in the sort conditions available in the Pivot and Chart sections. This method corresponds to the Sort by Values feature in the Pivot and Chart sections where the second list selection orders each value of the target item specified in the first list selection by its corresponding numeric value in the second list.

Syntax:

```
Expression.SortByFact(FactName As String,SortFunction As BqSortFunction,
[optional]SortOrder As BqSortOrder)
```

Expression Required:

An expression that returns a **PivotLabelsTotals** or **CategoryItems** collection

Constants:

The BqSortFunction constant group contains these values:

- bqSortFunctionAverage
- bqSortFunctionCount
- bqSortFunctionMaximum
- bqSortFunctionMinimum
- bqSortFunctionNonNullAverage
- bqSortFunctionNonNullCount
- bqSortFunctionNullCount
- bqSortFunctionSum

The BqSortOrder constant group contains the bqSortAscend and bqSortDescend values.

Example:

This example shows how to sort the Product Name item by its corresponding numeric value *Amount Sales*:

```
ActiveDocument.Sections["Pivot2"].TopLabels["Product Name"].SortByFact("Amount Sales",
bqSortFunctionSum, bqSortAscend)
```

SortByLabel (Method)

Applies To:

PivotLabels collection, CategoryItems collection

Description:

Sets the primary sort criterion on an item by label or name rather than by reference to corresponding numeric data values. This method corresponds to the Sort by Labels feature in the Pivot and Chart sections

Syntax:

```
Expression.SortByLabel([SortOrder As BqSortOrder])
```

Expression Required:

An expression that returns a **PivotLabelsTotals** or **CategoryItems** collection.

Constants:

The BqSortOrder constant group contains the bqSortAscend and bqSortDescend values.

Example:

This example shows how to sort the top labels *Product Name* by region:

```
ActiveDocument.Sections["Pivot2"].TopLabels["Product Name"]. SortByLabel(bqSortAscend)
```

SortNow (Method)

Applies To:

SortItems collection

Description:

Sets the Sort Now feature on items placed on the Sort Line in the Results section. The Sort Now feature initiates the sorting function immediately on items on the Sort Line. This method is required to use the SortOrder (Property) (see the *Object Model Guide to Properties and Constants*).

Syntax:

```
Expression.SortNow()
```

Expression Required:

An expression that returns a **SortItems** collection

Example:

This example shows how to specify the SortNow method for items on the Sort Line in the Table section. In the example, the SortNow method is associated with a command button. When the command button is invoked, the text *Start SortNow* displays in the first text box. When the script within the try-catch block is executed, the text *End SortNow* displays in a second text box:

```
ActiveSection.Shapes["CommandButton1"].OnClick()
TextBox1.Text = "Start SortNow"
try
{
ActiveDocument.Sections["Table"].SortItems.SortNow()}
catch(e)
{
TextBox2.Text = e.toString()
}
TextBox1.Text ="End SortNow"
```

Spring (Method)

Applies To:

Field object, Table object, ReportPivot collection, ReportChart collection, Shapes collection

Description:

Enables you to maintain relative vertical spacing between dynamic objects. That is, you can spring one object to another so that if the first object is moved, increased or diminished, the second object moves in the same flow. To unspring an item, see the "UnSpring (Method)" on page 133.

Syntax:

```
Expression.Spring(Name as String)
```

Expression Required:

An expression that springs a report object.

Example:

This example shows how to spring the table and Smart Report objects:

```
ActiveDocument.Sections["Report"].Body.Tables["Table"].Spring("Chart")ActiveDocument.Sec
tions["Report"].Body.Tables["Table"].Spring("Pivot")
```

SyncWithDatabase (Method)

Applies To:

DataModel object

Description:

Causes a Data Model to synchronize itself with the underlying database tables.

Syntax:

```
Expression.SyncWithDatabase()
```

Expression Required:

An expression that returns a **Data Model** object

Example:

This example shows how to synchronize a Data Model with the database:

```
var MyDM = ActiveDocument.Sections["Datamodel"].DataModel
MyDM.SyncWithDatabase()
```

UnHide (Method)

Applies To:

OLAPMeasure object

Description:

Retrieves hidden measure items from the selected row or column to the OLAPQuery section.

Syntax:

```
Expression.UnHide()
```

Expression Required:

An expression that shows a hidden **OLAPMeasure object**

Example 1:

This example shows how to hide the fact *Amount Sales*.

```
ActiveDocument.Sections["Chart"].Facts["Amount Sales"].Hide()
```

Example 2:

This example shows how to hide the OLAP measure *Sales Average*:

```
ActiveDocument.Sections["OLAPQuery"].Measures["Sales Average"].UnHide()
```

UnhideAll (Method)

Applies To:

AxisLabels (XLabels, Ylabels, and ZLabels)

Description:

Restores all hidden label value item(s) that are hidden through the "HideSelection (Method)" on page 64 and "FocusSelection (Method)" on page 61.

Syntax:

```
Expression.XLabels.UnhideAll()
```

Expression Required:

An expression that unhides an **AxisLabels** item

Example:

This example shows how to unhide all label value items on the Xlabels:

```
ActiveDocument.Sections["AllChart"].XLabels.UnhideAll()
```

Unselect (Method)

Applies To:

ControlsListBox

Description:

Deselects objects in list boxes.

Syntax:

```
Expression.Unselect(Index As Number)
```

Note: Index is the nth item in the ListBox (index based 1).

Expression Required:

An expression that deselects a list box object

Dependency:

The MultiSelect (Property) must be enabled for the list box object in order to use this method.

Example:

In the following example, a list box was populated with four values, These values can be selected and counted in a text box. The Unselect method was added for each of the four values and any out of bound values:

```
//Selects all values in ListBox1 and performs a count
var cnt = ListBox1.Count
for (var i = 1; i <= cnt; i++)
{
ListBox1.Select(i)
}
TextBox1.Text=ListBox1.SelectedList.Count
//Unselects first index value in ListBox1
ListBox1.Unselect(1)
TextBox1.Text=ListBox1.SelectedList.Count
//Unselects second index value in ListBox1
ListBox1.Unselect(2)
TextBox1.Text=ListBox1.SelectedList.Count
//Unselects third index value in ListBox1
ListBox1.Unselect(3)
TextBox1.Text=ListBox1.SelectedList.Count
//Unselects fourth index value in ListBox1
ListBox1.Unselect(4)
TextBox1.Text=ListBox1.SelectedList.Count
```

UnSpring (Method)

Applies To:

Field object, Table object, ReportPivot collection, ReportChart collection, Shapes collection

Description:

Removes the relative vertical spacing between dynamic objects. That is, you can unspring one object from another so that if the first object was sprung (moved, increased or diminished), the second object moved in the same flow. To spring an item, see the "Spring (Method)" on page 130.

Syntax:

```
Expression.Unspring(Name as String)
```

Expression Required:

An expression that unsprings a report object

Example:

This example shows how to spring and unspring a table and chart object in the Body section:

```
ActiveDocument.Sections["Report"].Body.Tables["Table"].Spring("Chart")
ActiveDocument.Sections["Report"].Body.Charts["Chart"].UnSpring()
```

UseAlternateMetadataLocation (Method)

Applies To:

Connection object, MetaDataConnection object

Description:

Sets an alternate data source for retrieving metadata information.

Syntax:

```
Expression.UseAlternateMetadataLocation(Value As Boolean, [MetadataOce As String])
```

Expression Required:

An expression that returns a **Connection** object

Example:

This example shows how to change the metadata location for the current Data Model:

```
var MyDM = ActiveDocument.Sections["DataModel"].DataModel
var MyOCE = "c:\\OCEs\\MetaOracle.oce"
MyDM.Connection.UseAlternatieMetadataLocation(true,MyOCE)
```

Write (Method)

Applies To:

Console

Description:

Prints the output text specified by the OutputData parameter to the Console window.

Note: The Console.Write() object model syntax is not supported in an Interactive Reporting document to be deployed in EPM Workspace.

Syntax:

```
Expression.Write(OutputData As Value)
```

Expression Required:

An expression that returns a **Console** object

Example:

This example shows how to print the names of Interactive Reporting document file sections on a single line:

```
Console.Write(ActiveDocument.Name +"'s sections are: ")
for (j=1; j < ActiveDocument.Sections.Count; j++)
    Console.Write(ActiveDocument.Sections[j].Name + ", ")
```

WriteIn (Method)

Applies To:

Console

Description:

Prints output text specified by the OutputData parameter to the Console window and inserts a line after the inserted text.

Note: The Application.Alert() object model syntax is not shown in an Interactive Reporting document file deployed in the Oracle Enterprise Performance Management Workspace, Fusion Edition, but the text in the first argument is written to theInteractive Reporting Server .log file.

Syntax:

```
Expression.Writeln(OutputData As Value)
```

Expression Required:

Returns a **Console** object

Example:

This example shows how to print the names of Oracle's Hyperion® Interactive Reporting document file sections on individual lines:

```
Console.Writeln(ActiveDocument.Name +"'s sections are: ")
for (j=1; j < ActiveDocument.Sections.Count; j++)
Console.Writeln("Section #"+j +" = " +ActiveDocument.Sections[j].Name)
```

Index

U

W