## Contents

**Preface** .................................................................................................................. vii  
**Audience** .............................................................................................................. viii  
**Document Structure** ............................................................................................ viii  
**Related Documents** ............................................................................................... viii  
**Where to Find Documentation** ................................................................................ ix  
**Help Menu Commands** .......................................................................................... ix  
**Conventions** ........................................................................................................... x  
**Additional Support** ............................................................................................... xi  
  - Education Services ................................................................................................ xi  
  - Consulting Services ............................................................................................. xi  
  - Technical Support .............................................................................................. xii  
**Documentation Feedback** ..................................................................................... xii  

### CHAPTER 1 Methods and the Object Model .............................................................. 13

### CHAPTER 2 Methods ............................................................................................... 15  
  - Activate (Method) ............................................................................................... 16  
  - Add (Method) ..................................................................................................... 17  
  - AddAll (Method) ............................................................................................... 19  
  - AddAllTopics (Method) .................................................................................... 20  
  - AddComputed (Method) .................................................................................. 21  
  - AddComputedItem (Method) .......................................................................... 22  
  - AddDrillThroughValue (Method) .................................................................... 23  
  - AddExportSection (Method) .......................................................................... 24  
  - AddFilter (Method) .......................................................................................... 26  
  - AddFilterValue (Method) ................................................................................. 28  
  - AddTopic (Method) ........................................................................................... 30  
  - AddTotal (Method) ........................................................................................... 31  
  - AddTotals (Method) ......................................................................................... 32  
  - Alert (Method) .................................................................................................... 33
<table>
<thead>
<tr>
<th>Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>AliasTable (Method)</td>
<td>34</td>
</tr>
<tr>
<td>AuditSQL (Method)</td>
<td>35</td>
</tr>
<tr>
<td>AutoSizeHeight (Method)</td>
<td>36</td>
</tr>
<tr>
<td>AutoSizeWidth (Method)</td>
<td>37</td>
</tr>
<tr>
<td>Call (Method)</td>
<td>38</td>
</tr>
<tr>
<td>ChartThisPivot (Method)</td>
<td>39</td>
</tr>
<tr>
<td>Close (Method)</td>
<td>40</td>
</tr>
<tr>
<td>Connect (Method)</td>
<td>41</td>
</tr>
<tr>
<td>Copy (Method)</td>
<td>43</td>
</tr>
<tr>
<td>CreateConnection (Method)</td>
<td>44</td>
</tr>
<tr>
<td>CreateDateGroup (Method)</td>
<td>45</td>
</tr>
<tr>
<td>CreateLimit (Method)</td>
<td>46</td>
</tr>
<tr>
<td>CustomSQLFrom (Method)</td>
<td>47</td>
</tr>
<tr>
<td>CustomSQLWhere (Method)</td>
<td>48</td>
</tr>
<tr>
<td>Disconnect (Method)</td>
<td>49</td>
</tr>
<tr>
<td>DoEvents (Method)</td>
<td>50</td>
</tr>
<tr>
<td>DownloadToResults (Method)</td>
<td>51</td>
</tr>
<tr>
<td>DrillDown (Method)</td>
<td>52</td>
</tr>
<tr>
<td>DrillThrough (Method)</td>
<td>53</td>
</tr>
<tr>
<td>DrillUp (Method)</td>
<td>54</td>
</tr>
<tr>
<td>Duplicate (Method)</td>
<td>55</td>
</tr>
<tr>
<td>ExecuteBScript (Method)</td>
<td>56</td>
</tr>
<tr>
<td>Export (Method)</td>
<td>57</td>
</tr>
<tr>
<td>ExportToStream (Method)</td>
<td>59</td>
</tr>
<tr>
<td>FocusSelection (Method)</td>
<td>61</td>
</tr>
<tr>
<td>GetCell (Method)</td>
<td>62</td>
</tr>
<tr>
<td>Hide (Method)</td>
<td>63</td>
</tr>
<tr>
<td>HideSelection (Method)</td>
<td>64</td>
</tr>
<tr>
<td>ImportDataFile (Method)</td>
<td>65</td>
</tr>
<tr>
<td>ImportSQLFile (Method)</td>
<td>66</td>
</tr>
<tr>
<td>InterruptQueryProcess (Method)</td>
<td>67</td>
</tr>
<tr>
<td>Item (Method)</td>
<td>68</td>
</tr>
<tr>
<td>ItemIndex (Method)</td>
<td>70</td>
</tr>
<tr>
<td>Layer (Method)</td>
<td>71</td>
</tr>
<tr>
<td>LoadFromFile (Method)</td>
<td>72</td>
</tr>
<tr>
<td>LoadSharedLibrary (Method)</td>
<td>73</td>
</tr>
<tr>
<td>ModifyComputed (Method)</td>
<td>74</td>
</tr>
<tr>
<td>ModifyRepositoryFileAnalyzer (Method)</td>
<td>75</td>
</tr>
<tr>
<td>ModifyRepositoryFileOther (Method)</td>
<td>76</td>
</tr>
</tbody>
</table>
Preface

Welcome to the Hyperion System 9 BI+ Interactive Reporting Object Model and Dashboard Development Services Developer’s Guide, Volume III: Object Model Guide to Methods. This preface discusses the following topics:

- “Purpose” on page viii
- “Audience” on page viii
- “Document Structure” on page viii
- “Related Documents” on page viii
- “Where to Find Documentation” on page ix
- “Conventions” on page x
- “Additional Support” on page xi
- “Documentation Feedback” on page xii
Purpose


Audience

This book is written for developers and designers who create documents using Hyperion System 9 BI+ Interactive Reporting™ Studio™ and Interactive Reporting™ Web Client™ (those users with Data Model and Analyze, or Query and Analyze privileges) and who need to create front-ends using the Dashboard functionality.

Document Structure

Volume III: Object Model Guide to Methods is one of seven books that explain how to use Hyperion System 9 BI + Interactive Reporting (see "Related Documents" on page viii).

Related Documents

In addition to the Hyperion System 9 BI+ Interactive Reporting Object Model and Dashboard Development Services Developer’s Guide, Volume III: Object Model Guide to Methods, the following documentation is available:

Hyperion System 9 BI+ Interactive Reporting Studio User’s Guide provides an overview of Hyperion System 9 BI + Interactive Reporting and explains the user interface and basic commands. It includes how to retrieve data, how to query new data and change existing queries, and how to query a single database as well as multiple databases. It also covers how to work with query results.

Hyperion System 9 BI+ Interactive Reporting Object Model and Dashboard Development Services Developer’s Guide, Volume I: Dashboard Design Guide describes how to create custom applications in the Dashboard section, how to use JavaScript to script and to control Interactive Reporting documents (.bqys), how JavaScript programs are interpreted by the Interactive Reporting engine, how JavaScript programs are used to provide dynamic control of a document, and how JavaScript is used to respond to user events and the document lifecycles.


Dashboard Studio presents the wizard that works with a set of extensible and customizable templates to create interactive, analytical dashboards without the need to write programming logic.

Hyperion System 9 BI+ Interactive Reporting Object Model and Dashboard Development Services Developer’s Guide, Volume VI: Architect discusses the integrated development environment for Interactive Reporting Studio, that can be used to edit and debug JavaScript.

Where to Find Documentation

All Interactive Reporting documentation is accessible from the following locations:

- The HTML Information Map is available from the Interactive Reporting Help menu for all operating systems; for products installed on Microsoft Windows systems, it is also available from the Start menu.
- Online help is available from within Interactive Reporting After you log on to the product, you can access online help by clicking the Help button or selecting Help from the menu bar.
- The Hyperion Download Center can be accessed from the Hyperion Solutions Web site.

➤ To access documentation from the Hyperion Download Center:

1. Go to the Hyperion Solutions Web site and navigate to Services > WorldWide Support > Download Center.

   **Note:** Your Login ID for the Hyperion Download Center is your e-mail address. The Login ID and Password required for the Hyperion Download Center are different from the Login ID and Password required for Hyperion Support Online through Hyperion.com. If you are not sure whether you have a Hyperion Download Center account, follow the on-screen instructions.

   2. In the Login ID and Password text boxes, enter your e-mail address and password.

   3. In the Language list box, select the appropriate language and click Login.

   4. If you are a member on multiple Hyperion Solutions Download Center accounts, select the account that you want to use for the current session.

   5. To access documentation online, from the Product List, select the appropriate product and follow the on-screen instructions.

Help Menu Commands

Table i describes the commands that are available from the Help menu in Interactive Reporting.
### Table I  Help Menu Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Help on This Topic</td>
<td>Launches a help topic specific to the window or Web page.</td>
</tr>
<tr>
<td>Contents</td>
<td>Launches the Interactive Reporting help.</td>
</tr>
<tr>
<td>Information Map</td>
<td>Launches the Interactive Reporting Information Map, which provides the following assistance:</td>
</tr>
<tr>
<td></td>
<td>● Online help in PDF and HTML format</td>
</tr>
<tr>
<td></td>
<td>● Links to related resources to assist you in using Interactive Reporting</td>
</tr>
<tr>
<td>Technical Support</td>
<td>Launches the Hyperion Technical Support site, where you submit defects and contact Technical Support.</td>
</tr>
<tr>
<td>Hyperion Developer's</td>
<td>Launches the Hyperion Developer Network site, where you access information about known defects and best practices. This site also provides tools and information to assist you in getting started using Hyperion products:</td>
</tr>
<tr>
<td>Network</td>
<td>● Sample models</td>
</tr>
<tr>
<td></td>
<td>● A resource library containing FAQs, tips, and technical white papers</td>
</tr>
<tr>
<td></td>
<td>● Demos and Webcasts demonstrating how Hyperion products are used</td>
</tr>
<tr>
<td>Hyperion.com</td>
<td>Launches Hyperion’s corporate Web site, where you access a variety of information about Hyperion:</td>
</tr>
<tr>
<td></td>
<td>● Office locations</td>
</tr>
<tr>
<td></td>
<td>● The Hyperion Business Intelligence and Business Performance Management product suite</td>
</tr>
<tr>
<td></td>
<td>● Consulting and partner programs</td>
</tr>
<tr>
<td></td>
<td>● Customer and education services and technical support</td>
</tr>
<tr>
<td>About Interactive Reporting</td>
<td>Launches the About Interactive Reporting dialog box, which contains copyright and release information, along with version details.</td>
</tr>
</tbody>
</table>

### Conventions

The following table shows the conventions that are used in this document:

### Table II  Conventions Used in This Document

<table>
<thead>
<tr>
<th>Item</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arrows</td>
<td>Arrows indicate the beginning of procedures consisting of sequential steps or one-step procedures.</td>
</tr>
<tr>
<td>Brackets [ ]</td>
<td>In examples, brackets indicate that the enclosed elements are optional.</td>
</tr>
<tr>
<td>Bold</td>
<td>Bold in procedural steps highlights user interface elements on which the user must perform actions.</td>
</tr>
<tr>
<td>CAPITAL LETTERS</td>
<td>Capital letters denote commands and various IDs. (Example: CLEARBLOCK command)</td>
</tr>
</tbody>
</table>
### Table II  Conventions Used in This Document  (Continued)

<table>
<thead>
<tr>
<th>Item</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+O</td>
<td>Keystroke combinations shown with the plus sign (+) indicate that you should press the first key and hold it while you press the next key. Do not type the plus sign.</td>
</tr>
<tr>
<td>Ctrl+Q, Shift+Q</td>
<td>For consecutive keystroke combinations, a comma indicates that you press the combinations consecutively.</td>
</tr>
<tr>
<td>Example text</td>
<td>Courier font indicates that the example text is code or syntax.</td>
</tr>
<tr>
<td>Courier italics</td>
<td>Courier italic text indicates a variable field in command syntax. Substitute a value in place of the variable shown in Courier italics.</td>
</tr>
<tr>
<td>ARBORPATH</td>
<td>When you see the environment variable ARBORPATH in italics, substitute the value of ARBORPATH from your site.</td>
</tr>
<tr>
<td>$n, x$</td>
<td>Italic $n$ stands for a variable number; italic $x$ can stand for a variable number or a letter. These variables are sometimes found in formulas.</td>
</tr>
<tr>
<td>Ellipses (...)</td>
<td>Ellipsis points indicate that text has been omitted from an example.</td>
</tr>
<tr>
<td>Mouse orientation</td>
<td>This document provides examples and procedures using a right-handed mouse. If you use a left-handed mouse, adjust the procedures accordingly.</td>
</tr>
</tbody>
</table>
| Menu options          | Options in menus are shown in the following format. Substitute the appropriate option names in the placeholders, as indicated.  
  
  **Menu name > Menu command > Extended menu command**  
  
  For example: 1. Select **File > Desktop > Accounts**. |

### Additional Support

In addition to providing documentation and online help, Hyperion offers the following product information and support. For details on education, consulting, or support options, click the Services link at the Hyperion Solutions Web site.

### Education Services

Hyperion offers instructor-led training, custom training, and e-Learning covering all Hyperion applications and technologies. Training is geared to administrators, end users, and information systems professionals.

### Consulting Services

Experienced Hyperion consultants and partners implement software solutions tailored to clients’ particular reporting, analysis, modeling, and planning requirements. Hyperion also offers specialized consulting packages, technical assessments, and integration solutions.
Technical Support

Hyperion provides enhanced telephone and electronic-based support to clients to resolve product issues quickly and accurately. This support is available for all Hyperion products at no additional cost to clients with current maintenance agreements.

Documentation Feedback

Hyperion strives to provide complete and accurate documentation. Your opinion on the documentation is of value, so please send your comments by going to http://www.hyperion.com/services/support_programs/doc_survey/index.cfm.
Welcome to *Volume III: Object Model Guide to Methods*. This guide is designed to help you learn about the methods (actions) associated with the Interactive Reporting Object Model (a hierarchical representation of Interactive Reporting).

In most cases, a method is an action that returns a value and may need an argument. For example, methods can be used to create, activate, open, close, save, add, copy, remove, process, export, recalculate, and so on. Whereas a property is associated with what an object is, a method is associated with what an object does.

The *Volume III: Object Model Guide to Methods* is filled with real examples and we encourage you to copy code from it. It will help you create scripts for your dashboard section that are important to your organization.
A function associated with an object is called a *method*. The methods for an object represent the actions that a script can request from that element.

For example, the document section object has a method called `Activate()` which can be used to activate the section. This method corresponds to the user clicking on the section in the Section/Catalog pane. The method performs all the background operations needed to hide the current section and causes the selected section to display and initialize itself appropriately.

This chapter provides an alphabetical reference to the methods available for the Interactive Reporting Object Model a more detailed discussion of the Object Model, see *Volume 1: Dashboard Design Guide*. 
**Activate (Method)**

**Applies To:** ChartSection object, DataModelSection object, Document object, DashboardSection object, OLAPQuerySection object, PivotSection object, QuerySection object, ReportSection object, ResultsSection object, Sections collection, TableSection object, WebClientDocument object

**Description:** The Activate (Method) is used to switch the focus of a document or section.

**Syntax:**

```
Expression.Activate()
```

**Expression Required:** An expression that returns an object for any of the following sections:

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

**Example:** This example shows how to unhide and activate a section:

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

Applies To: AggregateLimits collection, AppendQueries collection, CategoryItems, ChartSection, Columns collection, ControlsDropDown object, ControlsListBox object, Documents collection, Joins collection, Limits collection, LimitValues collection, LocalJoins collection, LocalResults collection, OLAPLabels collection, OLAPMeasures collection, OLAPSlicers collection, Parentheses collection, PivotLabel object, PivotLabelTotals collection, Requests collection, Sections collection, SortItems collection, TopLabels collection, Topics collection

Description: The Add (Method) is a common method for most collections. It adds an object to a collection and returns a reference to the newly added object.

Note: The Session.Form.Add(), Session.URL.Add() and Session.Cookies.Add() object model syntax is not supported in a Hyperion System 9 BI + Workspace document.

Note: The Add() method works differently for LimitValues (Collection), AvailableValues (Collection), CustomValues (Collection), and SelectedValues (Collection) (see the Object Model Guide to Objects and Collections). For the AvailableValues (Collection), the Add (Method) does nothing, because the values are obtained from the database. For the CustomValues collection, the Add (Method) adds value to the list. For the SelectedValues (Collection), the Add (Method) adds a value to the selected list.

Syntax: Expression.Add(ItemName As String)

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

- CategoryItems
- ChartSection
- Columns
- ControlsDropDown
- ControlsListBox
- Documents
- Joins
- Limits
- LimitValues
- LocalJoins
- LocalResults
- OLAPLabels
- OLAPMeasures
- OLAPSlicers
Example 1: This example shows how to create a new limit, add values to the limit, and then add the limit to the limit line:

```javascript
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:

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

Example 3: This example shows how to add two new topics to a Data Model and add a join between the topics:

```javascript
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 shows how to add a Pivot section type to the Results section:

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

Note: A Chart, Pivot, and Table section type must be associated with a parent section, such as Results. A Query, Dashboard, or Report section type does not have to be associated with a parent section.
AddAll (Method)

**Applies To:** SelectedValues collection (instantiate from the LimitValues Collection)

**Description:** The AddAll() method of the SelectedValues (Collection) enables you to select all values from either the AvailableValues (Collection), or CustomValues (Collection) depending on what is selected. Use this method in conjunction with the LimitValueType (Property) so that you can determine in advance which limit value set is selected. The value associated with this property is a member of the constant group called BqLimitValueType. Two possible values of BqLimitValueType are: bqLimitValueTypeAvailable and bqLimitValueTypeCustom.

**Note:** You can select a single value at a time using the “Add (Method)” on page 17 of the SelectedValues (Collection); however, you must know all the values in advance. 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 and added to the limit line in the Query section. Then, all available values in the Limit dialog box are added:

```csharp
//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 chose to define programmatically your own join paths, use the AddAllTopics() method of the DefinedJoinPaths (Collection) to select all topics based on the items on the Request and Limit lines. This method corresponds to selecting all available topics from the Define Join Path dialog box and populating the Topics in Join Path list:


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

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

*Applies To:* Columns collection

*Description:* The AddComputed (Method) creates a new 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 shows how to create a computed column that concatenates the string “Manager =” 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 new item. This method enables you to specify the name, expression, and index for the computed item.

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

The name is the name of the computed item and 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 Interactive Reporting expression that displays in the Computed Items dialog box.

The optional index determines the computed item’s position in a particular pane. For example, an index of two places is as the second item in the Y-Fact pane.

When you apply the AddComputedItem method to a Query Request object, you can use the BqDataType constant to confirm or change an item’s data type. This enables you to preserve the precision of a mixed-data type computations, or to change the way a data item is handled (for example, interpreting number as strings).

Attention to data types is most important you are when computing items in the Query section. Here the computation is performed on the database server, and the computed item 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:

```csharp
ActiveDocument.Sections["Chart"].Facts.AddComputedItem (‘Double_Sales’, ‘Unit_Sales *2’,2)
```
AddDrillThroughValue (Method)

 Applies To:  OLAPQuery object

 Description:  Adds a label names as a drill-through value.

 Syntax:  Expression.AddDrillThroughValue(string LabelName)

 Expression Required:  An expression that returns an add drill-through label value

 Example:  The following 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 format, making it easy to distribute data to many users through corporate intranets or Web sites. Using this scripting method executes a high-fidelity series of XHTML pages that match the original Interactive Reporting reports as closely as HTML can; creates a set of .htm, .css and .gif files; and if charts or Dashboard sections are included in the export set, creates.jpg files. The resulting file set is a frame-based HTML display that includes a report navigation frame, a report display area, and hyperlinks to move among the multiple pages of a specific report.

When exporting selected sections, specify the section name in the AddExportSection (Method). A single call to AddExportSection Method) must be specified for each section to be exported. After specifying all sections to be exported, the Document level “Export (Method)” on page 57 is called. This method enables you to specify the export file format.

Regardless of the order of the AddExportSection (Method) calls, the exported document preserves the original fixed section ordering of an Interactive Reporting (.bqy) document, minus sections not selected for export. Invalid AddExportSection (Method) calls, either as a result of invalid section type or an invalid section name, are ignored.

When sections are exported successfully, the Export (Method) clears the export buffer. If sections are not exported successfully, use the “RemoveExportSection (Method)” on page 123 to flush the export buffer of sections. That is, all sections set for export are cleared from the export buffer. For instance, if you specify a Report, Pivot, and Chart section to be exported by way of the AddExportSection (Method), a call to the RemoveExportSections (Method) nullifies the section set up for export. Consequently, a call to the Export (Method) assumes that you do not want to select individual sections for export but instead prefer that all sections be exported.

The exported document resides in the default export directory wherever the briopqy.exe file is located. The export directory can be modified by explicitly specifying a path for the filename argument in the Export() method. For example, c:\temp\myfile.htm and myfile.htm are valid arguments for filename. The .htm extension is used to denote the HTML file type. A .htm extension is used even if .htm is specified as in the following example:

```vbscript
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 an object for any of the following items:
- ChartSection
- PivotSection
- TableSection
- Section

**Example 1:** This example shows how to export selected sections of an Interactive Reporting (.bqy) document:

```csharp
//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 and then later cleared from the export buffer. The Export method in the last part of the script enables all sections in the document to be exported:

```csharp
//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) supports all operator types for side labels and measures for all supported OLAP engines.

The “AddFilterValue (Method)” on page 28 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 the following values:

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

The BqOperator constant group consists of the following 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 has not been set as a variable:

```csharp
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.

```csharp
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 26 supports all operator types for side labels and measures for all supported OLAP engines.

Note: If you are using this method to apply a filter to a measure value, this method can be used only against an Essbase database. In addition, 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 the following 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. In this example, 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 includes the filter value within brackets so that a URL address can be used.

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

*Applies To:* DefinedJoinPath collection

*Description:* If you chose to 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:
```
```
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:** The AddTotals (Method) 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: The Alert (Method) displays a simple dialog box. Up to three buttons can be displayed on the dialog box with custom names. When the user selects a button, an integer is returned that corresponds to the number of the button. If the user selects button #1, the number 1 is returned, and so on.

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 the user’s response:

```javascript
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 specify an alias to assign user-friendly names to database physical member and/or generation/level names. Essbase stores the aliases in an Alias Table in the cube. Since a cube can have multiple alias tables, you can select the alias table to use and modify the query based on the value you enter.

The AliasTable (Method) prompts the user to process the query. The PromptOption argument determines if a dialog appears so that the user can specify a value of either 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)
```
AuditSQL (Method)

Applies To: Query object, DataModel object

Description: The AuditSQL (Method) enables you to define a SQL Statement that is executed when the audit event is triggered. That is, you record how Interactive Reporting, a database server, or network resources are being used. When triggered, the SQL statements update an audit log table, which the administrator can query independently to track and analyze usage 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 the following 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 is executed:
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 is retrieved to the client workstation:
ActiveDocument.Sections["Query"].DataModel.AuditSQL(bqAuditPostProcess,"Select username from all_users")
AutoSizeHeight (Method)

**Applies To:** PivotLabel collection, PivotFact object, OLAPLabel object, OLAPMeasure object

**Description:** By default, Interactive Reporting truncates Pivot fact columns evenly and without regard to the length or height of data values. Numeric data that does not fit within the height or length of the cell is replaced with pound signs (#). To size the height of a Pivot fact column automatically so that all values are displayed within the column, use the AutoSizeHeight method.

**Syntax:**

```csharp
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:

```csharp
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

**Description:** By default, Interactive Reporting truncates Pivot fact columns evenly and without regard to the length or height of data values. Numeric data that does not fit within the height or length of the cell is replaced with pound signs (#). To size the width of a Pivot fact column automatically so that all values are displayed within the column, use the AutoSizeWidth method.

**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 the width of the Unit Sales fact column:

```javascript
ActiveDocument.Sections["Pivot"].Facts["Unit Sales"].AutoSizeWidth()
ActiveDocument.Sections["Pivot"].Facts["Unit Sales"].AutoSizeHeight()
```
**Call (Method)**

**Applies To:** SharedLibrary object

**Description:** Use the Call (Method) to invoke 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:** The ChartThisPivot (Method) creates a new Chart section using the 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 then change the display characteristics of the chart:
```
MyChart = ActiveDocument.Sections("Pivot").ChartThisPivot()
MyChart.Title = "Chart Created from Pivot"
```
Close (Method)

**Applies To:** Document object,

**Description:** Closes the document. This method is equivalent to selecting File > Close.

**Note:** The Document.Close() object model syntax is not supported in a Hyperion System 9 BI + Workspace document or in the 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:

```plaintext
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 a 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 prior to 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 OCE and document content. For example, if the OCE associated with the query is set to prompt the user when the document 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 (.bqy) document file, and when a Interactive Reporting (.bqy) 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 41.

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 49 to disconnect the current connection, and connect to another database. The Disconnect (Method) is available for Designer only:

```csharp
// < 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.oe")
    MyConnection.Connect(true)
}
```
Copy (Method)

**Applies To:** ChartSection object, PivotSection object, Section object, TableSection object, OLAPQuerySection object

**Description:** The Copy (Method) makes a copy of the section and puts it on the clipboard.

**Syntax:** `Expression.Copy()`

**Expression Required:** An expression that returns an object for any of the following sections:

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

The Copy (Method) is not available for the following objects:

- DashboardSection
- QuerySection
- ReportSection
- DataModelSection

The Copy (Method) does not work in a document to be deployed in the Workspace.

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

```csharp
ActiveDocument.Sections["Results"].Copy()
```
CreateConnection (Method)

Applies To: Application

Description: Creates a stand-alone connection object. Use this method to create .OCE files, which are not automatically associated with a Data Model. CreateConnection() returns a connection object. Refer to the Connection (Object) for a complete list of its methods and properties.

Note: The Application.CreateConnection() object model syntax is not supported in a Hyperion System 9 BI + document.

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 .OCE 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:

```csharp
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:** The CreateLimit (Method) creates a new, 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 is different for regular limits, computed item limits, and aggregate limits. For regular limits the argument is a reference to the table topic and the topic item; for example, CreateLimit("Sales_Facts.Amount_Sales"). For both computed item limits and aggregate limits the argument is a reference to the item’s Display Name on the request line, for example, 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:

```vbnet
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:

```vbnet
ActiveDocument.Sections["Query"].ResetCustomSQL();
```
CustomSQLFrom (Method)

**Applies To:** QuerySection object

**Description:**
Sets the FROM clause of a SQL statement prior to 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 127.

CustomSQLFrom (Method), “CustomSQLWhere (Method)” on page 48, and “ResetCustomerSQL (Method)” on page 127 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:

```csharp
//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 prior to 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 47, and the “ResetCustomerSQL (Method)” on page 127 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:

```csharp
```
Disconnect (Method)

Applies To: Connection object, metaDataConnection object

Description: The Disconnect (Method) drops the connection between the connection 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:** The DoEvents() method 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 a Hyperion System 9 BI + Workspace document.

**Syntax:** Application.DoEvents()

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

```javascript
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)
  DoEvents();
}
```
DownloadToResults (Method)

Applies To: OLAPQuery object

Description: The DownloadToResults (Method) enables you to download an OLAPQuery data set to an OLAPResults section within the document. When an OLAPResults section has been 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 > Tools > Options > Program Options > OLAP tab > 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 has been 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 24.

**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:

```csharp
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 both relational and OLAP queries has been set up correctly. This method uses the BqDrillThroughPrompt constant group.

**Note:** The dialog prompt for DrillThrough() is not a requirement in the Hyperion System 9 BI + Workspace. In the Hyperion System 9 BI + 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:**
```
```

Also note the following three 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 the following 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 24.

**Note:** DrillDown and DrillUp Measures are specific to 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, DataModelSection object, DashboardSection object, OLAPQuerySection object, PivotSection object, QuerySection object, ResultsSection object, TableSection object, ReportSection object

**Description:** Creates an exact copy of a section.

**Syntax:**

```
Expression.Duplicate()
```

**Expression Required:** An expression that returns an object for any of the following: sections:

- ChartSection
- DataModelSection
- DashboardSection
- PivotSection
- ReportSection

**Example:**

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

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

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, DataModelSection object, Document object, DashboardSection object, OLAPQuerySection object, PivotSection object, QuerySection object, ResultsSection object, Section object, TableSection object, ReportSection object

**Description:** Creates a new 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 Studio and Interactive Reporting Web Client reference the umask command of the user running the BIService 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.

**Syntax:** 

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

**Expression Required:** An expression that returns an object for any of the following sections:

- ChartSection
- DataModelSection
- DashboardSection
- OLAPQuerySection
- PivotSection
- QuerySection
- Section
- TableSection

**Constants:** The BqExportFileFormat constant group consists of the following values:

- bqExportFileFormatCSV
- bqExportFileFormatExcel2
- bqExportFileFormatExcel5
- bqExportFileFormatHTML
- bqExportFileFormatJPEG
- bqExportFileFormatLotus123
- bqExportFormatPDF
- bqExportFileFormatText
Methods

- bqExportFormatOfficeHTML
- bqExportFormatOfficeMHTML

For information on exporting a Interactive Reporting (.bqy) 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 a Interactive Reporting (.bqy) document file to the HTML format dynamically (for use with the Hyperion System 9 BI + Workspace), see HTMLPregenerateMode (Property), HTMLPregenerateFirstLastNPages (Property), DHTMLExportBreakRowCount (Property), and DHTMLVerticalPageBreakUnits (Property) in the 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:

```javascript
//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, DataModelSection object, Document object, DashboardSection object, OLAPQuerySection object, PivotSection object, QuerySection object, ResultsSection object, Section object, TableSection object, ReportSection object

**Description:** The ExportToStream (Method) enables you to use data streaming in Hyperion System 9 BI + 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 is called, this information is ignored when streaming is enabled. In the case of the full clients (Designer/Explorer), 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 BIService 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.

**Syntax:**

```
Expression.[optional]String Filename,
[optional]BqExportFileFormat FileFormat,
[optional]Boolean IncludeHeaders,
[optional]Boolean DataStreaming,
[optional]Boolean Prompt )
```

**Expression Required:** An expression that returns an object for any of the following sections:

- ChartSection
- DataModelSection
- DashboardSection
- OLAPQuerySection
- PivotSection
- QuerySection
- Section
- TableSection
**Constants:** The ExportToStream (Method) uses the BqExportFileFormat constant group, which consists of the following values:

- bqExportFileFormatCSV
- bqExportFileFormatExcel2
- bqExportFileFormatExcel5
- bqExportFileFormatHTML
- bqExportFileFormatJPEG
- bqExportFileFormatLotus123
- bqExportFormatPDF
- bqExportFileFormatText
- bqExportFormatOfficeHTML
- bqExportFormatOfficeMHTML

**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)
```
FocusSelection (Method)

Applies To: AxisLabels (XLabels, YLabels, and ZLabels)

Description: The FocusSelection (Method) 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:

```javascript
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:** The GetCell (Method) returns the value of an individual 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 from the values in a Results section:

```javascript
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)
}
```
Hide (Method)

Applies To: Chart Fact object

Description: The Hide (Method) enables you to hide a Chart fact object. When this script is 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

Example1: This example shows how to hide the fact Amount Sales:
ActiveDocument.Sections["Chart"].Facts["Amount Sales"].Hide()
HideSelection (Method)

*Applies To:* AxisLabels (XLabels, YLabels and ZLabels)

*Description:* The HideSelection (Method) enables you to hide selected label value items. Use this method to concentrate your view on selected item(s) of interest. If you need to hide a Chart YFact object, use the YFact instead of the YLabels object. It is easy to understand that XLabels and ZLabels are labels on the X and Z axis (i.e. Q1, Q2...). However the behavior of YLabels is more difficult. You could think of YLabels as the layers in the Y direction. For example, a stack bar could have 3 layers in the Y direction but there are only two items in the YFact outline.

*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 and then hide them in the Chart:

```javascript
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)

Appplies To: Sections collection, WebClientDocument object

Description: Imports a data file into a Query section.

Syntax: 
Expression.Import(Filename As String, FileType As BqImportDataFileFormat)

Expression Required: An expression that returns a Sections object

Constants: The BqImportDataFileFormat constant group contains the following values:
- bqImportFormatCommaText
- bqImportFormatExcel
- bqImportFormatTabText

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 into an existing query, and retrieves the data set from the database server. When you import a file, it is scanned to determine the number of columns that will be returned by the SQL, with the request line becoming populated with a column indicator for each of the columns. Using this feature, you can take advantage of SQL statements that you have already written.

Before using this method, be sure that you are connected to a database server. The Query section to which you are importing the SQL must have no tables. In addition, the SQL file to be imported must begin with a SELECT statement and you should know the number of columns to be displayed in the Results section. When the SQL file has been imported into the query, you can drag items from the table to the Request line, use the Custom SQL feature, or display its properties. The imported SQL file cannot be edited, but you can specify a user-friendly name for the Request line item and identify its data type.

**Syntax:** Expression.ImportSQLFile(Filename As String,numColumns As Number)

**Expression Required:** An expression that returns a Query object

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

```javascript
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:* The InterruptQueryProcess (Method) is an Interactive Reporting document level function. This method stops the processing sequence and should be used only with the “OnPreProcess (Method)” on page 100 event. This method takes no arguments.

*Syntax:* `Expression_INTERRUPTQUERYPROCESS()`

*Expression Required:* Interactive Reporting document

*Example:* This example displays the InterruptQueryProcess method for an active document:

```
ActiveDocument_INTERRUPTQUERYPROCESS()
```
Item (Method)

Applies To: AggregateLimits collection, AppendQueries collection, Columns collection, Controls collection, ControlsDropDown object, ControlsListBox object, DMCatalogItems collection, DMResults collection, Documents collection, Joins collection, Limits collection, LimitValues collection, ListSelection object, Parentheses collection, PivotLabel object, PivotLabelTotals collection, Values collection, RecentFiles collection, Requests collection, Results collection, Sections collection, SortItems collection, Shapes collection, Toolbars collection, TopicItems collection, TopLabels collection, Topics 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 aHyperion System 9 BI + Workspace document.

Syntax: Expression.Item(NameOrIndex) As Object

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

- Column
- Control
- ControlsDropDown
- ControlsListBox
- DerivableQueries
- DMCatalogItem
- DMResults
- Document
- Join
- LabelValues
- Limit
- LimitValues
- ListSelection
- LocalJoins
- LocalResults
- OLAPLabel
- OLAPMeasure
- OLAPSlicer
This example shows how to return the third section named *Query* in the current document:

```javascript
var MySection = ActiveDocument.Sections.Item(3)
```

or

```javascript
var MySection = ActiveDocument.Sections[3]
```

or

```javascript
var MySection = ActiveDocument.Sections.Item("Query")
```

or

```javascript
var MySection = ActiveDocument.Sections["Query"]
```
ItemIndex (Method)

 Applies To:  List box object

 Description:  The ItemIndex (Method) 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:

 ListingBox1.SelectedList.ItemIndex()
Layer (Method)

Applies To: Field object, Table object, ReportPivot collection, ReportChart collection, Shapes collection

Description: The Layer (Method) 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 brings the triangle. 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 the following 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:** The LoadSharedLibrary (Method) 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 a Hyperion System 9 BI + Workspace document.

**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 Kernel32.dll for 4 seconds with 5000Hz:

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

```vbnet
'//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 Analyzer 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 a document 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 Analyzer document.

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

**Applies To:** EmbeddedBrowser object, HyperLink object

**Description:** Enables you to modify a Interactive Reporting (.bqy) document 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 a document 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 77.
**ModifyRepositoryFileBQY (Method)**

**Applies To:** EmbeddedBrowser object, HyperLink object

**Description:** The ModifyRepositoryFileBQY enables you to modify a Interactive Reporting (.bqy) document repository object for an embedded browser object or hyperlink object. When this method is 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 a document name argument for this method.

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

**Expression Required:**
- **DocumentName** – Required; String; Maps to the name of the object selected from the repository.
- **BQYSectionName** – Optional; String; Maps to the BQY 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 a BQY document repository object (corresponds to the Toolbar field on the Document Options dialog). Uses the bqRepositoryToolbarType (Property).
- **SmartcutParams** – Optional; String; Maps to the Other Parameters control used for any additional parameters appended to the URL.

**Example:**
The following example shows how to change a repository document, 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 Hyperion System 9 BI + Workspace:

```csharp
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.WriteLine("Caught 1a: " + e.ToString())
}
}

else
{
    try
    {
        var sDocumentName = TextBox1.Text
    }
    catch(e)
    {
        Console.WriteLine("Caught 1b: " + e.ToString())
    }

    //Set section name argument value
    Console.WriteLine("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.WriteLine("Caught 2a: " + e.ToString())
    }

    }
    else
    {
        try
        {
            var sBQYSectionName = TextBox2.Text
        }
        catch(e)
        {
            Console.WriteLine("Caught 2b: " + e.ToString())
        }
    }

    //Set Toolbar argument value
    Console.WriteLine("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.WriteLine("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.WriteLine("Caught 3b: " + e.ToString())
    }
}

// Set parameter argument value
Console.WriteLine("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.WriteLine("Caught 4a: " + e.ToString())
    }
}
else
{
    try
    {
        var sDocParams = TextBox5.Text
    }
catch(e)
    {
        Console.WriteLine("Caught 4b: " + e.ToString())
    }
}

// Use the argument values in the method
Console.WriteLine("Step 5")

try
{
    ActiveSection.Shapes["EmbeddedBrowser1"].ModifyRepositoryFileBQY(sDocumentName, sBQYSectionName, cToolbarType, sDocParams) 
}
catch(e)
{ }  
Console.WriteLine("Caught 5: " + e.ToString())
}
}

Console.WriteLine("End " + ActionName)
ModifyRepositoryFileBQYJob (Method)

Applies To:  EmbeddedBrowser object, HyperLink object

Description:  The ModifyRepositoryFileBQY (Method) enables you to modify a BQY 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 a document 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 (.bqy) document 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 (.bqy) document repository object (corresponds to the Toolbar field on the Document Options dialog).

Smartcut Parameters – 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 77.
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 a document 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 a 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 77.
ModifyRepositoryFileReports (Method)

 Applies To: EmbeddedBrowser object, HyperLink object

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

 ModifyRepositoryFileReports(String ServerNameAndPortNumber, String FolderAndPathAndDocumentName, bqReportsType ReportsType, [Optional] bqExportFileFormat RepositoryReportsDisplayFormat, [Optional] bqRepositoryToolbarType ReportsToolbarType, [Optional] String RepositorySmartcutParams)

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

 Expression Required: Parameters:
 ServerNameAndPortNumberName – Required; String; Maps to the server name and port number of the Smartcut for the document.
 FolderAndPathAndDocumentName – Required; String; Maps to the folder, path and document name of the Smartcut for the document.
 bqReportsType – Returns the Report document type using the BqReportsType constants group. Valid values for this constants are:
   ● bqReportsTypeBookReport
   ● bqReportsTypeReport
   ● bqReportsTypeSnapshotBookReport
   ● bqReportsTypeSnapshotReport

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

 RepositoryReportsToolbar 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) do not display for Reports populating an Interactive Reporting (.bqy) document external content control. The third Reports toolbar, Point of View (POV), is made an option for display within the Dashboard. The default setting is the third POV toolbar not to be shown.
The HTML/PDF radio buttons displayed in Reports behave in accordance to the third POV toolbar setting; if the POV toolbar is shown, the HTML/PDF radio buttons are shown, and if POV toolbar is hidden, the HTML/PDF radio buttons are hidden.

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

Move (Method)

**Applies To:** GroupItems object, ReportGroup object, TableFacts object

**Description:** Moves an object in the ReportGroup collection. For example, you can use this method to reverse the order of two items in the Facts pane of the Table Outliner.

**Syntax:**

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

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

- Group Items object
- ReportGroup object
- TableFacts object

**Example:**

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

```csharp
//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.WriteLine(e.ToString())
}
```
New (Method)

**Applies To:** Documents collection

**Description:** The New (Method) creates a new blank Interactive Reporting document.

**Syntax:**

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

**Expression Required:** An expression that returns a `Documents` object

**Example:**

This example shows how to create a new Interactive Reporting document:

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

Applies To: ResultsSection object, TableSection object

Description: The OfficeHTMLFormulasEnabled Property enables generation of Microsoft Excel formulas when exporting Results and Tables sections to Microsoft Office HTML and MHTML formats. Adding formulas allows for automatical recalculation of Computed Items and Break/Grand Total on Excel's worksheet.

Action: Read-write, Boolean

Example: This example shows how to disable the formula generation:

```plaintext
Documents["Sales.bqy"].Sections["Results"].OfficeHTMLFormulasEnabled = false
```
OnActivate (Method)

Applies To: Dashboard Section

Description: The OnActivate() method is an Interactive Reporting section level function. This method is available regardless of the state of the application and can be accessed 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 Button
- GraphicsLine
- 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

**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 document 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 Hyperion System 9 BI + 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:

```javascript
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

*Description:* The OnChange() method is an Interactive Reporting Dashboard Object level function. This method is only available when a Dashboard section is included in the Interactive Reporting document, and the Dashboard section contains a text box. 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 Hyperion System 9 BI+ Workspace document.

*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, Shape

**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.

**Syntax:**
```
Expression.OnClick()
```

**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

Description: Launches a client-side JavaScript OnClientClick event when a user clicks on a control or enters text in a text box in the Hyperion System 9 BI + 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

Description: Launches a client-side JavaScript OnClientEnter event when a user enters a text box in the Hyperion System 9 BI + 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

 Description: Launches a client-side JavaScript OnClientExit event when a user exits a text box in the Hyperion System 9 BI + 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

Description: The OnDeactivate() method is an Interactive Reporting Dashboard section level event. This method is available regardless of the state of the application and can be accessed through scripting. The OnDeactivate() method will execute 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

**Description:** The OnDoubleClick() method is an Interactive Reporting Dashboard object level function. This method is only available when a Dashboard section is included in an Interactive Reporting document and the Dashboard section contains a list box. 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

**Description:** The OnEnter() method is an Interactive Reporting Dashboard object level function. This method is triggered when a text box object gets the focus. This method is only available when a Dashboard section is included in the Interactive Reporting document and the Dashboard section contains a text box. The event is triggered when the text box object gets the focus.

**Note:** The ActiveDocument.Sections["Dashboard"].Shapes["TextBox1"]. OnEnter() object model syntax is not supported in an Hyperion System 9 BI + Workspace document.

**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

Description: The OnExit() method is an Interactive Reporting Dashboard object level function. This method is triggered when a text box object has lost the focus. This method is only available when a Dashboard section is included in the Interactive Reporting document and the Dashboard section contains a text box.

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

**Description:** The OnPostProcess() method is an Interactive Reporting document 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 OnPostProcess method executes a script stored under the OnPostProcess event trigger. This method takes no arguments.

**Note:** Calling the “Process (Method)” on page 110 from the “OnPreProcess (Method)” on page 100 or OnPostProcess (Method) events can result in an infinite loop.

**Syntax:** Expression.OnPostProcess()

**Expression Required:** An expression that returns an Interactive Reporting document object

**Example:** This example displays the OnPostProcess method for the active document:

```javascript
ActiveDocument.OnPostProcess()
```
OnPreProcess (Method)

**Applies To:**  Document object

**Description:**  The OnPreProcess() method is an Interactive Reporting document level function. The OnPreProcess method will execute a script stored under the OnPreProcess event trigger. The method takes no arguments.

**Note:**  Calling the “Process (Method)” on page 110 from the OnPreProcess (Method) or “OnPostProcess (Method)” on page 99 events can result in an infinite loop.

**Syntax:**  Expression.OnPreProcess()

**Expression Required:**  An expression that returns an Interactive Reporting document object

**Example:**  This example displays the OnPreProcess method for the active document:

```
ActiveDocument.OnPreProcess()
```
OnRowDoubleClick (Method)

Applies To: Dashboard Section

Description: The OnRowDoubleClick() method is an Interactive Reporting Dashboard object level function. This method 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

 Description: The OnSelection() method is an Interactive Reporting Dashboard object level function. This method is triggered after a selection has been made from a drop-down box. This method is only available when a Dashboard section is included in the Interactive Reporting (.bqy) document 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

** Description: ** The OnShutdown() method is an Interactive Reporting document 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 OnShutdown method executes a script stored under the OnShutdown event trigger. This method takes no arguments.

** Note:** 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 a document without executing Interactive Reporting. The second line of the script shows how to turn off the Prompt to Save dialog box when an OnShutdown() method is executed:

```javascript
Documents["Dashboardtrigger.bqy"].OnShutdown()
Application.Quit(false)
```
OnStartup (Method)

**Applies To:** Document object

**Description:** The OnStartup() method is an Interactive Reporting document level function. It is executed when a document is opened and can be used to initialize the document and application for the user. 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 OnStartup event trigger. This method takes no arguments.

**Syntax:**

```plaintext
Expression. OnStartup()
```

**Expression Required:** An expression that returns an Interactive Reporting Document object

**Example:** This example displays the OnStartup method for an active document:

```plaintext
ActiveDocument.OnStartup()
```
Open (Method)

**Applies To:** Connection object, MetaDataConnection object, Documents collection

**Description:**
- **Documents** – Opens an existing Interactive Reporting document.
- **Connection** – Opens an existing Interactive Reporting connection file.

**Note:** The Documents.Open() object model syntax is not supported in a Hyperion System 9 BI + Workspace document.

**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:

```javascript
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):

```javascript
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 the Avalanche environment the following mapping occurs for the OpenUrl method:

OpenURL ”_parent” target is mapped to “_top”

Open URL “_self” target os 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:

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

**Example 2:** This example shows how to open a local Interactive Reporting (.bqy) document in the Interactive Reporting Web Client:

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

or

```java
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()
```
PrintOut (Method)

** Applies To:** ChartSection object, DataModelSection object, OLAPQuerySection object, PivotSection object, QuerySection object, ResultsSection object, Section object, TableSection object, ReportSection object

**Description:** Sends the information in a Report section to the printer.

**Note:** The ActiveDocument.Sections("SectionName").PrintOut() object model syntax is not supported in an Hyperion System 9 BI + Workspace document.

**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 the following sections:
- ChartSection
- DataModelSection
- OLAPQuerySection
- PivotSection
- QuerySection
- Section
- TableSection

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

```javascript
var StartPage = 1
var EndPage = 1
var NumCopies = 2
ActiveDocument.Sections("Pivot").PrintOut(StartPage,EndPage,NumCopies)
```
Process (Method)

**Applies To:** 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 a [OLAPQuerySection](#) or a [QuerySection](#) object

**Example:** This example shows how to process every query in a document:

```csharp
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:** 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 a document with three queries which appear as Query4, Query2, and Query1, the queries are processed in that order.

If no query order has been defined, queries are processed in the order in which they appear in the Section Catalog. For example in a document 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 Interactive Reporting (.bqy) document in an Alert box, set the processing the *Query* section to the second position in the Query Processing Order dialog, include the *Query* section in a *Process All* command, and then execute the Process All command for the document:

```
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:** This method provides you with the option to process stored procedures to obtain results.

This method is used in conjunction with the “SetStoredProcParam (Method)” on page 137.

**Syntax:**

```csharp
Expression.ProcessStoredProc()
```

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

```csharp
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 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 user is creating a new 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.

**Note:** The Application.Quit() object model syntax is not supported in a Hyperion System 9 BI + Workspace document.

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

**Syntax:**

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

**Expression Required:** An expression that returns an Application object

**Example:** This example shows how to quit Interactive Reporting silently:

```plaintext
Application.Quit(false)
```
Recalculate (Method)

**Applies To:** ChartSection object, DataModelSection object, DashboardSection object, OLAPQuerySection object, PivotSection object, QuerySection object, ResultsSection object, Section object, TableSection object, ReportSection object

**Description:** The Recalculate (Method) forces a section to recalculate itself. This method is important if you are using 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 a Results section to recalculate its 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: The RefreshAvailableValues (Method) generates a list of values for a limit. This method is equivalent to 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:* Use the RefreshDataNow (Method) to refresh a section immediately if you have selected to manually refresh the current section through the Object Model or the user interface. This method is used in conjunction with the RefreshData (Property) when the property value has been set to the bqRefreshDataManually constant.

*Syntax:* Expression.RefreshDataNow()

*Expression Required:* An expression that returns an object for the Pivot or Chart sections

*Example:* In the following example the Pivot section is set to be refreshed manually and immediately when the command is executed:

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

Applies To: CategoryItems collection, ChartSection object, Column object, ControlsDropDown object, ControlsListBox object, DataModelSection object, DashboardSection object, Join object, Limit object, OLAPQuerySection object, PivotLabels collection, PivotSection object, QuerySection object, ReportSection, Requests collection, ResultsSection object, Section object, SortItems collection, TableSection object, Topic object

Description: The Remove (Method) removes an individual item from a category. The Remove (Method) is called without a name or index to delete an individual object.

Syntax: Expression.Remove(NameOrIndex) or Expression.Remove()

Expression Required: An expression that returns an object to any of the following 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 (.bqy) document:

```
ActiveDocument.Sections["Pivot"].Remove()
```
**RemoveAll (Method)**

**Applies To:** AggregateLimits collection, AxisLabels collection, CategoryItems collection, Columns collection, ControlsDropDown object, ControlsListBox object, Joins collection, Limits collection, LimitValues collection, Parentheses collection, PivotLabelsTotals collection, Requests collection, Topics collection, TopLabels collection

**Description:** Removes all the items from a collection.

**Syntax:**

```
Expression.RemoveAll()
```

**Expression Required:** An expression that returns a collection for any of the following 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 chose to 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"].RemoveAllTopics()
```
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 are exported successfully, the “Export (Method)” on page 57 clears the export buffer. If sections are unsuccessful in being exported, then use this method to flush the export buffer. All sections set for export are cleared from the export buffer. For instance, if you specify a Report, Pivot, and Chart section to be exported by way of the “AddExportSection (Method)” on page 24, a call to RemoveExportSections (Method) nullifies the section set up for export. You could then specify the Export (Method) to export all sections.

**Syntax:**

```
Expression.RemoveExportSections()
```

**Example:** In this example sections are set to be exported using the AddExportSection (Method), cleared from the export buffer using the RemoveExportSections (Method), and then all of the documents sections are exported using the 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:** The RemoveFilterValue (Method) can be used to remove 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:
  ```csharp
  ActiveDocument.Sections["OLAPQuery"].Measures["Profit"].RemoveFilterValue()
  ```
- Expression to remove a single value based on a literal string value in a filter array:
  ```csharp
  Expression.RemoveFilterValue(string SearchStringOrMemberNameOrNumberRowsColumnIndexORUserDefinedAttributeOrSubstitutionVariable)
  ```
- Expression to remove a single value based on the position in a filter array:
  ```csharp
  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:
```csharp
ActiveDocument.Sections["OLAPQuery"].TopLabels["Line Name"].RemoveFilterValue()
```
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"].RemoveTopics(String DefinedJoinPathName)

Example: In this example, all topics are removed from the MyJoinPath join path:
RemoveTotal (Method)

**Applies To:** OLAPLabels collection

**Description:** The RemoveTotal (Method) 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 Alert message “Label Totals Have been Removed” displays:

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

 Applies To: QuerySection object

 Description: The ResetCustomerSQL (Method) resets the original SQL statement prior to processing and forces the Custom SQL window to open in a query. The “CustomSQLFrom (Method)” on page 47, “CustomSQLWhere (Method)” on page 47, and ResetCustomerSQL (Method) correspond to the edit SQL functionality in the Custom SQL dialog. However, no Custom SQL dialog displays when this method is executed.

 Note: To use the Custom SQL feature, the query’s 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.ResetCustomerSQL()

 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 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 (PeriodsQuarter='Q1')")
 ActiveDocument.Sections["Query"].Process()
 ActiveDocument.Sections["Query"].ResetCustomerSQL();
ResizeToBestFit (Method)

**Applies To:** Column object

**Description:** The ResizeToBestFit (Method) changes the width of a column to fit the data without clipping any information or displaying too much white space.

**Syntax:**

```
Expression.ResizeToBestFit()
```

**Expression Required:** An expression that returns a `Column` object

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

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

**Applies To:** OLAPCatalog object

**Description:** The RetrieveDimensions (Method) refreshes the dimension values in the Catalog pane.

**Syntax:**

```
Expression.RetrieveDimensions()
```

**Expression Required:** An expression that returns a Dimensions object

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

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

 Applies To: Connection object, MetaDataConnection object, Document object, WebClientDocument object

 Description: Saves the changes to a document or to an Interactive Reporting connection file (.OCE).

 Note: The ActiveDocument.Save() object model syntax is not supported in a Hyperion System 9 BI + Workspace document.

 Syntax: Expression.Save()

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

 - Connection
 - Document
 - WebClientDocument

 Example: This example shows how to create a new Interactive Reporting document and save it:

 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 a document or Interactive Reporting connection file (.oce) with a new name and/or location.

**Note:** The ActiveDocument.SaveAs() object model syntax is not supported in a Hyperion System 9 BI + Workspace document.

**Syntax:**

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

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

- Connection
- Document
- WebClientDocument

**Example:** This example shows how to save a document using a different name:

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

**Applies To:** ControlsDropDown object, ControlsListBox object, ControlsTextBox 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 any of the following items:

- ControlsDropDown
- ControlsListBox
- ControlsTextBox

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

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

Applies To: Application

Description: Sends a SQL string to a data source. No data is retrieved from the database.

Currently, this method does not send a SQL statement to the same database session to which your query is connected.

Note: If your SendSQL string is sending data modification commands, your database may require a commit statement. The commit behavior of the database may restrict which type of SQL string you may be able to send.

Because the SendSQL method requires an .OCE as an argument, it does not apply to a script written specifically 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 .OCE:

```javascript
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: The SetDrillThrough (method) enables you to define the mapping between a multi-dimensional database and a relational database. You must call the SetDrillThrough method for as many times as there are map or unmap actions.

 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 a selected dimension to a topic name. Use the boolean value true, to map a dimension and a topic. Use the boolean value false, to unmap a dimension. When a dimension is mapped to a topic, an internal link is created between the dimension and the selected topic.

 ● 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: The following 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. This method is a web-enabled method and does not apply to the Interactive Reporting Studio (Designer/Explorer). It can be used to automate logging on to the OnDemand Server.
SetPassword (Method)

**Applies To:** Connection object, MetaDataConnection object

**Description:** Sets the password that is used by the Interactive Reporting connection file (.oce) when connecting to the database.

**Syntax:**

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

**Note:** You must enclose the password with parentheses. If you do not, the string is created as a variable and there is no way to 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:

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

**Applies To:** QuerySection object

**Description:** The SetStoredProcParam (Method) provides you with the option to set up (select) stored procedures for obtaining 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 in conjunction with the “ProcessStoredProc (Method)” on page 112.

**Syntax:**

```plaintext
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:

```plaintext
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.

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

**Example 3:** This example shows how to pass a single parameter from a Dashboards object. It precedes the ProcessStoredProc method. The value of 1 in this example matches the position in the argument:

```plaintext
ActiveDocument.Sections["Query"].SetStoredProcParam(txtBeginDate.Text,1)
```
Shell (Method)

Applies To: Application object

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

This method is the preferred way to open the browser from the Desktop client (for example, from the Dashboard section). If you need to open an URL from a web-based client (for example, the web client), use the “OpenURL (Method)” on page 106.

Note: The Application.Shell() object model syntax is not supported in an Hyperion System 9 BI +Workspace document.

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:

```javascript
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:

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

 Applies To: OLAPQuery object

 Description: The ShowAsChart (Method) charts the OLAPQuery data set, and creates an OLAPChart and OLAPResults section automatically.

 Syntax: Expression.ShowAsChart()

 Example: The following script has been 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:* The SortByFact (Method) 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 the following 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`:

```vba
ActiveDocument.Sections["Pivot2"].TopLabels["Product Name"].SortByFact("Amount Sales", bqSortFunctionSum, bqSortAscend)
```
SortByLabel (Method)

**Applies To:** PivotLabels collection, CategoryItems collection

**Description:** The SortByLabel (Method) 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:

```csharp
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: The Spring (Method) 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 149.

 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.Sections["Report"].Body.Tables["Table"].Spring("Pivot")
SyncWithDatabase (Method)

Applies To: DataModel object

Description: The SyncWithDatabase (Method) 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:

```javascript
var MyDM = ActiveDocument.Sections["Datamodel"].DataModel
MyDM.SyncWithDatabase()
```
UnHide (Method)

**Applies To:** OLAPMeasure object

**Description:** The UnHide (Method) retrieves hidden measure items from the selected row or column to the OLAPQuery section.

**Syntax:**

```plaintext
Expression.UnHide()
```

**Expression Required:** An expression that shows a hidden OLAPMeasure object

**Example1:** This example shows how to hide the fact *Amount Sales*.

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

**Example2:** This example shows how to hide the OLAP measure *Sales Average*:

```plaintext
ActiveDocument.Sections["OLAPQuery"].Measures["Sales Average"].UnHide()
```
UnhideAll (Method)

 Applies To: AxisLabels (XLabels, Ylabels, and ZLabels)

 Description: The UnhideAll (Method) enables you to restore 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:** The Unselect (Method) causes an item in a list box to be unselected whether it has been selected or not.

**Syntax:** Expression.Unselect(Index As Number)

**Note:** Index is the nth item in the ListBox (index based 1).

**Expression Required:** An expression that unselects 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 has been populated with four values. These values can be selected and counted in a text box. The Unselect method has been added for each of the four values and any out of bound values:

```javascript
//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:** The UnSpring (Method) enables you to remove 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 144.

**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:

```vba
ActiveDocument.Sections["Report"].Body.Tables["Table"].Spring("Chart")
ActiveDocument.Sections["Report"].Body.Charts["Chart"].UnSpring()
```
UseAlternateMetadataLocation (Method)

**Applies To:** Connection object, MetaDataConnection object

**Description:** The UseAlternateMetadataLocation (Method) 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:

```javascript
var MyDM = ActiveDocument.Sections["DataModel"].DataModel
var MyOCE = "c:\OCEs\MetaOracle.oce"
MyDM.Connection.UseAlternateMetadataLocation(true, MyOCE)
```
Write (Method)

 Applies To:  Console

 Description:  The Write (Method) prints the output text specified by the OutputData parameter to the Console window.

 Note:  The Console.Write() object model syntax is not supported in a Hyperion System 9 BI + Workspace document.

 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 document 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 + ", ")
Writeln (Method)

 Applies To:          Console

 Description:  The Writeln (Method) prints the output text specified by the OutputData parameter to the
                Console window and puts a new line after the inserted text.

 Note:  The Application.Alert() object model syntax is not shown in an Hyperion System 9 BI +
        Workspace document, but the text in the first argument is written to the BI Server .log file.

 Syntax:  Expression.Writeln(OutputData As Value)

 Expression
 Required:  An expression that returns a Console object

 Example:  This example shows how to print the names of document 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

A
Activate (Method), 16
Add (Method), 17
AddAll (Method), 19
AddAllTopics (Method), 20
AddComputed (Method), 21
AddComputedItem (Method), 22
AddDrillThroughValue (Method), 23
AddDrillValue (Method), 26
AddExportSection (Method), 24
AddFilter (Method), 26
AddFilterValue (Method), 26, 28
AddTopic (Method), 30
AddTotal (Method), 31
AddTotals (Method), 32
Alert (Method), 33
AliasTable (Method), 34
AuditSQL (Method), 35
AutoSizeHeight (Method), 36
AutoSizeWidth (Method), 37

C
Call (Method), 38
ChartThisPivot (Method), 39
Close (Method), 40
commands, Help menu, ix
Connect (Method), 41
consulting services, xi
Copy (Method), 43
CreateConnection (Method), 44
CreateDateGroup (Method), 45
CreateLimit (Method), 46
CustomSQLFrom (Method), 47
CustomSQLWhere (Method), 48

D
Disconnect (Method), 49
documents
conventions used, x
feedback, xii
documents, accessing
Hyperion Download Center, ix
Hyperion Solutions Web site, ix
Information Map, ix
online help, ix
DoEvents (Method), 50
DrillInto (Method), 53
DrillThrough (Method), 53
DrillUp (Method), 54
Duplicate (Method), 55

E
education services, xi
ExecuteBScript (Method), 56
ExecuteBScript_method, 56
Export (Method), 57
ExportToStream (Method), 59

F
FocusSelection (Method), 61

G
GetCell (Method), 62

H
Help menu commands, ix
Hide (Method), 63
HideSelection (Method), 64
Hyperion Consulting Services, xi
Hyperion Download Center
accessing documents, ix
Hyperion Education Services, xi
Hyperion product information, xi
Hyperion support, xi
Hyperion Technical Support, xii

I
ImportDataFile (Method), 65
ImportSQLFile (Method), 66
InterruptQueryProcess (Method), 67
Item (Method), 68
ItemIndex (Method), 70

L
Layer (Method), 71
LoadFromFile (Method), 72
LoadSharedLibrary (Method), 73

M
ModifyComputed (Method), 74
ModifyRepositoryFileAnalyzer (Method), 77
ModifyRepositoryFileBQY (Method), 77
ModifyRepositoryFileBQYJob (Method), 81
ModifyRepositoryFileOther (Method), 76
ModifyRepositoryFileReports (Method), 83
ModifyRepositoryFileSQRJob (Method), 82
Move (Method), 85

N
New (Method), 86

O
OfficeHTMLFormulasEnabled (Property), 87
OnActivate (Method), 88
OnCellDbClick_meth, 89
OnCellDoubleClick (Method), 89
OnChange (Method), 90
OnClick (Method), 91
OnClientChange (Method), 91, 98
OnClientEnter (Method), 93, 98
OnClientExit (Method), 94
OnDeactivate (Method), 95
OnDoubleClick (Method), 96
OnEnter (Method), 97
OnExit (Method), 98
OnPostProcess (Method), 99
OnPreProcess (Method), 100
OnRowDoubleClick (Method), 101
OnSelection (Method), 102
OnShutdown (Method), 103
OnStartup (Method), 104
Open (Method), 105
OpenURL (Method), 106

P
PivotThisChart (Method), 107
PivotTo (Method), 108
PrintOut (Method), 109
Process (Method), 110
ProcessAll (Method), 111
ProcessStoredProc (Method), 112
ProcessToTable (Method), 113

Q
Quit (Method), 114

R
Recalculate (Method), 115
Refresh (Method), 116
RefreshAvailableValues (Method), 117
RefreshDataNow (Method), 118
Remove (Method), 119
RemoveAll (Method), 121
RemoveAllTopics (Method), 122
RemoveExportSection (Method), 123
RemoveFilterValue (Method), 124
RemoveTopic (Method), 125
RemoveTotal (Method), 126
Removing a section programmatically, 120
ResetCustomerSQL (Method), 127
ResizeToBestFit (Method), 128
RetrieveDimensions (Method), 129

S
Save (Method), 130
SaveAs (Method), 131
Select (Method), 132
SendSQL (Method), 133
SetDrillThrough (Method), 134
SetODSPassword (Method), 135
SetPassword (Method), 136
SetStoredProcParam (Method), 137
Shell (Method), 138
ShowAll (Method), 139
ShowAsChart (Method), 140
SortByFact (Method), 141
SortByLabel (Method), 142
SortNow (Method), 143
Spring (Method), 144
SyncWithDatabase (Method), 145

T

technical support, xii

U

UnHide (Method), 146
UnhideAll (Method), 147
Unselect (Method), 148
UnSpring (Method), 149
UseAlternateMetadataLocation (Method), 150

W

Write (Method), 151
Writeln (Method), 152