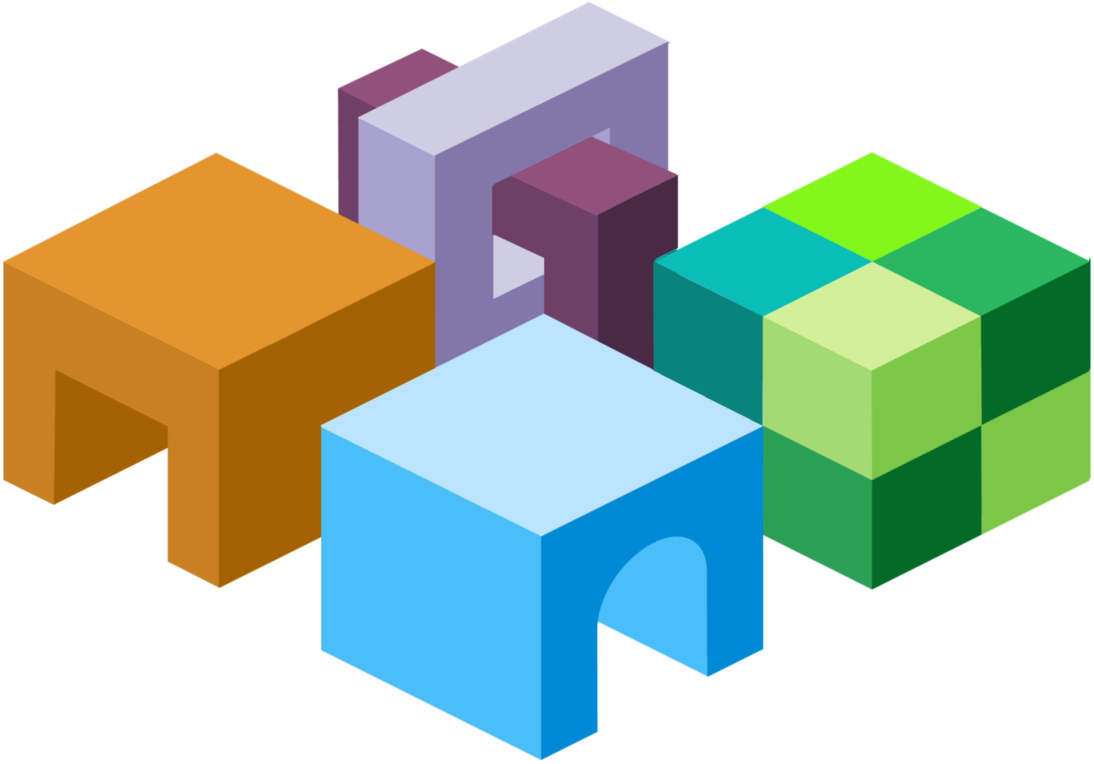


HYPERION® INTERACTIVE REPORTING

RELEASE 11.1.2

OBJECT MODEL AND DASHBOARD DEVELOPMENT SERVICES DEVELOPER'S GUIDE

VOLUME I: DASHBOARD DESIGN GUIDE

ORACLE®

ENTERPRISE PERFORMANCE MANAGEMENT SYSTEM

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

Authors: EPM Information Development Team

Contents

Part I
Overview

In Overview:

1 Working with Dashboard Sections

In This Chapter

Dashboard Sections

Dashboards are customizable documents that enable developers to build and deploy analytic applications and end users to access information.

The dashboard sections of Oracle's Hyperion® Interactive Reporting provide a streamlined, push-button approach to querying databases, combining reports, and enhancing controls to develop and deliver prepackaged business content that includes these items:

- Simple forms to collect multiple input parameters for a report
- Executive dashboard applications, with visual drill-down from high-level metrics to underlying data
- Browser-style navigation pages to assist users in maneuvering within and between documents

Within Oracle's Hyperion® Interactive Reporting Studio and Oracle's Hyperion® Interactive Reporting Web Client, developers can customize dashboard sections to create interfaces that focus on the data views relevant to end users. When an end user opens an Interactive Reporting document file, the customized dashboard section is displayed.

Users navigate dashboard sections by clicking buttons and entering parameters. Button clicks, item selections, or navigation sequences invoke scripts that are processed in the background. No in-depth knowledge of the data structure or the Interactive Reporting Studio and Interactive Reporting Web Client applications is required.

Inserting, Renaming, Deleting, and Changing Mode of Dashboard Sections

Like other report sections, dashboard sections can be added to an Interactive Reporting document. Within Interactive Reporting documents, the dashboard section is displayed at the top of the Sections pane.

Inserting Dashboard Sections

Newly added dashboard sections are listed after existing dashboard sections.

➤ To insert dashboard sections, select **Insert**, then **New Dashboard**.

Renaming Dashboard Sections

The first dashboard section inserted in an Interactive Reporting document file is, by default, named *Dashboard*. Subsequent dashboard sections are numbered sequentially; for example, Dashboard2, Dashboard3, and so on.

➤ To rename dashboard sections:

1. From **Sections,** select a dashboard section.
2. Select **Edit**, then **Rename Section.**

   Section Label is displayed.
3. In **Label,** enter a name, and click **OK.**

Deleting Dashboard Sections

➤ To delete dashboard sections:

1. From **Sections,** select a dashboard section.
2. Select **Edit**, then **Delete Section**

   Delete Section is displayed.
3. Click **Delete.**

Switching Between Design and Run Modes

Dashboard section modes:

- **Design**—Used to design dashboard sections. Available objects are displayed in the Elements
- **Run**—Used to deploy dashboard sections to end users. The Elements is empty.

All dashboard sections within an Interactive Reporting document file use the same mode. Dashboard sections of Interactive Reporting document files open by default in Run mode. Changing one section to Design mode changes all sections to Design mode.

➤ To switch to Design mode, select **Dashboard**, then **Design Mode** or click , or press **Ctrl +D**.

Working with Dashboard Objects

Embeddable objects used to construct custom dashboard sections:

- **Sections**—Results, chart, pivot, table, and OLAP sections from the active Interactive Reporting document file. When you embed a section in a dashboard section, the section is resized, and its data is updated.

  Note: In Run mode, you can sort active embedded tables and results (from the shortcut menu) and resize table and results columns.

- **Graphics**—Objects, such as lines, rectangles, and pictures, for which you can set colors and border properties. These are available from Elements.
  - Line
  - Hz line
  - Vt line
  - Rectangle
  - Round rectangle
  - Oval
  - Text label, can be used as a caption
  - Picture, insert BMPs, GIFs, JPGs, and PNGs
- **Controls**—Objects for which you can set fonts and default values. Controls, which provide users a way to interact with the application, can be populated with values at design time or dynamically populated using JavaScript.

Table 1 Dashboard Control Objects

Control Object	Suggested Use
Command button	Initiate or activate a process
Radio button	Select one from a group of choices
Check box	Toggle an option, on and off or true and false
List box	List multiple values from which users can select one or more
Drop down	List multiple values from which users can select one

Control Object	Suggested Use
Text box	Gather and display user input
Embedded browser	Display external content through a browser
HyperLink	Link to external content through a hyperlink

- **Resources**—Pictures or images loaded in the Resource Manager.

Inserting Dashboard Objects

➤ To embed objects in dashboard sections:

1 **Select Dashboard, then Design Mode, or press Ctrl+D.**

2 **From Elements, expand the folder that contains the object to be inserted.**

3 **Drag the object onto the content area.**

➤ To insert control and graphic objects:

1 **Select Dashboard, then Design Mode, or press Ctrl+D.**

2 **Select Dashboard, then Insert Graphic or Dashboard, thenInsert Control, and, from the menu, select a graphic or control object.**

3 **Click within the content area to insert the control or graphic.**

Deleting Dashboard Objects

➤ To delete embedded sections, controls, and graphics:

1 **Select Dashboard, then Design Modeor press Ctrl+D.**

2 **From the content area, select one or more objects.**

To select multiple objects, press and hold Ctrl while selecting objects.

3 **Select Dashboard, then Remove Selected Items.**

External Content in Dashboards

You can incorporate external Web-based content into your dashboards, embedding the content in your reports sections and thus enhancing your reports. You can embed stock tickers, calendars, and document objects and launch the content of a URL-based hyperlink in a separate window.

Embeddable external Web-based dashboard objects:

- Interactive Reporting document files

- Interactive Reporting jobs
- Oracle's Hyperion® SQR® Production Reporting jobs
- Oracle's Hyperion® Web Analysis
- Oracle Hyperion Financial Reporting, Fusion Edition

Interactive Reporting Studio and Interactive Reporting Web Client provide two controls for working with Web-based content:

- Embedded browser control—Essentially, an instance of a Web browser window, positioned on the dashboard page. Similar to an embedded section object, except that the content is anything that can be referenced through a URL and rendered in a Web browser window, rather than in a section. There are no recognizable events in the control; that is, you can not create scripts on an embedded browser control.
- Hyperlink control—A URL-based hyperlink control. The content may be displayed in a pop-up window or the current window.

Working with Embedded Browser Controls

An embedded browser control renders and executes in all Interactive Reporting environments (Interactive Reporting Studio, Interactive Reporting Web Client, and Oracle Enterprise Performance Management Workspace, Fusion Edition).

Attributes and behavior of embedded browser controls, typical of all dashboard controls:

- Is added to the dashboard section (in Design mode) by dragging an instance from Elements, sizing the instance, and setting properties
- Uses common dashboard-control properties: name, visible, locked, tab order, accessibility, cut, copy, paste, auto-alignment and sizing

Properties of embedded browser controls, similar to properties of embedded sections:

- Show Scrollbars
- Reference to the content—For embedded sections, the reference is the name of a chart, pivot, or table that is contained in the Interactive Reporting document file. For the embedded browser control, the reference is a URL. If the content is a generic URL (such as a stock ticker), the designer can reference it by entering the URL. If the content is an Interactive Reporting document, file in the repository, the designer can use GUI controls to select the Interactive Reporting document file and the preferred parameters, thus creating the appropriate Smartcut URL.

  Note: A report section that is contained in one Interactive Reporting document file and displayed in an embedded browser object of another Interactive Reporting document file cannot be printed.

➤ To create instances of embedded browser controls:

1 Select **Insert**, then **New Dashboard.**

Inserting a dashboard section changes the document to Design mode. The content area is blank, and Elements displays the sections, graphics, controls and resources available for embedding in a dashboard section.

2 From **Elements**, expand **Controls**, and drag the embedded browser control onto the content area.

3 Double-click the control object.

Objects is displayed.

4 In **Name** enter a unique name for the control.

The default name is EmbeddedBrowser followed by a number.

5 In **Title** enter a title for the dashboard section.

The Title is only used when the dashboard section is printed.

6 Select **Visible** to display the embedded browser control when the dashboard section is executed.

By default, Visible is selected.

7 Select **Locked** to prevent the embedded browser control from being moved or deleted.

By default, Locked is not selected.

8 From **Show Scrollbar,** to indicate when scroll bars are displayed on the embedded browser control, select a scrolling option:

- Always
- Never
- Automatic (the default)

9 **Optional:** To define explicitly the URL of the content to be displayed in the embedded browser, in **URL**, enter the URL address.

When the embedded browser is rendered, the URL is used “as is” (server information is not appended).

You can reference Oracle's Hyperion Reporting and Analysis repository content by entering an explicit URL. In this case, the URL is treated like any other external reference (no special authentication support).

It is important to plan how the content of the URL is displayed in the embedded browser. An action from within the embedded content can result in the browser, rather than only the contents of the embedded browser control, being replaced.

10 **Optional:** To select a repository document (and associated properties) or an object to be displayed in the embedded browser control or HyperLink object, select **Repository**, and perform one or more actions:

- **Document**—Enter manually the path and name of the preferred repository document
- **Browse**—Browse the Reporting and Analysis repository for the preferred document
- **Options**—Launch the Document Options dialog box so that you can select display properties for a EPM Workspace document (See “Document Options” on page 20)

11 Click **OK.**

Referencing A Name With A Single-Byte Character

If you are using the Embedded Browser control of the Dashboard section to embed content from the repository, and it references a name (for example, file name, directory, section, parameter) that contains a single-byte character with a value greater than 127 (for example, accent characters), and the SmartCut encoding for URLs is set to "default", then the characters display as scrambled and the embedded content may not display properly.

➤ To change the encoding property for SmartCuts:

1. Log in to EPM Workspace with a user ID that has an Administrator role.
2. Select **Administration** on the Module menu.
3. Click the **System** tab and navigate to **SmartCut**.
4. Change the Encoding for URLs property to **UTF8**.

Browsing the Repository

With proper authentication, you can select a document to embed from the Reporting and Analysis Repository dialog box. Authentication is provided through an authentication service, which reviews user credentials at login time and enables users to connect. The service also determines a user's group membership, which, with user roles, affects what content and other system objects (resources) the user can view and modify.

For users authenticated to a global service manager (GSM), the Foundation Browse dialog box is displayed automatically. Authentication is assumed for Interactive Reporting Web Client and EPM Workspace documents, because they receive a URL stamp when they are created. The stamp provides GSM access.

Interactive Reporting Studio users who are not authenticated are prompted to enter a valid server address on the Connect to Server dialog box (Tools, then Connect to Server). When the user selects Connect (and the connection succeeds), the server address is persisted in the Windows registry. Thus, the address can be used as the default server address, if no valid address can be obtained from the session or the Interactive Reporting document.

After a valid server address is determined, the Interactive Reporting Studio application contacts the data access servlet, and an authentication dialog box is displayed. After user credentials are validated, the user is authenticated, and the Reporting and Analysis Repository dialog box is displayed.

➤ To log into the Reporting and Analysis repository and embed an object:

1. In the **Objects** dialog box, select **Repository**.
2. Click **Browse**.

   Connect to Server is displayed.
3. In the **Connect to Server** dialog box, enter a server address, and click **Connect**.

   The Reporting and Analysis Login dialog box is displayed.

4 **Enter your user name and password, and click Login.**

After you are successfully authenticated, Browse Listing is displayed.

5 **Navigate to the folder that contains the repository object to be embedded.**

6 **Double-click the object.**

Verify that the full path name and object name are displayed to the right of the Selection field.

7 **Click Apply.**

The document name is displayed in the Document field of the Object dialog box.

Document Options

You use the Document Options dialog box to select display properties for a document to be viewed in EPM Workspace. The properties that you select prepare a Smartcut (a URL reference to a Reporting and Analysis repository object, for example, an Interactive Reporting document or job).

Document types available for display in EPM Workspace:

- Interactive Reporting document (BQY)
- Interactive Reporting Job
- Production Reporting Job
- Web Analysis
- Financial Reporting
- Others

Note: If a hyperlink control display type is New Window or Current Window, the Document Options dialog box looks and behaves as described in “Working with Embedded Browser Controls” on page 17.

If Interactive Reporting Studio and Interactive Reporting Web Client determines the document type prior to the display of the Document Options dialog box, then the Document Type drop-down list presets the document type, and no document-type choices are available.

If Interactive Reporting Studio and Interactive Reporting Web Client cannot determine the document type prior to display of the Document Options dialog box, the default is Others, and you can select one of the five specific document types.

Depending on the document, specific options are available.

Table 2 Interactive Reporting Document Type Display Options

Option	Description
Document Type	For Interactive Reporting Studio and Interactive Reporting Web Client, select Interactive Reporting document.
Section	Specify which Interactive Reporting document section is displayed. If the list of EPM Workspace-enabled sections is determined prior to display of the Document Options dialog box, the list is displayed. The default section name is the name of the default section of the Interactive Reporting document. If the section list cannot be determined, a blank text box is displayed, and you can enter the section name manually. If the text box is blank, the default section is displayed when Browse is clicked in the Properties dialog box.
Toolbar	Specify which EPM Workspace toolbar is displayed: ● Standard—Contains the controls for commonly used operations, such as Open and Save ● Paging—Contains the section paging controls (Page Left, Page Up, Page Down, and Page Right) and indicates the current page (Page x of *y*) ● Section Navigation—Contains the navigational controls (Back, Forward, and Dashboard Home) ● Paging and Section Navigation—Contains the controls for paging and section navigation ● None
Smartcut Parameters	Specify key value pairs for the document and add parameters to the URL. To add a parameter, enter the parameter into the Smartcut Parameters text box, and click Add. When the URL is formed, the parameter is appended to the end of the URL. For example, the Smartcut Parameter “ShowMenubar” specifies state of the toolbar when the Interactive Reporting document is opened. Parameter values include: ● Standard ● Navigation (for Paging) ● Section Navigation and Navigation ● None ● Numbers (1, 2, 3, 4 respectively) To remove parameters from the list, select the parameters, and click Remove.

Table 3 Interactive Reporting Job Document Type Display Options

Option	Description
Document Type	For Interactive Reporting Studio and Interactive Reporting Web Client, select Interactive Reporting Job.
Section	Specify which Interactive Reporting document section is displayed. If the list of EPM Workspace-enabled sections is determined prior to display of the Document Options dialog box, the list is displayed. The default section name is the name of the default section of the Interactive Reporting document. If the section list cannot be determined, a blank text box is displayed, and you can enter the section name manually. If the text box is blank, the default section is displayed when Browse is clicked in the Properties dialog box.

Option	Description
Toolbar	Specify which EPM Workspace toolbar is displayed: ● Standard—Contains the controls for commonly used operations, such as Open and Save ● Paging—Contains the section paging controls (Page Left, Page Up, Page Down, and Page Right) and indicates the current page (Page x of *y*) ● Section Navigation—Contains the navigational controls (Back, Forward, and Dashboard Home) ● Paging and Section Navigation—Contains the controls for paging and section navigation ● None
Smartcut Parameters	To add a parameter, enter the parameter into the Smartcut Parameters text box, and click Add. When the URL is formed, the parameter is appended to the end of the URL. To remove parameters from the list, select the parameters, and click Remove.

Table 4 Production Reporting Job Document Type Display Options

Option	Description
Document Type	For Production Reporting, select Production Reporting Job.
Run Job	Specify whether the job should be executed. Not selected (not to be run) is the default.
Smartcut Parameters	Specify key value pairs for the document and add parameters to the URL. To add a parameter, enter the parameter into the Smartcut Parameters text box, and click Add. When the URL is formed, the parameter is appended to the end of the URL. To remove parameters from the list, select the parameters, and click Remove.

Table 5 Web Analysis Document Type Display Options

Option	Description
Document Type	For Web Analysis, select Web Analysis.
Smartcut Parameters	Specify key value pairs for the document and add parameters to the URL. When the URL is formed, the parameter is appended to the end of the URL. To remove parameters from the list, select the parameters, and click Remove.

Table 6 Financial Reporting Document Type Display Options

Option	Description
Document Type	For Financial Reporting, select Oracle Hyperion Financial Reporting, Fusion Edition
Display Format	Specify the type of format in which to display the report (HTML or PDF). The default is HTML.
Smartcut Parameters	Specify key value pairs for the document and add parameters to the URL. When the URL is formed, the parameter is appended to the end of the URL. To remove parameters from the list, select the parameters, and click Remove.

Table 7 Others Document Type Display Options

Option	Description
Document Type	Select Others to add parameters to the URL.
Smartcut Parameters	Add parameters; for example, add Oracle's Hyperion® SQR® Production Reporting Job parameters or Interactive Reporting document limits (filters). To add a parameter, enter the parameter into the Smartcut Parameters text box, and click Add. When the URL is formed, the parameter is appended to the end of the URL. To remove parameters from the list, select the parameters, and click Remove.

Setting Dashboard Home Sections

Use the Dashboard Home dialog box to select a section to set as the home dashboard section (the first section that is displayed when the document is opened). In addition for Interactive Reporting users only, use the Dashboard Home dialog box to enable the preloading of the active home section in order to increase loading performance in EPM Workspace. This option is enabled only when the selected dashboard section in an Interactive Reporting document contains a minimum of one dashboard with at least one embedded browser.

Section rules define the first displayed section when a document is opened:

- Defined through the object model, for example, Activate()
- Selected on the Dashboard Home dialog box
- If not defined through the object model or the user interface, the section that was last saved is opened first (This behavior is a change in Release 9.3, that may cause the document to open to the home section. In releases earlier than Release 9.3, the same document might have opened to the last saved section or another section)
- If a home dashboard is not set, then the last saved section is opened (If the user cannot access the last saved section; for example, due to an adaptive state which prevents a user from viewing some sections, then the document is opened in a section available to the user)

In releases earlier than Release 9.3, the default home section was the first section added to the document. In Release 9.3, there is no default home section; that is, if no home section is defined, none is assumed in Interactive Reporting.

To enable the Dashboard Home toolbar button when an Interactive Reporting document file is deployed in the EPM Workspace, Dashboard home must be explicitly set in Interactive Reporting Studio or Interactive Reporting Web Client.

Note:

➤ To specify a home dashboard section:

1 **From the Dashboard menu, select Home Dialog.**

2 **Select the dashboard to use for the home section.**

3 **Optional:** To use a hidden dashboard section, select **View**, then **Unhide section** and select the section before displaying the Dashboard Home dialog box.

When you save the Interactive Reporting document, the section is no longer hidden.

4 To preload embedded browser objects, click **Preload.**

5 Click **OK.**

Note:

Setting a home dashboard section in Design mode activates (Dashboard Home) on the toolbar.

Working with HyperLink Controls

Hyperlink controls behave like embedded browser controls, except that their URL-based content is displayed as follows:

- Current window
- Top window
- New window
- Named window
- No window

Hyperlink-control behavior depends upon the display window selection and the type of application (Interactive Reporting Studio, Interactive Reporting Web Client, or EPM Workspace)

Table 8 Hyperlink control behavior

Target	Interactive Reporting Studio	Interactive Reporting Web Client	EPM Workspace
New window	Launches new window	Launches new window	Launches new window
Current window	Launches new window	Replaces current window	Replaces content of the EPM Workspace tab, if the content is derived from the Oracle's Hyperion Reporting and Analysis repository. Otherwise, opens the hyperlink object in a new window.
Top window	Launches new window	Replaces current window	Opens in a new EPM Workspace tab, if the content is derived from the repository. Otherwise, opens the hyperlink object in the top window.
Named window	Launches new named window	Launches new named window	Launches new named window

Other properties of hyperlink controls are similar to the properties of text label graphics and the OpenURL method. Hyperlink controls are added like embedded browser controls are added, except that the control name in Elements is Hyperlink.

➤ To create instances of hyperlink controls:

1 **Select Insert, then New Dashboard.**

Inserting a dashboard section changes the document to Design mode. The content area is blank, and Elements displays the available sections, graphics, and control objects.

2 **From Elements, expand Controls.**

3 **Drag the hyperlink control onto the content area, and double-click the control.**

Properties is displayed.

4 **On the Object tab, enter a unique name for the hyperlink control.**

The default name is Hyperlink.

5 **Enter a title for the dashboard section.**

The Title is only used when the dashboard section is printed.

6 **Select Visible to display the hyperlink control when the dashboard section is executed.**

By default, Visible is selected.

7 **Select Locked to prevent the hyperlink control from being moved or deleted in dashboard Design mode.**

By default, Locked is not selected.

8 **From Display in, select where to display the contents of the hyperlink control:**

- **New Window**—A pop-up window is created every time the hyperlink is clicked. This option, which is the default, maps to the OpenURL OM method "_blank" target.
- **Current Window**—If the execution application is Interactive Reporting Web Client or EPM Workspace, the content replaces the EPM Workspace content area (for example, excluding the surrounding browse application panes). In Interactive Reporting Studio, a pop-up window is displayed.
- **Top Window**—If the execution application is Interactive Reporting Web Client or EPM Workspace, the content replaces the top HTML window. This option maps to the OpenURL OM methods "_top" target.
- **Named Window**—A named pop-up window is created. If a named window is currently displayed, the URL replaces the content of the current window. When this option is selected, the user can enter the new window name in the Target window text box that is displayed below the drop-down list. This option maps to the user-specified, target-name option of the OpenURL method.

9 **Optional: To define explicitly the URL of the content to be displayed in the hyperlink control, in URL, enter the URL address.**

When the hyperlink control is rendered, the URL is used “as is” (server information is not appended).

You can reference repository content by entering an explicit URL. In this case, the URL is treated like any other external reference (no special authentication support).

It is important to plan how the content of the URL is displayed in the hyperlink control. An action from within the hyperlink content can result in the browser, rather than only the contents of the hyperlink control, being replaced.

10 **Optional:** To select a repository document (and associated properties) to be displayed in the hyperlink control, select **Repository,** and perform one or more actions:

- **Document**—Enter the path and name of the preferred repository document.
- **Browse**—Browse the repository for the preferred document.
- **Options**—Launch the Document Options dialog box, from which a user can select display properties for a selected EPM Workspace document. If the display type is New Window or Current Window, the dialog box looks and behaves as described in "Working with Embedded Browser Controls" on page 17. Also see "Document Options" on page 20.

11 Click **OK.**

Embedded Section Objects

Results, Pivot, Chart, Table, OLAP, and CubeQuery sections can be embedded in any Interactive Reporting document and viewed in EPM Workspace. Data is updated in EPM Workspace as it is updated in the original sections.

The limitations to Interactive Reporting document files embedded in a dashboard through an embedded browser or hyperlink control include:

- Online help is available through a toolbar or a dialog box.
- Number formatting options are not available.
- The Reference sub dialog box of the Computed Item dialog box is not available.

Properties that define what users can do with embedded sections:

- View-only—Can view static reports. (The reports are displayed as thumbnails in the dashboard section—as currently defined in the native-report section. Users cannot interact with the reports.)
- Hyperlink—Can navigate to original sections by clicking thumbnails in the dashboard section.
- Active—Embedded section objects in active mode enables users to interact with reports. Selecting a live report activates it in-place for drill down, pivoting, and other analysis functions.

Note: When context menu changes made through the Object Model have been associated with embedded section objects of the same dashboard, the entire dashboard should be refreshed for changes to take effect. To refresh an entire dashboard, switch to another section, and then switch back to the same section. This was designed so that menu changes could be executed from the Startup script or from the user interface and saved with the document. It is not expected to be a frequent operation.

Embedded Section Object Scrollbar and Auto-Sizing Properties

Scrollbar options are only available when an embedded object is view only. Scrollbar properties on the Object tab determine how and when they are displayed: The horizontal and vertical scrollbars can be used to scroll only the current page of the original section

- If the Show Scrollbar option is set to always, vertical and horizontal scrollbars are displayed adjacent to but outside the container boundaries of the sections and do not obstruct them.
- If the Show Scrollbar option is set to never, vertical and horizontal scrollbars are not displayed.
- If the Show Scrollbar is set to automatic, The horizontal and vertical scrollbars appear when necessary and not as a default.

If Auto-Size is enabled, the object data is scaled to fit within the container boundary of the section, and the object cannot be scrolled. This means only the first page of the report is displayed. Disabling the autosize option makes the scrollbars active.

Embedded Section Object View Only Behavior

Embedded section objects in view only mode enables users to view, but not interact with, reports. View-only is the default setting for all embedded sections supported for embedded objects in the EPM Workspace. The behavior of view-only embedded section objects is:

- Results and Table—Selection of columns, rows, or column titles is not permitted.
- OLAPQuery, CubeQuery and Pivot—Selection of report cells and handles is not permitted. (For a Pivot embedded section object in active mode, a fact cell can be double-clicked to launch a script if there is an OnCellDoubleClick event and JavaScript associated with the event . The event is associated with the dashboard object, not with the pivot section. See Embedded Pivot Section Object Interactivity)
- Chart—Selection of labels, bars, lines, and pie slices is not permitted.
- Results, table, OLAPQuery, pivot, and chart—EPM Workspace speed-menu options, which typically are displayed for selected sections, are not available.
- Reports—An embedded Report section is shown as a contiguous band that includes all the pages of the section. design time helpers such as the Ruler and specific graphical markups to outline boundaries of report elements are not displayed.

Embedded Section Object Active Mode in the EPM Workspace

In Active mode embedded section objects are displayed on the page in the same position, of the same size and with the same Active options applied as in Interactive Reporting and Interactive Reporting Web Client and within the feature constraints of the EPM Workspace.

An embedded section object in active mode includes this behavior:

- If the "Scrollbars Always Shown" property of the embedded section object is enabled, vertical and horizontal scrollbars are displayed adjacent to, but outside the defined object's container boundary, and do not obstruct the object. Vertical and horizontal scrollbars are always available.
- If the scrollbar property is not set, vertical and horizontal scrollbars appear only when the object has the focus and are adjacent to, but outside the defined object's container boundary and not obstruct the object. The scrollbars continue to show until the user applies focus to another object, selects a toolbar button (other than the EPM Workspacepaging toolbar buttons) or until the Dashboard section is exited.
- Scrolling of the object using either of the object's scrollbars (vertical or horizontal) enables only the scrolling of the current page if the parent section
- If the parent section has multiple pages and the embedded section object has focus, the paging buttons in the EPM Workspace become enabled to allow for the paging of the parent section.

.

Embedded Section Objects Shortcut Menu

You can customize the shortcut menus added to active embedded section objects in the Dashboard sections of Interactive Reporting Studio, Interactive Reporting Web Client and EPM Workspace. Shortcut menus are enabled by selecting the object and right-clicking or pressing Shift + [F10].

Shortcut menu functionality is added through the Interactive Reporting Studio, or Interactive Reporting Web Client. The designer selects which options are eligible on a shortcut menu from a list of all applicable options. Actions performed on the active embedded section have an immediate effect on the parent section. Actions performed on the parent section of the embedded section object and Object Model scripted command also update the embedded section object. The list of available shortcut menu options is context sensitive: and depends on:

- type of application in which the section is available.
- what component of the report is selected (for example, label, or fact, etc.)
- state when the component selection is made (if items have been hidden, if additional columns are available for drill down, etc).

Several of the shortcut menu options listed for the individual sections EPM Workspace are not available as shortcut menu options in Interactive Reporting Studio or Interactive Reporting Web Client and vice versa. They may be are available by other means such as the Standard Toolbar or mouse controls (i.e. double-click column border). All actions which are possible for the active embedded section object in Interactive Reporting Studio or Interactive Reporting Web Client are included as a shortcut menu options for the EPM Workspace.

Adding a Shortcut Menu to an Embedded Section Object

You can specify which shortcut menu are displayed for an embedded section object.

➤ To add a shortcut menu to an embedded section object:

1 **In the Interactive Reporting Studio or Interactive Reporting Web Client, add an embedded section object to a Dashboard section in design mode.**

Note: An embedded section object cannot be added in the EPM Workspace.

Embeddable sections are:

- Results and Tables
- Chart
- Pivot
- OLAPQuery
- CubeQuery

2 **Select the embedded section object and on the shortcut menu, select Properties.**

The Properties dialog box is displayed.

Note that multiple component selection can be performed in two ways:

- Shift + Click—Selects a range of components
- Ctrl + Click—Selects multiple components one at a time (also enables you to deselect components one at a time)

3 **Select the Object tab.**

4 **In Make Embedded Section, enable Active.**

5 **Select OK.**

6 **Select the embedded section object again and on the shortcut menu, select Shortcut Menu.**

The Shortcut Menu dialog box is displayed. The dialog shows an Available and Selected Pane, the shortcut menu name and the name of the application in which the shortcut menu is available (Interactive Reporting Web Client is included under "Desktop". By default all applicable shortcut menu options have been added to the Selected pane.

7 **To accept the default shortcut menu options, select OK.**

Note: The relative order of the shortcut menu items cannot be changed.

To remove a default shortcut menu option, select the option in the Selected pane and click ⇦.

Active Embedded Results/Table Section Object Shortcut Menus

Shortcut menus can be launched by selecting the column title or column body for individual and multiple columns. Multiple column selection can be performed in two ways:

- Shift + Click—Selects a range of columns
- Ctrl + Click—Selects multiple columns one at a time (also enables you to deselect of columns one at a time)

When a column or columns are selected, the EPM Workspace shortcut menu displays with the following options:

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

Active Embedded Pivot Section Object Shortcut Menu

Shortcut menus can be launched by selecting the top or side labels, fact column, fact column label, column title or the pivot handles of the top and side labels. Individual or multiple selections of pivot components is available. Multiple component selection can be performed by selecting Ctrl + Click.

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

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

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

- Sort Ascending (EPM Workspace)
- Sort Descending (EPM Workspace)
- Refresh Pivot (EPM Workspace)

Active Embedded Chart Section Object Shortcut Menu

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

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

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

Active Embedded OLAPQuery/CubeQuery Section Object Shortcut Menu

Selection within the object is permitted by either selecting the top or side labels, fact column, fact column label or column title or by selecting the handles of the top and side labels. Individual selections of these components can be performed as well as the selection of multiple components. Multiple component selection can be performed in two ways:

- Shift + click—Selects a range of columns
- Ctrl + click—Selects multiple columns one at a time (also enables you to deselect of columns one at a time)

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

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

Embedded Section Object Paging

Paging standards used by embedded objects:

- If the original section includes multiple pages and the embedded section is selected, the paging buttons of the EPM Workspace toolbar are enabled. When a toolbar button is clicked, the updated page view is displayed within the borders of the embedded section.
- Embedded-section paging is independent of original-section paging, and the paging of each embedded section is independent of the paging of other sections, even if the other sections are embedded in the same dashboard section and derived from the same original section.

- For hyperlinks, page view cannot be updated.
- In EPM Workspace, hyperlinks are displayed in new tabs, if the hyperlink content is derived from the repository and the hyperlink is set to display in the top window. If you select the current window and the content is from the repository, the repository content replaces the current-tab content. Otherwise, the content is displayed in a new window.
- If an action performed on the original section invalidates the current-page view of the embedded section, the embedded-section view moves to the next valid page.
- Non-paging-related actions performed on original sections (such as column modification and formatting) are propagated to all embedded sections (regardless of active-page or embedded-section mode). In these cases, embedded sections retain their current page views.

Embedded Section Object Design Considerations

Reminders for designers working with embedded objects:

- Because embedded section scrollbars are sometimes displayed outside defined boundaries, objects and controls within embedded sections must be placed carefully, to avoid obstructing the scrollbars.
- When you connect to a computer to start the BI Service on Windows, ensure that the color-setting for the display is at least 16-bits. If the color setting is less than 16-bits, users may encounter extremely long response times when opening chart sections of Interactive Reporting documents. Color setting is a significant pre-startup step, especially when services are started remotely (for example using VNC, Terminal Services, Remote Administrator, or Timbuktu), because many remote administration clients connect with only 8-bit colors by default.

Embedded Pivot Section Object Interactivity

Pivot tables embedded in dashboards as active sections can be scripted to enable users to double-click cells and build interactivity around them. This feature is available on all fact cells of pivot tables. When a fact cell is double-clicked, the OnCellDoubleClick event is fired, and JavaScript associated with the event can be executed. The event is associated with the dashboard object, not with the pivot section.

Embedded pivot section interactivity is available for Interactive Reporting Studio and Interactive Reporting Web Client. It is not available for documents deployed in EPM Workspace.

Prior to executing the script associated with the OnCellDoubleClick event, certain object-tree properties are set. The properties are then valid from wherever within the object model that the object is accessible, such as from document startup and shutdown scripts. If the pivot section is recalculated, the properties are reset to the initial, default state, as if no double-click event had occurred.

Properties set prior to executing OnCellDoubleClick events:

- OnCellDoubleClick (Method)
- FactName (Property)

- CellValue (Property)
- TopLabelValues (Array)
- SideLabelValues (Array)

See "Object-Level Events" on page 60.

Gauges and (Live) Charts

Gauges and (Live) Charts enable you to transform the Interactive Reporting data set into Adobe Flash based graphics. They are snapshots of real time data graphically embedded in the Dashboard section. When you need to present compact data driven visuals, use these objects to develop the solution.

Interactive Reporting offers four types of gauges, six types of (Live) Charts, and a Slider control.

Available gauges are:

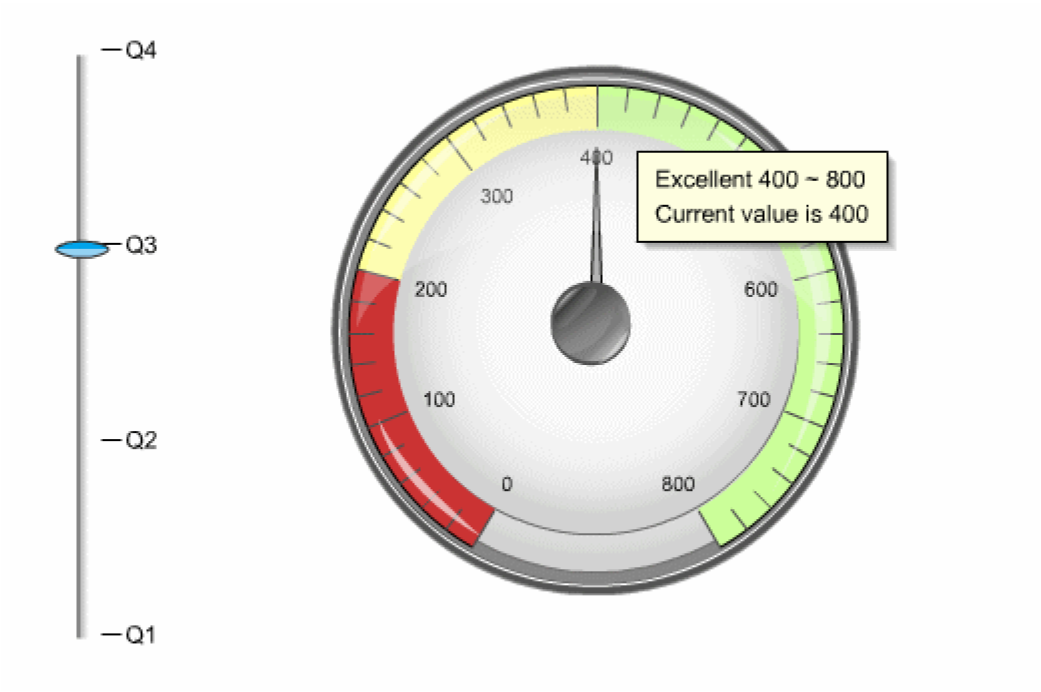
- Speedometer
- Thermometer
- Bullet
- Traffic Light

Available (Live) Charts are:

- (Live) Bar Chart (in cluster, stack or 100% formats)
- (Live) Block Chart (in pyramid or cone formats)
- (Live) Funnel Chart
- (Live) Line Chart (in line or area formats)
- (Live) Pie Chart (in pie or donut formats)
- (Live) Radar Chart

The XML template drives the overall appearance of gauges and (Live) Charts. The template provides a consistent look, and each object inherits the properties specifically defined for it by the template. It is provided in the installed version of Interactive Reporting. Many properties of each object can be changed without editing the template directly. For example you can select a gauge theme, choose the type of a (Live) Chart (cluster, stack or 100% stack for a (Live) Bar chart), or define which labels and values are rendered.

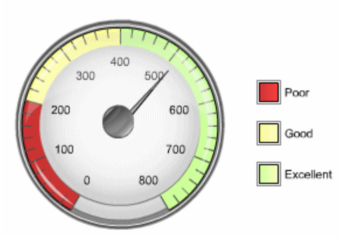
Below is an example of a Speedometer gauge (right) which has been associated with a Slider (left).

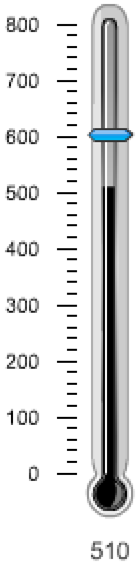
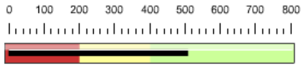
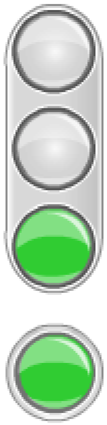


Note: Adobe Flash must already be installed before you can display the gauges and (Live) Charts in Interactive Reporting. See the *epm_install_start_here.pdf* for more information.

Gauges

Gauges simulate familiar measuring devices, such as a speedometer or thermometer. They illustrate at a glance the performance relationships between actual and target values. Gauges can be embedded as stand-alone graphics, or made dynamic by association with a Slider. The Slider is a graphical control which drives the display of data by category in the gauge.

Gauge	Description
	**Speedometer**—Features a speedometer like object. The radial design provides an easily to recognize mechanism for measuring values. The Speedometer gauge can be shown in 360 degree, 180 degree, 90 degree left or 90 degree formats.

Gauge	Description
 800 700 600 500 400 300 200 100 0 510	**Thermometer**—Illustrates values in a thermometer like graphic. This Thermometer gauge is particularly useful for comparing levels based on a certain “temperature” level.
	**Bullet**—Features a linear shape that is easily scalable to a smaller space in the dashboard.
	**Traffic Light**—Illustrates the “safety” or health of a specific indicator in a single or multiple traffic light like graphic.

Adding Gauges to the Dashboard

One or multiple gauges can be added to the Dashboard section. For each gauge, you select actual and target fact values from the same section.

➤ To add a gauge to the Dashboard:

1 Open a dashboard section in design mode.

2 From the **Elements**, expand **Gauges**.

3 **Select a gauge and drag it to the Content pane.**

Available gauges include:

- Bullet
- Speedometer
- Thermometer
- Traffic Light

The gauge object is added to the Content pane.

4 **Select the gauge object to launch the data layout.**

5 **In the Elements pane, expand the Results or Table section.**

6 **Drag a fact column to the (actual) Fact pane of the data layout.**

7 **To apply a data function to the actual value in step 6, select the value in the data layout and on the shortcut menu, choose Data Function.**

8 **Select the data function from the sub-menu.**

Available data functions are:

- Sum (default function)
- Average
- Count
- Maximum
- Minimum

9 **Drag a fact column to the Target pane of the data layout.**

The Traffic Light gauge does not include a target range. As a result, no Target pane is available when the Traffic Light object is selected.

10 **To apply a data function to the target value in step 9, select the target value and on the shortcut menu, choose Data Function.**

11 **Select the data function from the sub-menu.**

Available data functions are:

- Sum
- Average
- Count
- Maximum
- Minimum

12 **To modify gauge properties, select the gauge object and on the shortcut menu, select Properties.**

For more information, see “Gauge Properties ” on page 38.

13 **Associate the gauge with a Slider.**

For more information, see Adding a Slider to the Dashboard.

14 **To view the gauge, switch to the Dashboard section in run mode.**

Gauge Properties

Use the Gauge Properties dialog box to set the display options for the gauges.

General Properties	Description
Theme	Refers to the parts of the gauge that can be altered to change the look without changing its functionality. Available options: ● realistic—Renders the graphic with a three-dimensional appearance. ● simplistic—Renders the gauge with a one dimensional appearance. If a gauge is scaled on the smaller side, use the simplistic theme.
Subtype	Select the degrees of orientation of the gauge. Individual gauge sub-types include: **Speedometer** ● Speedo_Full—360 degree orientation ● Speedo_180—180 degree orientation ● Speedo_90left—90 degree left orientation ● Speedo_90right—90 degree right orientation **Thermometer** ● Vertical_Thermometer—Vertical thermometer orientation ● Horizontal_Thermometer—Horizontal thermometer orientation **Bullet** ● Vertical_Bullet—Vertical bullet orientation ● Horizontal_Thermometer—Horizontal bullet orientation **Traffic Light** ● Vertical_Traffic_Light—Vertical Traffic Light orientation ● Horizontal_Traffic_Light—Horizontal Traffic Light orientation ● Single_Traffic_Light

Scale Properties	Description
Auto	Enable the calculation of the lower and upper values scale of the data set automatically. The scale is recalculated automatically when the data set is refreshed. To specify a minimum and maximum scale manually, disable this field.
Min.	Specify the lowest value of the numeric scale from the actual data set in this field. This option is disabled if Auto is enabled.
Minor Interval	Specify the minor interval unit placed on the scale of gauge. For example, if you select 10, tick marks are placed at every tenth minor interval. This option is disabled if Auto is enabled.

Scale Properties	Description
Max.	Specify the highest value of the numeric scale from the actual data set in this field. This option is disabled if Auto is enabled.
Major Interval	Specify the major interval unit placed on the scale of gauge. For example, if you select 20, tick marks are placed at every twentieth major interval. This option is disabled if Auto is enabled.

Show Properties	Description
Legend	Enables the display of the gauge legend.
Tickmarks	Enables the display of short lines extending from the label to the value.
Value	Enables the display of the featured actual value beneath the gauge.
Scale	Enables the display of the numeric ranges in the gauge.

Color Range Properties	Description
Min	For each color range, specify the minimum value from which to apply a color. Only the first minimum color range value can be blank.
Max	For each color range, specify the maximum value to which a color is applied. Only the last maximum color range value can be blank.
Color	For each color range to be applied, double-click the Color field and select a color from the Color Palette.
Tooltips	For each color range, specify a description of the range. For example, you might assign “poor” for the color red, or “excellent” for the color green. The tool tip is specific to the color range. The default tool tip description is “Current value is xxx”.
Add	Adds another color range.
X	Removes a color range.

Color Range Behavior

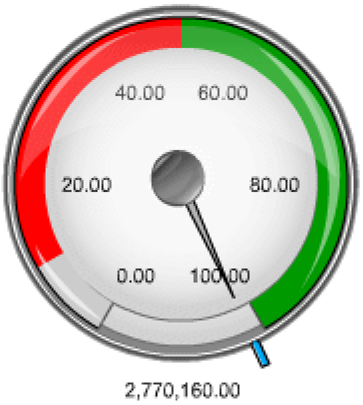
Color ranges are not rendered the same way for two identical gauges with approximate starting and ending values if one gauge has one or more blank color range buckets, and the other has all explicitly defined.

For example in the first sample, the numeric scale for the color ranges is defined at:

Table 9 Speedometer 1 Color Range

Min	Max	Color
10	50	Red
50	100	Green

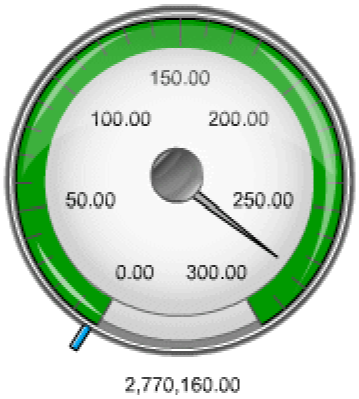
The result is:



In the second example, the numeric scale for the color ranges is defined at:

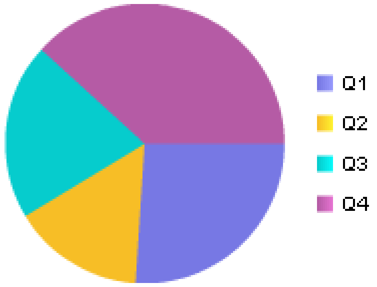
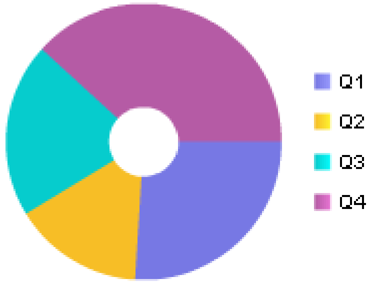
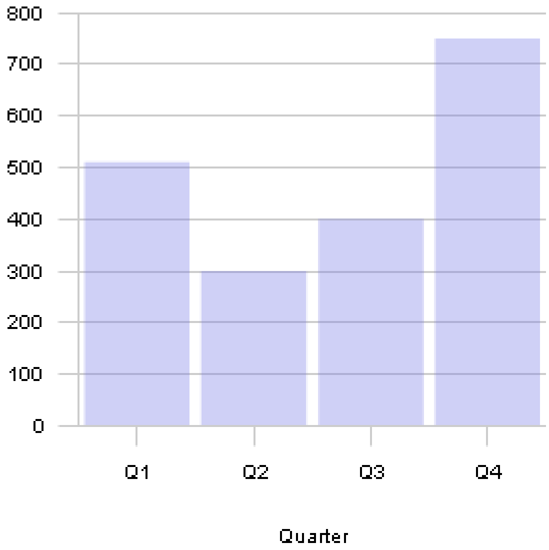
Table 10 Speedometer 2 Color Range

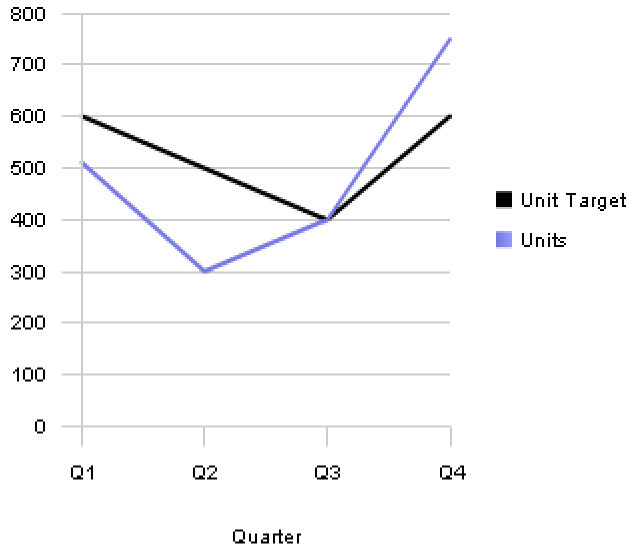
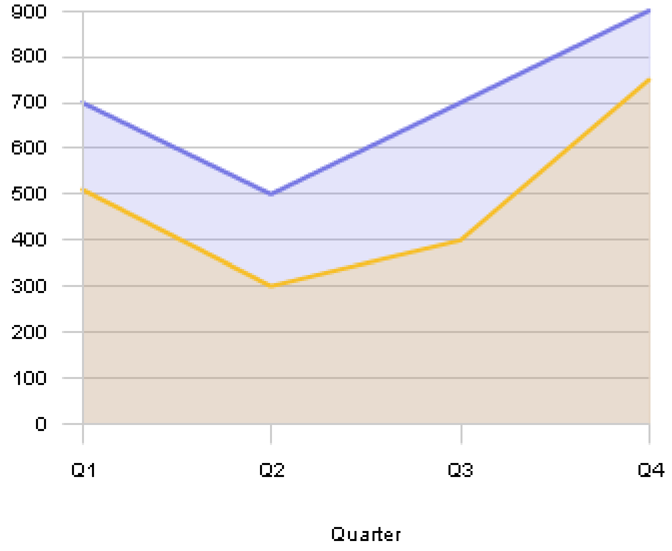
Min	Max	Color
	50	Red
50		Green

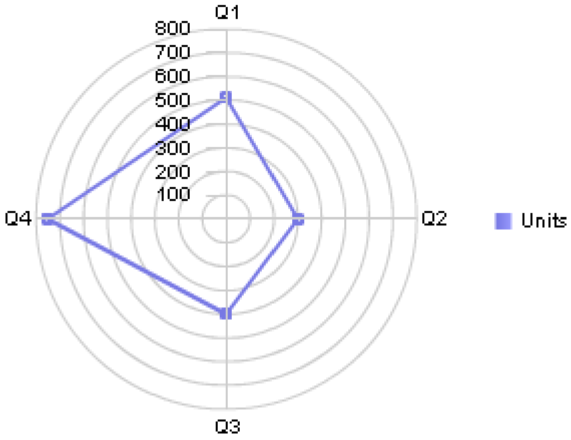
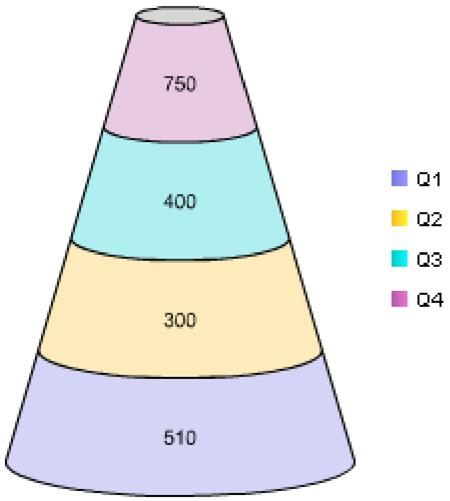


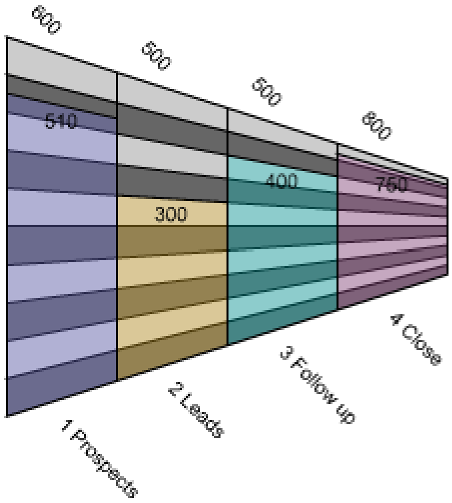
(Live) Charts

Use (Live) Charts to present your data in a Flash based animated chart format. (Live) Charts consists of six chart types, each or which can be customized to show different degrees of information. Like gauges, they can be driven by a slider control. Interactive Reporting (Live) charts include:

Name	(Live) Chart Type	Description
Pie	(Live) Pie Chart in 2-D format:  (Live) Pie Chart in donut format:  	(Live) Pie charts display data as shares or “slices” to a whole in a pie-shaped graphic. These types of charts are useful for relative comparisons within the same data series, for example if you need to compare advertising costs of different brands. (Live) Pie charts can be displayed in a standard pie format, or in donut format. Charts can be “exploded” such that slices are moved away from the center of the chart.
Bar	(Live) Bar Chart  	(Live) Bar charts displays data values within the same category by bar length. This type of chart is useful for comparing data series within a category.

Name	(Live) Chart Type	Description
Line	(Live) Line Chart in line format:  (Live) Line Chart in area format:  	(Live) Line charts displays items or a series in one or more categories at equal intervals. They are effective for comparing highs and lows in a continuum. Values are depicted by the height of a point determined in the y-axis, and categories are represented on the x-axis. (Live) Line charts are also effective in showing numeric increases or decreases within the same category. The (Live) Line chart can be displayed in the traditional or area format.

Name	(Live) Chart Type	Description
Radar		(Live) Radar charts display data values in relation to a center point. This chart is effective for demonstrating variances between multiple components associated with a single value. A point nearer to the center on an axis shows a low value, and a point closer to the edge show a higher value. Each data point is displayed as a marker, and you can plot data along multiple axes.
Block	(Live) Block Chart in cone format:  (Live) Block in Pyramid format	(Live) Block charts consists of a cone shape or pyramid shaped chart. Like (Live) Bar charts, (Live) Block charts are useful for comparing values using a common scale, where the values need to be rendered hierarchically. (Live) Block charts are best used with a small data set since they taper at the end.

Name	(Live) Chart Type	Description
Funnel		(Live) Funnel charts illustrate discreet components within a larger context. The height of the bar represents how close the actual value (Fact) is to a target values. For example, if the target value is 100, and the actual value is 50, the bar is drawn half way. (Live) Funnel charts are traditionally used to demonstrate phases in the sales pipeline (for example, prospects, qualified leads, tested customer requirements, proposals sent and negotiations to close).

Adding (Live) Charts to the Dashboard

➤ To add a (Live) Chart:

1 Open a Dashboard section in design mode.

2 From the **Elements Catalog**, expand **Live Charts.**

3 Select a (Live) Chart and drag it to the Content pane.

Available (Live) Charts include:

- (Live) Bar Chart
- (Live) Line Chart
- (Live) Pie Chart
- (Live) Radar Chart
- (Live) Block Chart
- (Live) Funnel Chart

A (Live) chart object is added to the Contents pane.

4 Select the (Live) Chart object and select **Data Layout.**

5 In the **Elements** pane, expand the **Results** or **Table** section.

6 Drag a fact column to the (actual) **Fact** pane of the data layout.

7 To apply a data function to the Fact value in step 6, select the Fact value in the data layout and on the shortcut menu, choose **Data Function.**

8 Select the data function from the sub-menu.

Available data functions are:

- Sum
- Average
- Count
- Maximum
- Minimum

9 **Drag a Category column to the XAxis, Wedge, or Slice pane of the data layout.**

Depending on the type of (Live) Chart, two or more data layout panes are displayed. The values that can be in the data layout panes are based on the chart type:

- (Live) Bar Chart—Accepts Fact values and X-Axis values.
- (Live) Line Chart—Accepts Fact values, Target Fact values and X-Axis values.
- (Live) Pie Chart—Accepts Fact values, and Slice values.
- (Live) Radar Chart—Accepts Fact values and Axis values.
- (Live) Block Chart—Accepts Fact values and Wedge values.
- (Live) Funnel Chart—Accepts Fact values, Target values, and X axis values.

10 **To apply a data function to category (numeric only) value in step 9, select the value and on the shortcut menu, choose Data Function.**

11 **Select the data function from the sub-menu.**

Available data functions are:

- Sum
- Average
- Count
- Maximum
- Minimum

12 **Optional for (Live) Line Charts only, drag a fact column to Target Fact pane of the data layout.**

13 **To change (Live) Chart properties, select the (Live) Chart object and on the shortcut menu, select Properties.**

For more information about chart specific properties, see (Live) Chart Properties.

14 **Optional: Associate the (Live) Chart with a Slider.**

For more information, see: Adding a Slider to the Dashboard.

15 **To view the (Live) Chart, switch to the Dashboard section in run mode.**

(Live) Chart Properties

Use the (Live) Chart Properties dialog box to set the display options for the following (Live) Charts:

- (Live) Bar Chart Properties
- (Live) Line Chart Properties
- (Live) Pie Chart Properties
- (Live) Radar Chart Properties
- (Live) Block Chart Properties
- (Live) Funnel Chart Properties

Note: The Live Chart color scheme is derived from the color scheme defined in the Default Fonts and Styles dialog box.

(Live) Bar Chart Properties

Use the (Live) Bar Chart Properties dialog box to define specific properties associated with a (Live) Bar Chart.

Table 11 (Live) Bar Chart General Properties

General Properties	Description
Type	Specify the bar chart type. Available options include: ● Cluster—Show data series grouped together on one axis. ● Stack—Shows data series arranged in a layered format. Stacked bar charts offer similar complexity to clustered bar charts by adding component value items within chart bars or areas. By stacking items and assigning a different color to each item, you can effectively display trends among comparable or related items, or visually emphasize a sum of several indicators. ● 100% Stack—Shows the data series arranged in a stacked format as a percentage of the whole total. This format enables you to compare percentage across different data series.
Effect	Select either a 2-D, 3-D, or gradient effect. A 2D or 3D effect refers to the perspective of the rendered chart. A gradient defines the direction of the blend pattern from one color to another, and simulates depths of color from light to dark. Available options are: ● 2-D—The 2-D effect is flat and shows values only on the horizontal and vertical axes (X and Y axis). ● 3-D—The 3-D effect shows rotation and depth in addition to the horizontal and vertical axes. ● gradient-Vert—Sets a top to bottom gradient. ● gradient-Hort—Sets a left to right gradient.
Orientation	Specify the direction of the bars. Available options are vertical or horizontal.
Show Legend	Enables the displays of the chart legend.
Show Bar Values	Enables the display of bar values.

Table 12 (Live) Bar Chart Category Axis Properties

Category Axis Property	Description
Title	Enables the display of the category axis title.

Category Axis Property	Description
Show Labels	Enables the display of category axis labels.

Table 13 (Live) Bar Chart Fact Axis Properties

Fact Axis Property	Description
Title	Enables the display of the fact axis title.
Show Labels	Enables the display of fact labels.
Auto	Enable to calculate the lower and upper values scale of the data set automatically. The scale is recalculated automatically when the data set is refreshed. To specify a minimum and maximum scale manually, disable this field.
Min:	Enter the minimum data value on the values scale. This option is disabled if Auto is enabled.
Max:	Enter the maximum data value on the value scale. This option is disabled if Auto is enabled.

(Live) Line Chart Properties

Use the (Live(Line Chart Properties dialog box to define specific properties associated with a (Live) Line Chart or Area Chart.

Table 14 (Live) Line Chart General Properties

General Properties	Description
Type	Select the format for the (Live) Line Chart. Available options include: ● Line—Displays items as data points connected by a line. or a series in one or more categories at equal intervals. Line charts are effective for comparing highs and lows in a continuum. Values are depicted by the height of a data point determined in the y-axis, and categories are represented on the x-axis. ● Area—Displays values as data points connected by a line. The area below the line is filled. Values are represented by the height of the point as measured by the y-axis. Category labels are displayed on the x-axis. Area charts are typically used to compare values over time.
Effect	Select either a 2-D, or gradient effect. A 2-D effect refers to the perspective of the rendered chart. A gradient defines the direction of the blend pattern from one color to another, and simulates depths of color from light to dark. Available options are: ● 2-D—Two dimensional refers to the actual dimensions of the graphic. The 2-D effect is flat and shows values on the horizontal and vertical axis (X and Y axis). ● gradient-Vert—Sets a top to bottom gradient. ● gradient-Hort—Sets a left to right gradient.
Marker Style	Select the style of a marker. A marker depicts a data point in a cell. Valid options are: ● Circle ● Square ● Triangle ● Diamond ● None

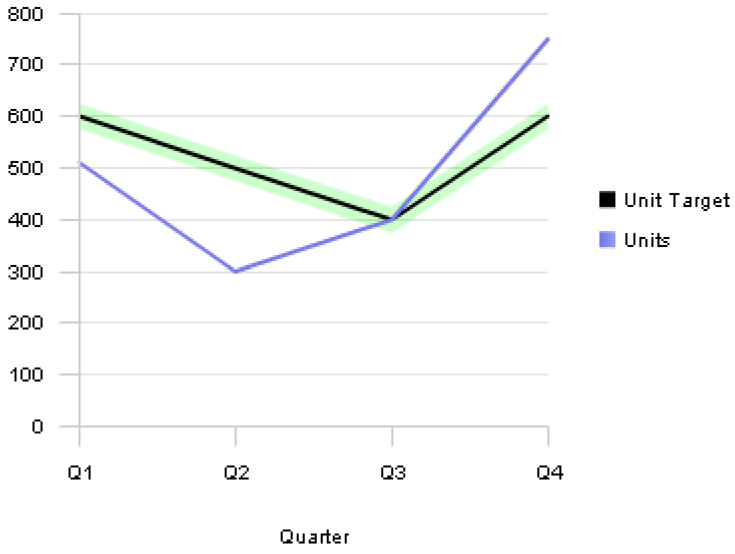
General Properties	Description
Show Title	Enables the display of the chart title.
Show Legend	Enables the display of the chart legend.
Target Range	(Live) Line charts can include a colored band around the target line to indicate the absolute or percentage distance from the target markers. A unique color can also be associated with the area above or below the specified band. From left to right, the first color box corresponds to the upper range, the second color box corresponds to the middle range, and the third color box corresponds to the lower range. By default the upper and lower ranges have a transparent color. To select an alternate color for a range, double click the box and select a color from the palette. This option is not available for a (Live) Line Chart in area chart format.
(% or Absolute Deviation)	For the target range above, select one of the following options: ● % of Deviation—Select a percentage amount in the first box and % in the second box. The percentage is added to the data point. ● Absolute—Select a real number in the first box and Absolute in the second box. In this example "Units" is the actual range and shown in blue, "Unit Target" is the target range and shown in black. The target range band is shown in light green, and the absolute deviation is 25 (shown in the green band): 

Table 15 (Live) Line Chart Category Axis Properties

Category Axis Properties	Description
Title	Enables the display of the category axis title.
Show Labels	Enables the display of category axis labels.

Table 16 (Live) Line Chart Fact Axis Properties

Fact Axis Property	Description
Title	Enables the display of the fact axis title.
Show Labels	Enables the display of fact labels.
Auto	Enable to calculate the lower and upper values scale of the data set automatically. The scale is recalculated automatically when the data set is refreshed. To specify a minimum and maximum scale manually, disable this field.
Min:	Enter the minimum data value on the values scale. This option is disabled if Auto is enabled.
Max:	Enter the maximum data value on the value scale. This option is disabled if Auto is enabled.

(Live) Pie Chart Properties

Use the (Live) Pie Chart Properties dialog box to define specific properties associated with a (Live) Pie Chart.

Table 17 (Live) Pie Chart General Properties

General Properties	Description
Type	Select the (Live) Chart format. Available options include: ● Pie—Displays pieces (slices) of the pie drawn to represent the relative value of a measurable item to the whole. Only one category (data series) can be plotted in a Pie Chart. ● Donut—Drawn like a (Live) Pie Chart, the donut chart format does not include a center hole.
Effect	Specify the effect associated with the chart. A 2-D or 3-D effect refers to the perspective of the rendered chart. Available options are: ● 2-D—The 2-D effect is flat. ● 3-D—The 3-D effect shows rotation and depth.
Show Legend	Enables the display of the chart legend.
Show Label	Enables the display of value labels.
Explode	Pulls individual slices from the center.

(Live) Radar Chart Properties

Use the (Live) Radar Chart Properties dialog box to define specific properties associated with a (Live) Radar Chart.

Table 18 (Live) Radar Chart General Properties

General Property	Description
Effect	Specify the effect for the area contained within the data value points. Available options are: ● Outline—The area shows only the lines connecting the data value points. ● Fill—The area contained within the data value points is filled with color.
Show Legend	Enables the display of the (Live) Radar Chart legend.

Table 19 (Live) Radar Chart Category Property

Category Axis Property	Description
Show Labels	Enables the display of category axis labels.

Table 20 (Live) Radar Chart Fact Axis Properties

Fact Axis Properties	Description
Title	Enables the display of the fact axis title.
Show Labels	Enables the display of fact labels.
Auto	Enable to calculate the lower and upper values scale of the data set automatically. To specify a minimum and maximum scale manually, disable this field.
Min:	Enter the minimum data value on the values scale. This option is disabled if Auto is enabled.
Max:	Enter the maximum data value on the values scale. This option is disabled if Auto is enabled.

(Live) Block Chart Properties

Use the (Live) Block Chart Properties dialog box to define specific properties associated with a (Live) Block Chart.

Table 21 (Live) Block Chart General Properties

General Properties	Description
Type	Select the format of the (Live) Block chart. Available options include: ● Pyramid—A pyramid shaped chart that shows the percentage of each data value to the whole with the smallest value at the top and the largest at the bottom. ● Cone—A conical shaped chart that show the percentage of each data value to the whole with the smallest value at the top and the largest at the bottom.
Effect	Select the effect associated with the chart. A 2-D or 3-D effect refers to the perspective of the rendered chart. Available options are: ● 2-D—The 2-D effect is flat and shows values only on the horizontal and vertical axes (X and Y axes). ● 3-D—The 3-D effect shows rotation and depth in addition to values on the horizontal and vertical axes.

General Properties	Description
Orientation	Select the position of the tapered end of the block chart. Available positions are: ● Up ● Down ● Left ● Right
Show Legend	Enables the display of the chart legend.
Show Label	Enables the display of value labels.
Explode	Pushes out individual values.

(Live) Funnel Chart Properties

Use the (Live) Funnel Chart Properties dialog box to define specific properties associated with a (Live) Funnel Chart.

Table 22 (Live) Funnel Chart General Properties

General Properties	Description
Effect	Select the effect associated with the chart. A 2-D or 3-D effect refers to the perspective of the rendered chart. Available options are: ● 2-D—The 2-D effect is flat and shows values only on the horizontal and vertical axes (X axis and Y axis). ● 3-D—The 3-D effect shows rotation and depth in addition to the horizontal and vertical axes.
Orientation	Select the position of the tapered end of the block chart. Available positions are: ● Up ● Down ● Left ● Right
Show Legend	Enables the display of the chart legend.
Show Bar Values	Enables the display of bar values.

Table 23 (Live) Funnel Chart Category Axis Properties

Category Axis Properties	Description
Title	Enables the display of the category axis title.
Show Labels	Enables the display of category axis labels.

Table 24 (Live) Funnel Chart Fact Axis Properties

Fact Axis Properties	Description
Title	Enables the display of the fact axis title.
Show Labels	Enables the display of fact axis labels.
Auto	Enable to calculate the lower and upper values scale of the data set automatically. The scale is recalculated automatically when the data set is refreshed. To specify a minimum and maximum scale manually, disable this field.
Min	Enter the minimum data value on the values scale. This option is disabled if Auto is enabled.
Max	Enter the maximum data value on the values scale. This option is disabled if Auto is enabled.

Sliders

A slider is a control that drives the data on a associated gauge or (Live) Chart. It is the mechanism that provides the real interactivity between you and the gauge or (Live) Chart object. As you move along the dimension labels on the slider, the gauge or (Live) Chart snaps onto the corresponding values. Slider dimensions must reference the same data set used by the gauge or (Live) Chart. A Slider can control multiple gauges or (Live) Charts in the same Dashboard section. Like other Dashboard controls, Slider properties can be modified.

Adding a Slider to the Dashboard

➤ To add a slider:

1 **Open a dashboard section in design mode.**

2 **From the Elements pane, expand Controls**

3 **Select the Slider control and drag it to the Content pane.**

A slider object is added to the Contents pane.

4 **Select the Slider object.**

The Category pane of the data layout is displayed.

5 **From the Elements pane, expand the Results or Table section.**

6 **Select a dimension (non-fact) item and drag it to the Category pane of the data layout**

7 **Select the slider and on the shortcut menu select Properties.**

The Properties dialog box is displayed.

8 **Select the Association tab.**

The Association dialog box is displayed.

9 **In the Available, pane, select a gauge or (Live) Chart and click ⏩ to add it to the Selected pane.**

10 **Select OK.**

To remove the association between the slider and the gauge, select the gauge in the Selected pane and click ⇦.

11 **Select the Slider tab.**

12 **From the Theme drop-down, select the theme of the Slider.**

Available options are: realistic or simplistic.

13 **From the Subtype drop-down, select the orientation of the Slider.**

Available options are: vertical or horizontal.

14 **Click OK.**

Slider Properties

Association Properties

Use the Association dialog box to link a gauge or (Live) Chart with the slider. Multiple gauges or (Live) Charts can be associated with one slider.

➤ To associate a gauge with a slider:

1 **In design mode, select the slider and on the shortcut menu, select Properties.**

The Properties dialog box is displayed.

2 **Select the Association tab.**

3 **In the Available, pane, select a gauge and click ⏩ to add it to the Selected pane.**

4 **Click OK.**

Table 25 Association Properties

Association Properties	Description
Available	Lists the gauges and (Live) Charts available to be associated with the selected Slider.
Selected	Lists the gauges and () Charts associated with the selected Slider.

Slider Properties

Use Slider Properties dialog box to set the appearance of the Slider control.

➤ To define properties of the slider:

1 **In design mode, select the Slider and on the shortcut menu, select Properties.**

The Properties dialog box is displayed.

2 **Select the of the slider.**

Available options are

- realistic—Renders the graphic with a three-dimensional appearance.
- simplistic—Renders the gauge with a one dimensional appearance

3 **Select the orientation of the slider from the Subtype field.**

Available options are:

- Vertical
- Horizontal

Table 26 Slider Properties

Slider Properties	Description
Theme	Refers to the parts of the slider that can be altered to change the look without changing its functionality. Available options: ● realistic–Renders the graphic with a three-dimensional appearance. ● simplistic–Renders the gauge with a one dimensional appearance. If a gauge is scaled on the smaller side, use the simplistic theme.
Subtype	Refers to the orientation of the slider. Available options are: ● Vertical_Slider ● Horizontal_Slider

Setting Dashboard Properties

You use the Properties dialog box to set properties for a dashboard section or for specific objects within a dashboard section. Many dashboard objects have unique properties. For example, radio buttons (option buttons) have Radio Group properties, and list boxes have Multiple Selection properties. Tab-order properties apply to all sections and are accessible in the Properties dialog boxes for the dashboard section and for individual objects.

➤ To set properties for objects and dashboard sections:

1 **Select an object on the content area or a dashboard section.**

2 **Right-click and select Properties, or select Dashboard, then Properties.**

Properties is displayed. The active tab depends on the selection made prior to invoking Properties.

3 **Set the properties for the object or dashboard section, by using the tabs.**

Available properties:

- **Alignment**—Horizontal and vertical alignment, text wrapping, and rotation
- **Border and Background**—Border color, width, style, and shadow and background color and pattern

- **Font**—Family, style, size, effects (underline, overline, double overline), and color
- **Object**—Name, title, visible, enable (control objects only), locked, scroll bars always shown, and auto-size; for embedded sections, view-only, active, or hyperlink
- **Picture**—File name, size, and effects—for dashboard background and graphic object pictures
- **Tab Order**—Object path that end users follow when Tab is pressed in Run mode
- **Accessibility**—User-defined and auto generated descriptive 508 text for each embedded section or graphic
- **Values**—User-defined values that populate list box, drop-down lists, or text box controls

4 Click **OK** to apply the settings and close **Properties.**

Using Design Tools

Interactive Reporting Studio gives you complete control of your dashboard section setup and provides layout and navigation tools that assist you in designing effective, high quality custom applications.

Layout Tools:

- Using Design Guides
- Using Grids
- Using Rulers

Layout Tools

Layout tools are available from the dashboard menu or the dashboard section toolbar.

Using Design Guides

Design guides are horizontal and vertical lines that you place in your report to help you align objects. Design guides are similar to grids in that objects automatically snap to align to the guides.

If rulers are visible, you can click a ruler and drag one or more design guides from the horizontal or vertical ruler.

➤ To display or not display design guides, select **Dashboard**, then **Design Guides**.

If guides were visible, they are now invisible. If guides were not visible, they are now visible. If guides are visible, a check mark is displayed before Design Guides.

Using Grids

Interactive Reporting Studio and Interactive Reporting Web Client provide a layout grid that automatically snaps all objects to the closest grid point.

➤ To display or not display the grid, select **Dashboard** , then **Grid**

If the grid was visible, it is now invisible. If the grid was not visible, it is now visible. If the grid is visible, a check mark is displayed before Grid.

Using Rulers

Horizontal and vertical rulers help you align items based on units of measure—inches, centimeters, and pixels. You select a unit by clicking in (measure indicator) at the intersection of the top and left rulers.

➤ To display and not display the ruler, select **Dashboard** , then **Ruler**.

If the ruler was visible, it is now invisible. If the ruler was not visible, it is not visible. If the ruler is visible, a check is displayed before Ruler.

Using the Dashboard Section Toolbar

The dashboard section toolbar provides icons allow you to maneuver multiple dashboard objects.

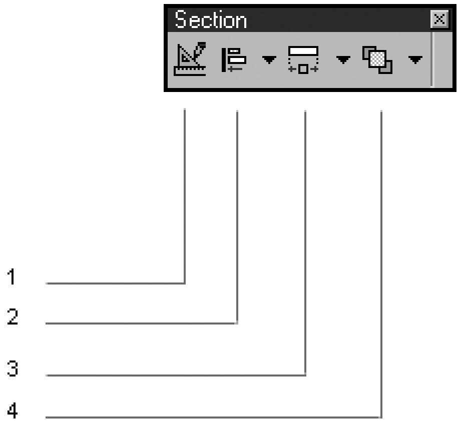


Table 27 Dashboard Section Toolbar

Numbered Item	Button Name	Description
1	Design/Run	Toggle between Design and Run modes

Numbered Item	Button Name	Description
2	Align	Aligns multiple selected objects to the first selected object (Select the first object, press and hold Ctrl, and select the remaining objects. Click ▼, and select an alignment option: left, center, right, top, middle, or bottom)
3	Make Same Size	Sizes multiple selected objects to the size of the first selected object. (Select the first object, press and hold Ctrl, and select the remaining objects. Click ▼, and select a sizing option: width, height, or both)
4	Layer	Stacks one object relative to other objects: bring to front, send to back, bring forward, and send backward (Use this feature to layer multiple objects so that only the sections of the objects that you want visible are displayed)

Using the Navigation Toolbar

You use the navigation toolbar to return to a dashboard section from another section when Sections, section title bar, toolbars, and menus are turned off.

The navigation toolbar is hidden by default, but you can use scripts to enable it. When activated, the toolbar is available in all sections and includes the ⇦ (Back), ⇨ (Forward), and ▦ (Dashboard Home—See “Setting Dashboard Home Sections” on page 23).

Use these scripts to work with the navigation toolbar.

This script activates the navigation toolbar.

```
//Syntax for turning on Navigation toolbar
Toolbars["Navigation"].Visible=true;
```

This script activates all toolbars except the navigation toolbar.

```
//Syntax for turning on all toolbars except the Navigation toolbar

j=Toolbars.Count

for (i=1; i<=j; i++) {
     if (Toolbars[i].Name != "Navigation") {Toolbars[i].Visible=true}
}
```

This script turns off all toolbars.

```
//Syntax for turning off all toolbars
j=Toolbars.Count

for (i=1; i<=j; i++) {
     Toolbars[i].Visible=false
}
```

2 Working with the Interactive Reporting Object Model

In This Chapter

The Object Model

The object model is the cornerstone for scripting customized interfaces (dashboards). The object model and Script Editor provide access to all levels of Interactive Reporting.

The object model is a hierarchical representation of Interactive Reporting and the Interactive Reporting Web Client objects and the actions (methods) and attributes (properties) that are used to manipulate the objects.

Objects include applications, documents, sections, limits, connections, graphics, controls, catalog items, topics, request lines, results columns, chart labels, pivot-side labels, facts, menu bars, status bars, toolbars, and so on.

Methods include create, activate, open, close, save, add, copy, remove, process, export, recalculate, and so on. For example, data-results objects (database-query results or tables that contain results data) have a recalculate method, which refreshes (recalculates) data based on updated parameters.

Properties include object names, values, alignments, colors, and so on. You can view properties or set (modify) the values of properties. For example, all graphics objects have a visible property —if set to true, an object is visible, and, if set to false, an object is invisible.

Table 28 Object Model Terminology

Term	Definition	Example	Interactive Reporting Example
Object	Items perceived as entities	Tree, leaf, fruit	Application, Section, Document
Method	Actions that are executed when an object receives messages	Grow, bear fruit, drop leaves	Activate, Copy, Add

Term	Definition	Example	Interactive Reporting Example
Property	Qualities or distinctive features (attributes)	Name, color, growing pattern	Active, Visible, Type
Collection	Groups of objects	Grove	Documents
Constant	Values that do not change or vary	Number	Constants

Typically, the object model is manipulated by JavaScript from inside dashboard sections and used to build self-contained analytic applications. On Windows systems, the object model is also accessible through the automation interfaces (OLE Automation) that enable Interactive Reporting to be controlled by external applications that can make OLE Automation calls (such as Excel or, VB).

Interactive Reporting Events

Custom applications (that is, dashboard sections) that are developed using Interactive Reporting and Interactive Reporting Web Client are event-driven. Events are actions; such as, clicking a button or opening a document, that are recognized by Interactive Reporting documents, or sections, or by dashboard objects. When an event occurs, Interactive Reporting and Interactive Reporting Web Client invoke the script attached to the event. The order in which events are executed depends upon the order of user actions.

You determine how events respond to actions by attaching scripts to events. For example, assume that you want a particular action to occur when a particular button is clicked. You attach a script that defines the response to the `OnClick` event associated with the button. When the button is clicked, Interactive Reporting and Interactive Reporting Web Client invoke the script.

Note: Since Interactive Reporting and Interactive Reporting Web Client Release 6.0, JavaScript has been used as the scripting language. Documents scripts created using the Brio scripting language are converted to JavaScript when documents are first opened.

Predefined event levels and the items with which they are associated:

- **Object-level events**—Dashboard objects
- **Section-level events**—Dashboard sections
- **Document-level events**—Interactive Reporting and Interactive Reporting Web Client documents

Object-Level Events

Note: All events starting with `OnClient` are executed in the client browser, not on the server. See “Client-Side JavaScript” on page 298.

Table 29 Predefined Object-Level Events Associated with Dashboard Objects

Event	Associated Object	Action that Invokes the Event
`OnCellDoubleClick`	Dashboard sections within Interactive Reporting or Interactive Reporting Web Client that contain pivot Embedded Section Objects (ESOs) in Active mode **Note:** Double-clicking label values does not initiate `OnCellDoubleClick`.	Double-clicking fact cells or using the`OnCellDoubleClick ()` method **Note:** Executing `OnCellDoubleClick` does not corrupt documents, but actions invoked by `OnCellDoubleClick` may corrupt documents. Whether the corrupt state persists, within the document or application, depends upon which action is invoked.
`OnClick`	Sections: Hyperlinked embedded sections (if not View-only or Active) Graphics: Line, horizontal line, vertical line, rectangle, round rectangle, oval, text label, and picture Controls: Command button, radio button, check box, list box, drop-down list, and text box	Clicking sections, graphics, or controls. The OnClick handler also supports the optional bqoEvent parameter, which provides access to the mouse cursor position relative to the Dashboard control. The bqoEvent parameter contains two properties: ClickX and ClickY. When the event occurs, the event handler has access to the event related information needed to process it. For example, the event might require the position of the mouse cursor for a picture OnClick event, or information about which table column was clicked for a table embedded section object OnClick event.
`OnDoubleClick`	List box	Double-clicking list-box values
`OnSelection`	List box and drop-down list	Selecting list-box or drop-down-list values
`OnChange`	Text box (not available for EPM Workspace)	Changing data in text boxes
`OnEnter`	Text box	Entering text boxes
OnExit	Text box	Leaving text boxes
`OnRowDoubleClick`	Active embedded results or table sections (if not View-only or Hyperlinked)	Double-clicking rows
`OnClientClick`	Command button, radio button, check box, list box, text box, and drop-down list	Clicking sections or controls
`OnClientDoubleClick`	List box	Double-clicking list-box values
`OnClientEnter`	Text box	Entering text boxes
`OnClientExit`	Text box	Leaving text boxes

Active Section-Level Events

Active section-level events are associated with dashboard sections.

Actions that invoke predefined section-level events:

- **OnActivate**—Entering a dashboard section
- **OnDeactivate**—Existing a dashboard section

Document-Level Events

Document-level events are associated with Interactive Reporting and Interactive Reporting Web Client documents.

Actions that invoke predefined document-level events:

- **OnStartUp**—Opening an Interactive Reporting document
- **OnShutDown**—Closing an Interactive Reporting document
- **OnPreProcess**—Before a query is processed
- **OnPostProcess**—After a query is processed

The execution of document events can be enabled or disabled by using options in Script Editor programmatically through the object model.

Note: The Save and Save As commands in Interactive Reporting Studio, Interactive Reporting Web Client, and EPM Workspace do not execute the document shutdown scripts. The scripts execute only when documents are closed.

Caution! `OnShutDown` events execute before the prompts of the Save dialog box.

Associating Document-Level Events

By default, the desktop and Interactive Reporting Web Client applications invoke the scripts attached to document-level events when the events are triggered. For example, the `OnStartup` document event is triggered when Interactive Reporting documents are opened.

On the other hand, Hyperion Scheduler does not invoke a script attached to a document-level event until the user sets the document-level event for the Interactive Reporting document manually. Settings for document-level events can be set based on how a document is to be deployed (on the desktop, or on EPM Workspace, or in Interactive Reporting Web Client).

➤ To associate document-level events:

1 Select **File** , then **Document Scripts.**

Script Editor is displayed. Document is selected by default in the Object drop-down list, and `OnStartup` is selected by default in the Event Trigger drop-down list.

2 From **Event Trigger**, select a document-level event to be associated with the Interactive Reporting document.

3 In **Enable For**, select the type of document to associate with the event.

Available types include All Clients (Interactive Reporting Studio, Thin Client (EPM Workspace), Plug-in Client (Scheduler), Oracle Hyperion Smart View for Office, Fusion Edition, (EPM Workspace).

Note: To remove a document-level event association, in Enable For, deselect the type of document.

Using Script Editor

You can use Script Editor (a built-in feature) to add scripts to events. Open the Script Editor for an object, an active dashboard section, or a document. Script Editor contains the object browser, description pane, events menu, and scripting pane.

➤ To open Script Editor, perform an action:

- From a section other than a dashboard section, select File, then Document Scripts
- From within a dashboard section, in Design mode, select Dashboard, then Scripts
- For an object, select Dashboard, then Scripts

Using the Object Browser

Script Editor provides an object browser in the left pane. The browser displays the object model, listing all available objects, properties, and methods. At the top of the hierarchy is Application, which represents the Interactive Reporting application and contains application-wide settings and options and methods, and properties. See Chapter 10, "Object Model Map."

Clicking objects or collections displays methods, properties, and internal objects. Double-clicking methods or properties generates scripts in the scripting pane of Script Editor.

The Application object contains a Documents collection and an ActiveDocument collection. Methods and properties of the active document `Sample1.bqy`, are available in two places in the object model hierarchy:

- Application, then Documents, then `Sample1.bqy`
- Application, then ActiveDocument

Scripts that access multiple open documents should use the Documents path to the methods and properties of a document. Scripts that affect only the currently active document can use the ActiveDocument path.

Using the Scripting Pane

You use the scripting pane to enter scripts that are attached to specific object events (such as mouse clicks, button clicks, and so on). Use the object model to access objects, properties, and methods. When you double-click an item in the object browser, a reference to the object, property, or method is displayed in Script Editor at the cursor location.

Using the Events Menu

Above the scripting pane is the Object drop-down list that includes all available objects associated with the selected document, section, or object. Adjacent to Object is the Event Trigger drop-down list. This list displays the events for the control object, the action that invokes the script attached to the event.

After selecting an event, you can enter JavaScript and reference the object model. To see or edit scripts that extend beyond the scripting pane boundaries, use the horizontal and vertical scroll bars.

Script Editor provides functions to: (cut), (copy), (paste), and (find and replace, see "Finding and Replacing Within Scripts" on page 65).

Other functions provided by Script Editor:

- **Check Syntax**—Verify the script is written and edited correctly
- **Go To Line number**—Navigate to a specific code line in the script

Cutting, Copying and Pasting Dashboard Objects

You can cut, copy, and paste embedded sections, controls, or graphic objects in a dashboard section.

➤ To cut, copy, and paste dashboard objects, embedded sections, or controls:

1 Select **Ctrl+D** to enter Design Mode.

2 Select an object, embedded section, or control on the content area.

Press **Ctrl+Click** to select multiple items. Selection handles display.

3 Select **Edit**, then **Cut**or select **Ctrl+X.**

The item is placed on the clipboard.

4 **Optional:** Select **Edit** , then **Copy** or select**Ctrl+C.**

5 Locate a position on the dashboard content area, position your cursor, and click.

6 Select **Edit**, then **Paste.**

The object, embedded section, or control is pasted to the new location.

Using the Description Pane and Online Help

When you select an item in the object model hierarchy, a brief description of the item is displayed in the description pane. For example, selecting the ActiveDocument properties item displays the description "Object/Document ActiveDocument."

➤ To display Help text for object model items:

1 Select the item.

2 **Click Help.**

A help dialog box is displayed. It provides information on the selected method or property, such as the type of argument expected.

Using a Sample JavaScript Script

Each level of the object model has a Methods folder that contains the methods (actions) applicable to objects at that level. You can use the methods to write scripts.

Testing Scripts Using the Interactive Reporting Execution Window

You can test scripts you are working on by copying and pasting them into the Interactive Reporting Execution window. For example, instead of closing and reopening a document to test the `OnStartup` script, copy and paste the script into the Execution window and press Enter. In Interactive Reporting, select View, then Execution Window.

Reviewing Error Messages in the Interactive Reporting Console Window

The Console window records all error messages that occur from the time Interactive Reporting starts until the application is closed or the window is cleared (with Edit Clear). In Interactive Reporting, select View, then Console Window.

➤ To view error messages in the Console window:

Finding and Replacing Within Scripts

The Script Editor Find and Replace function enables you to search scripts for strings, punctuation marks, and numbers. You can conduct partial-word and whole-word searches, apply case-sensitive constraints, and replace individual or multiple occurrences of your search item.

Table 30 Find/Replace Definitions

Field	Definition
Find What	Enter the search criteria—a string, punctuation mark, or number. If you do not stipulate whole word, the search criteria acts as a prefix. That is, “report” matches “reporting,” “reporter” and “reported.” Wildcard characters cannot be used.
Replace With	Enter the replacement text.
Match Whole Word	Instructs the Find/Replace feature to match only the entire text that matches exactly your search criteria. For example, “report” matches only “report”. It does not match “reports” “reporting”, “reporter” and “re-ported.”

Field	Definition
Match Case	Instructs the Find/Replace feature to match only the text that matches the case of your search criteria. For example, if you specify “Chart,” the found words must match “Chart” with a capital *C.*
Direction	Specify the direction from which to conduct the search, upward or downward. By default, the direction is downward.
Replace	Specify whether to replace, as indicated.
ReplaceAll	Specify whether to replace of all occurrences.
Close	Close the Find/Replace window.

Note: In JavaScript version 1.4, if a regular expression starts with ‘|’ , this character is treated as a vertical bar and not an alternate regexp metacharacter. For example, the regular expression “|aaa” matches the string “bbb|aaa” starting at the fourth position, and it does not match the string “aaabbb”. In JavaScript verion 1.5, the ‘|’ character is treated (when not quoted) as an alternate metacharacter . In this case, the regular expression “|aaa” means “empty string OR ‘aaa’”. For example “|aaa” matches the string “bbb|aaa” starting before the first character (and the matched string is empty). The same occurs with “aaabbb”. It matches the empty alternative before the first character. To make an older “|aaa” regular expression work in JavaScript 1.5, place quotes around the | character with a backslash.. For example, enter “|aaa” as “\|aaa”, or “|Target~”. In JS1.5 as “\|Target~”. Also note the JavaScript version 1.4 behavior not only occurs when ‘|’ is located at beginning of whole regular expression, but also at the beginning of regexp group. For example, you would need to change the regular expression “aaa(|bbb)” to “aaa(\|bbb)”. See http://developer.mozilla.org/en/docs/Core_JavaScript_1.5_Reference.

Note: When the Find/Replace feature finishes its search, the message "Reached the end of the script. All instances of search item replaced" or “Reached the end of the script. Cannot find Search item” is displayed.

3

Scripting Dashboard Controls

In This Chapter

Scripting Control Objects

The following topics explain how to associate JavaScript scripts with four of the Dashboard control objects: command buttons, radio buttons, check boxes, and list boxes. The exercises in these topics guide you through inserting a new Dashboard section in `Sample3.bqy`, adding control objects to the new section, and associating scripts with the controls.

Creating a New Dashboard Section

➤ To insert a new Dashboard section in `Sample3.bqy` and rename it:

1. Open the **Sample3.bqy** file from within Designer.
2. Select **Insert** , then **New Dashboard** to add a new Dashboard section to the Interactive Reporting document file.

   Inserting a new Dashboard section changes the Interactive Reporting document file to Design mode. The Content is blank and the Elements pane displays the sections, graphics, and control objects available for embedding in an Dashboard.
3. In the Section frame, double-click **Dashboard** to open the Section Label dialog box.
4. Type **Controls** in the Label field and then click **OK** to close the Section Label dialog box.

Changing a Control Object Title

When working with control objects, change the default *title* to a title the user understands.

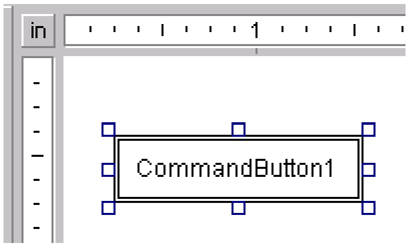
Note: The exercise in this section assumes that you have previously inserted a new Dashboard section in `Sample3.bqy` and renamed the new section *Controls*.

➤ To change an object's title:

1 **From the Elements pane, expand the Controls folder.**

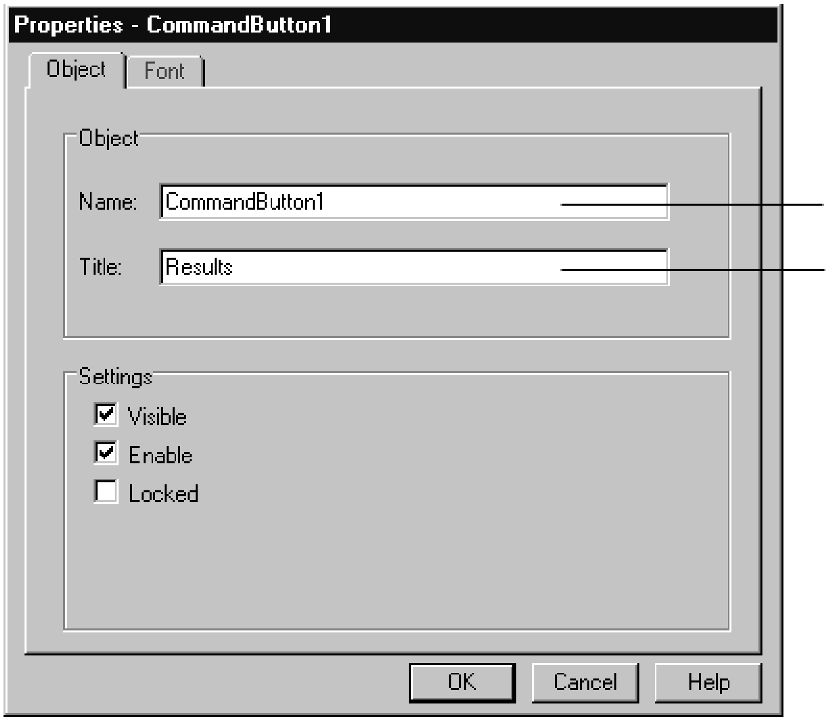
Note: The Controls folder is only visible in Design mode.

2 **Drag the desired control object (command button, text box, and so on) to the Content pane.**



3 **Double-click the control object to display the Properties dialog box.**

4 **Type a new entry in the Title field and click OK to view the results.**



Object

Name: CommandButton1

Title: Results

Settings

☑ Visible

☑ Enable

□ Locked

Name on the Script Editor
Title Bar and in the object model.

Title on the Button.

The entry in the Title field is displayed as a label on the control object. The entry in the Name field is displayed on the Title bar in the Script Editor and in the object model.

Associating Scripts with Command Buttons

A command button is typically used to initiate or activate a process or action.

Note: The exercise in this section assumes that you have previously inserted a new Dashboard section in `Sample3.bqy` and renamed the new section *Controls*.

Tip: If you need to change the text color or the color of the actual command button, use graphic object to create your own button. For more information, see Creating a Custom Button.

➤ To associate a script with a command button:

1 **Elements** pane, drag a command button control to the Content pane and use the Properties dialog box to specify its title as **RevSummary**

From the *Elements* pane, drag a command button control to the Content pane and use the Properties dialog box to specify its title as RevSummary.

(See "Changing a Control Object Title" on page 67 for detailed instructions.)

2 With the RevSummary object's selection handles visible, choose **Dashboard** , then **Scripts** or press **F8.**

The objects name (as shown in the Name field of the Properties dialog box) is displayed on the Title bar of the Script Editor, and the default event for the object (`OnClick`) is displayed in the Event drop-down box.

3 Use the Object browser to navigate to **ActiveDocument** , then **Sections**, then **RevSummary**, and then **Methods.**

4 Double-click **Activate.**

Interactive Reporting automatically enters the correct command in the Script Editor.

5 Click **OK** to save the script and close the Script Editor.

6 Toggle to Run mode, press **Ctrl+D** and click the **RevSummary** button.

The RevSummary section displays, as dictated by the script.

You have just learned to associate a script with a command button. You can review your script by activating the Controls Dashboard section, toggling to Design mode, and opening the Script Editor for the command button.

Example:

Create another button. Associate the button with a script that duplicates the RevSummary section. You can also try this on other Interactive Reporting document files.

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

Creating a Custom Button

You can create your own colorized button object rather than use the standard command button included in the Controls Catalog.

➤ To create a custom command button:

1 From the **Elements** pane, expand the folder containing the desired graphic object such as a rectangle or round rectangle object.

2 Select the graphic object; then, drag and drop the object into the Contents frame.

3 Double-click the graphic object to display the Properties dialog box.

The Object tab is displayed.

4 Type a new entry in the **Name** field and in the **Title** field.

The entry in the Title field is displayed as a label on the control object. The entry in the Name field is displayed on the Title bar in the Script Editor and in the object model.

5 Click the **Border and Background** tab.

6 Select the color you want to assign to the border of the button from the Color palette in the Border section.

7 Select the color you want to assign to the background of the bottom from the Background section.

8 Click **OK.**

9 Select **Scripts** from the shortcut menu.

The objects name (as show in the Name field of the Properties dialog box) is displayed on the Title bar of the Script Editor, and the default event for the object (`OnClick`) is displayed in the Event drop-down list box.

10 Add any scripts you want to associate with the custom button object.

For example, if you wanted to change the fill color of a radio button, you could type the following in the Script Editor:

```
//Button color, hex
ActiveSection.Shapes["RadioButton1"].Fill.Color = 0xFF3399
```

11 Click **OK.**

12 Type a new entry in the **Name** field and in the **Title** field.

13 Drag a text label object from the Graphics folder and place it on your custom button object.

14 Click the **Border and Background** tab.

15 Select the color you want to assign to the border of the text label from the Color palette in the Border section.

16 Select the color you want to assign to the background of the text label from the Background section.

17 Set font and size of the text if desired.

18 Click **OK.**

19 Select **Scripts** from the shortcut menu.

20 Copy and paste the script from the button object to the OnClick script for the text label.

If the name of the button object is displayed in the script associated with the text label, rename the button object to the name of the text label object. This will ensure that both the text label and button object remain dynamic.

Associating Scripts with Radio Buttons

Radio buttons are typically used to allow a user to select one option from a group of options; for example, to select one type of chart over another. The exercises in this section assume that you have previously inserted an Dashboard section in `Sample3.bqy` and renamed it *Controls*.

These exercises show you how to embed a chart and add radio buttons for user-control of the chart type.

➤ To embed a chart in an Dashboard section:

1 **In the Controls section, toggle to Design mode.**

When in Design mode, the Element pane is displayed below the Section frame as a hierarchical structure of available Dashboard objects.

2 **Elements pane, drag the chart named RevbyTime to the Content pane.**

The script in this exercise changes the chart type to a *line* chart. Before changing the chart with the script, verify that RevbyTime is a *vertical bar* chart. To verify and change the chart type, activate the RevByTime section and choose Format, then Chart Type, then Vertical Bar, and then return to the Controls section.

➤ To associate a script with a radio button:

1 **Elements pane, drag a radio button control to the Content pane and use the Properties dialog box to specify its title as Line.**

(See "Changing a Control Object Title" on page 67 for detailed instructions.)

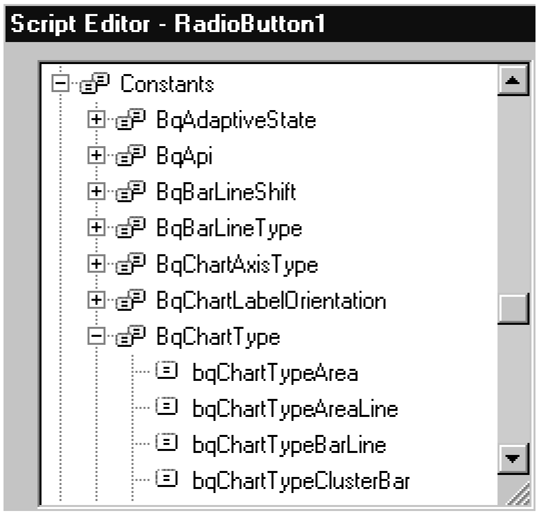
2 **With the Line object's selection handles visible, choose Dashboard, then Scripts or press [F8].**

3 **Use the Object browser to locate and expand the RevByTime section of ActiveDocument.**

4 **Expand the RevbyTime Properties folder, then double-click ChartType.**

Interactive Reporting automatically enters the first part of the script in the Scripting frame. *Property ChartType as BqChartType* is displayed in the Description frame. `BqChartType` is a constant whose values is displayed in the Constants collection of the Object browser.

5 **To select the applicable `BqChartType`, use the Object browser to scroll down to Constants and expand it, then expand the BqChartType collection.**



6 **In the Script Editor, type and equals sign (=) immediately after ChartType.**

7 **Double-click bqChartTypeLine.**

Interactive Reporting adds the rest of the script to the Scripting frame.

8 **Click OK to save the script and close the Script Editor.**

9 **Toggle to Run mode and click the Line radio button.**

The chart changes to a Line chart.

Example

Create a second radio button with the title Vertical Bar by toggling to Design mode and working through the preceding exercise.

Add a script to this radio button to change the chart type to vertical bar (`bqChartTypeVerticalBar`).

Tip: Radio buttons work in groups: when one button in the group is selected, the others in the same group are cleared. Set the group name in the Properties dialog box for each button. The default group name is RadioGroup.

Associating Scripts with Check Boxes

A check box is typically used to indicate whether an option should be turned on/off or is true/false.

The exercise in this section uses an `if...else` control structure. "JavaScript Control Structures" on page 101 goes into more detail on control structure syntax and usage.

Note: The exercise in this section assumes that you have previously inserted a new Dashboard section in `Sample3.bqy` and renamed the new section *Controls*. If you have been following the tutorial in sequence, you might try to associate a script to a check box on your own. The steps in the following procedure are very similar to the steps in the previous sections.

➤ To add a check box to the Controls section:

1 **Elements pane, drag a check box control to the Content pane.**

2 **Use the Properties dialog box to change the Name and Title properties as follows:**

- **Name**—chk_IntervalValues
- **Title**—Show/Hide Dollars

The Name *chk_IntervalValues* is used in the script for the Show/Hide Dollars check box.

Note: The rest of this exercise associates a script with the check box. The script turns on or off the display of revenue values (the `ShowIntervalValues` property) of the RevByTime chart.

Consider that a check box has two conditions: checked and unchecked (or cleared). Hence, the JavaScript needs to perform an action when a given condition is *true* and negate that action if it is *false*. "If" a condition exists, then a given action occurs; "else" the reverse happens.

➤ To associate the `ShowIntervalValues` property with the check box:

1 **With the Show/Hide Dollars check box object's selection handles visible, select Dashboard, then Scripts.**

2 **Type the following into the Script Editor:**

```
if ()
{
}
else
{
}
```

- The parentheses enclose a statement that tests the checked property of the check box (`chk_IntervalValues`).
- The first set of curly brackets, after the *if*, encloses the statement to execute if the test is true.
- The second set of curly brackets, after the *else*, encloses the statement to execute if the test is *not* true.

3 **Click in the parentheses after `if`, navigate to Controls Objects, then chk_IntervalValues, then Properties and then double-click Checked.**

```
if (chk_IntervalValues.Checked)
```

4 **Type ==true after Checked.**

```
if (chk_IntervalValues.Checked==true)
```

Use "==" to mean *is equal to* or *matches*.

5 **Click on the line between the curly brackets and add the statement to execute if the test is true.**

a. Use the Object browser to navigate to **Application**, then **ActiveDocument**, then **Sections**, then **RevByTime**, then **ValuesAxis**, then **Properties**, and then double-click **ShowIntervalValues**.

   You can see more of the object model by dragging the striped arrow at the bottom of the Object browser's scroll bar.

b. Type =**true** and a semicolon (;) at the end of the line in the Scripting frame.

```
if (chk_IntervalValues.Checked==true)
{
ActiveDocument.Sections["RevByTime"].ValuesAxis.ShowIntervalValues=true;
}
```

6 **Click on the line between the curly brackets (after else) and add the statement to execute if the test is false.**

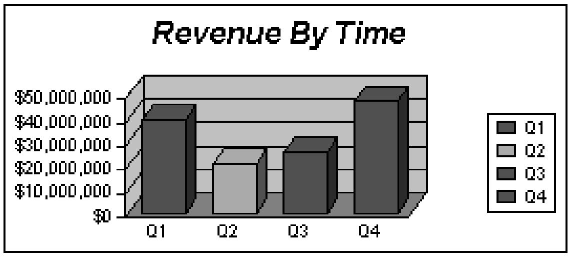
a. Use the Object browser to navigate to **Application**, then **ActiveDocument**, then **Sections**, then **RevByTime**, then **ValuesAxis**, then **Properties**and then double-click **ShowIntervalValues**.

b. Type =**false** and a semicolon (;) at the end of line in the Scripting frame.

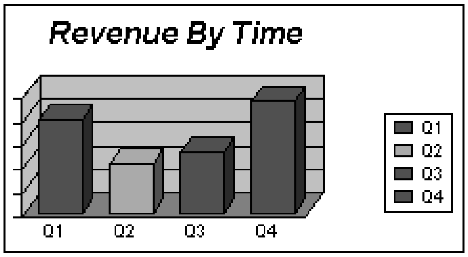
```
else
{
ActiveDocument.Sections["RevByTime"].ValuesAxis.ShowIntervalValues=false;
}
```

7 Click **OK** to save the script and close the Script Editor.

8 Toggle to Run mode, press **Ctrl+D**, and test how the check box works.



RevByTime Section with Show/Hide Dollars Check Box Selected.



RevByTime Section with Show/Hide Dollars Check Box Not Selected.

You have just learned to associate a script with a check box.

Example:

Create another check box. Associate the check box with a script that shows/hides another property of the chart.

Any property in the Property dialog (in the chart section) can be accessed through the object model and JavaScript. To view the RevByTime chart properties, activate this section by clicking the title in the Sections frame, click in white space close to the chart, right-click, and then select Properties from the menu that is displayed.

You can also try the exercise on some other Interactive Reporting document file.

Associating Scripts with List Boxes

A list box is typically used to list multiple values from which users can make one or more selections. This section introduces the list box with an exercise limited to creating the list values and displaying an alert with a single selection as the alert message.

Note: The exercise in this section assumes that you have previously inserted a new Dashboard section in `Sample3.bqy` and renamed the new section *Controls*.

Exercise: Associating a Script with a List Box

➤ To associate a script with a list box:

1. **Elements** pane, drag the list box icon from the to the Content pane of the Controls section.
2. With the list box control selection handles visible, choose **Dashboard**, then **Properties.**
3. Enter **Time** in the Name field and then clear the **Allow Multiple Selections** check box.

   When Allow Multiple Values is not selected, only one selection is allowed.
4. Click the **Values** tab, add the values **Today, Tomorrow,** and **Yesterday**, and then click **OK.**
5. In the Content frame, select the **Time** list box control and choose **Dashboard**, then **Scripts** or press **F8.**
6. Use the Object browser to navigate to **Application** , then **Methods** and then double-click **Alert.**

   ```
   Application.Alert()
   ```

   This method displays an alert box. The content of the alert is controlled with arguments added between the parentheses.
7. Click in the parentheses after Alert and use the Object browser to navigate to **Controls Objects**, then **Time**, then **SelectedList**, then **Methods** and then double-click **Item.**

   ```
   Application.Alert(Time.SelectedList.Item())
   ```

   The `SelectedList` object contains the list of user selections. `Time.SelectedList.Item()` needs a number as the argument to point to the specific item in the list of user selections.
8. Type **1** between the parentheses after Item.
9. Click **OK** to save the script and close the Script Editor.
10. Toggle to Run mode, press **Ctrl+D** and select an item in the Time list box.

    An alert is displayed, showing the selection.

You have just learned to associate a script with a list box.

Example

Edit the Alert to say **What a wonderful day!** after the selected item. Type a plus sign (+) in the parentheses after Item(1), and enclose the new phrase (a string) in quotes.

```
Application.Alert(Time.SelectedList.Item(1)+" What a wonderful day!")
```

4

JavaScript Syntax

In This Chapter

Basic JavaScript Syntax

JavaScript is a powerful programming language with three basic syntax rules, shown in the following table.

Rule	Example
JavaScript is case-sensitive.	*Alert* is not the same as *alert*.

Strings must be in quotes.	The following two statements define a variable *n* as the string *Hyperion*, and then insert the string value as an argument for the Alert method. The alert says `Brio`.
	`var n="Hyperion";`
	`Application.Alert(n);`
	The following two statements define *n* without quotes. The alert generates the error `Hyperion is not defined` because *Brio* is not a recognized JavaScript term.
	`var n=Hyperion;`
	`Application.Alert("The company name is "+n);`

Legal names (for variables, functions, and objects): ● Start with a letter and continue with only letters, numbers, or an underscore ● Do not use reserved words. ● Must be unique in context.	The first character must be a letter or an underscore(_), not a number. Subsequent characters may be any letter or digit or an underscore, but not a hyphen, period, or space. sample legal name: `_letters123` Names need to be unique in context. An Dashboard section cannot have two drop-down boxes with the same name, a function cannot have two variables with the same name, a document cannot have two sections with the same name See "Reserved Words" on page 88 for a complete list of reserved words.

JavaScript Code Structure

JavaScript uses *dot notation* or *object.method()* syntax. There is a dot, or period, between each model path segment and before the method or property. Methods always have parentheses. When there is a choice of Properties, they are specified with square brackets, or with square brackets and quotes when the choice is a string or name.

The following table summarizes the parts of a JavaScript.

Parts of Code	Examples

Object Model Paths:
● Start with an uppercase letter
● Separate path segments with a period (.)
	`ActiveDocument.Sections.Count`
	is the correct syntax to access the Count property of the Sections in Active Document, while
	`ActiveDocumentSections.Count`
	generates the following error because the separator between ActiveDocument and Sections is missing:
	`ActiveDocumentSections is not defined`

Methods (and Functions):
● Separate from the object path with a period (.)
● Include parentheses for arguments
	Activate() does not take arguments, but the parentheses are still required.
	`ActiveDocument.Sections["RevSummary"].Activate()`
	The Add() method requires a single argument, included in the parentheses.
	`Time.Add(TextBox1.Text)`
	The Alert() method requires at least one argument and allows for multiple optional arguments. Multiple arguments are separated by commas.
	`Application.Alert(TextBox1.Text,"Text Box")`

Properties:
● Separate properties from objects with a period (.)
● Refer to one of a collection of properties, by number, in brackets []
● Refer to one of a collection of properties, by name, in brackets, with quotes [""]
	When referring to the Count property of document sections, use:
	`ActiveDocument.Sections.Count`
	When referring to the first section (not the name, but the position in the section array in the object model), use:
	`ActiveDocument.Sections[1]`
	When referring to a specific section named RevSummary, use:
	`ActiveDocument.Sections["RevSummary"]`

Statement Separators:
● Statements must end with a return [Enter]
● End statements with both a semicolon (;) and a return [Enter] to avoid JavaScript errors
● Separate short statements on one line with a semicolon (;)
	Statements can be on separate lines:
	`Time.Add(TextBox1.Text);`
	`DropTime.Add(TextBox1.Text);`
	Multiple statements can be on one line, with a semicolon separating them:
	`Time.Add(TextBox1.Text);DropTime.Add(TextBox1.Text);`

Parts of Code	Examples

Comments
● Use // for single line or inline
● Use /* and */ for multiple lines
	`Time.Add(TextBox1.Text) // this is a comment`
	`// DropTime.Add(TextBox1.Text)`
	`// this line and the line above are both comments`
	
	`/*`
	`Everything in here is a comment until the end comment`
	`marker.`
	`*/`

JavaScript Operators

JavaScript provides one- or two-character symbols (operators) for use in assigning values, performing math, increasing and decreasing counters, and making comparisons.

It is important to use operators correctly to avoid JavaScript errors. You can avoid many errors if you understand:

- Assignment versus comparison operators
- How to use operators as characters in strings
- Concatenation versus addition

The following table lists available JavaScript operators.

Type of Operator	Symbol	Operation Performed
Assignment Operator returns the assigned value	=	Assign a value
Arithmetic Operators return the resulting value	+ +=	Addition or Concatenate Addition (or Concatenate) and assign resulting value
	- -=	Subtraction Subtraction and assign resulting value
	* *=	Multiplication: Multiplication and assign resulting value
	/ /=	Division Division and assign resulting value
	% %=	Modulus (integer remainder of dividing 2 operands) Modulus and assign resulting value
	++	Increment by 1 (x=x+1 is the same as x++)
	--	Decrement by 1 (x=x–1 is the same as x––)
Comparison Operators return a Boolean value `true` or `false`	==	Test if Equal

Type of Operator	Symbol	Operation Performed
	!=	Test if Not Equal
	>	Test if Greater Than
	<	Test if Less Than
	>=	Test if Greater Than or Equal To
	<=	Test if Less Than or Equal To
Logical Operators return a Boolean value `true` or `false`	&&	And (test if both operands are true)
	||	Or (test if one or the other operand is true)

Using Assignment versus Comparison Operators

JavaScript makes a distinction between assignment (=) and comparison (==) operators.

- Use = to assign the value on the right to the object on the left.
- Use == to test if the values on both sides match (the result is *true* if they match).

Note: The following exercise uses the Intelligence Client file `Sample3.bqy` and the RevByTime chart used in "Associating Scripts with Radio Buttons" on page 70. You can use any bqy document that includes a chart.

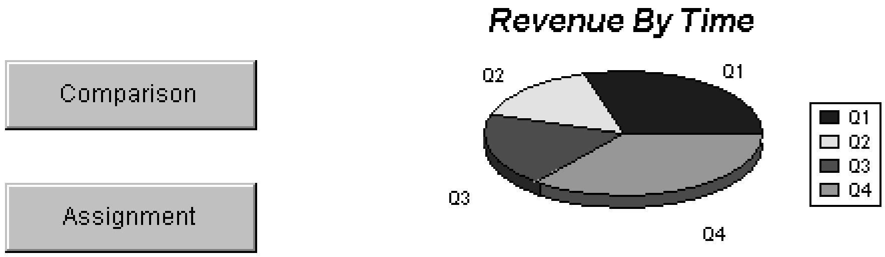
Example

Insert a new Dashboard section in `Sample3.bqy`. Add a line chart and two buttons titled *Comparison* and *Assignment*.

Add a JavaScript script to the Comparison button to compare the type of chart to `bqChartTypeVerticalBar` and open an alert that displays the result of the comparison. Test the comparison button. What does the alert say?

Script the Assignment button to change the chart type to `bqChartTypePie`. Try the comparison button again. What does the alert say now?

Figure 1 Assigning and Comparing chart type



Tip: Both JavaScript scripts act on the actual chart, not the view of the chart in the Dashboard section.

The script for the Comparison button uses == to test if ChartType *matches* bqChartTypeVerticalBar. The alert displays *true* if they match, *false* if they don't match.

The Assignment button's script uses = to set RevByTime's ChartType property equal to bqChartTypePie. The chart changes to a pie chart.

Exercise: Adding Comparison and Assignment Buttons

➤ To add a chart, and Comparison and Assignment buttons:

1 Open **Sample3.bqy** and insert a new **Dashboard** section. Rename it to **Equal Dashboard.**

Refer to "Creating a New Dashboard Section" on page 67 for information on renaming a section.

2 Add two command buttons by dragging them from the **Elements** pane to the Content pane.

3 Double-click one button and change the Title to **Comparison.**

4 Double-click the other button and change the Title to **Assignment.**

5 Drag the **RevByTime** chart from the **Elements** pane to the Content pane.

Verify that the chart is a vertical bar chart. (To verify and/or change the chart type, choose **Format, then Chart Type, then Vertical Bar** in the RevByTime section.)

Exercise: Using the Comparison Operator

➤ To script the Comparison command button to make a comparison and return an alert:

1 Select the **Comparison** button and choose **Dashboard**, **Scripts.**

2 Use the Object browser to navigate to **Application**, then **Methods**, and then double-click **Alert.**

3 Click in the parentheses of the Alert method.

4 **Navigate to ActiveDocument, then Sections, then RevByTime, then Properties and then double-click ChartType.**

Your script should look like this:

```
Application.Alert(ActiveDocument.Sections["RevByTime"].ChartType)
```

5 **Type the comparison operator (==) after the property ChartType.**

6 **Navigate to Constants, then BqChartType and then double-click bqChartTypeVerticalBar.**

Your script should now look like this:

```
Application.Alert(ActiveDocument.Sections["RevByTime"].ChartType==bqChartTypeVerticalBar
)
```

7 **Click OK to save the script and close the Script Editor.**

The Comparison button is ready to test.

In Run mode, click the Comparison button. The alert displays *true* if the chart is a vertical bar chart, *false* if the chart is any other chart type.

Exercise: Using the Assignment Operator

➤ To assign a specific chart type with the Assignment command button:

1 **In Design mode, select the Assignment button and choose Dashboard, then Scripts.**

2 **Navigate to ActiveDocument, then Sections, then RevByTime, then Properties and double-click ChartType.**

The script should look like this:

```
ActiveDocument.Sections["RevByTime"].ChartType
```

3 **Type the assignment operator (=) after the ChartType property.**

4 **Navigate to Constants, then BqChartType and double-click bqChartTypePie.**

The script should now look like this:

```
ActiveDocument.Sections["RevByTime"].ChartType=bqChartTypePie
```

5 **Click OK to save the script and close the Script Editor.**

6 **Toggle to Run mode and click the Assignment button, then click the Comparison button.**

Clicking the Assignment button assigns the chart type Pie to the RevByTime chart. Subsequent clicks on the Comparison button displays the alert *false* because the chart type is *not* vertical bar.

Including Operators in Strings

When JavaScript sees an operator, it performs the operation, even in strings. To tell JavaScript to treat an operator as a character, add the “escape” character, the backslash (\), in front of the operator.

Example:

The following exercise adds a script to the file c:\\dirname\\MyDoc.bqy to open the c:\\dirname \\YourDoc.bqy file. You can use any twoInteractive Reporting document for this exercise. Add a command button and a script to open a specificInteractive Reporting document.

Verify where the file is on your system, and copy the path to this file from your desktop. Escape the backslash in the directory path.

Exercise: Using Operators as Characters

➤ To create a command button that opens a file:

1 **Open Sample3.bqy and insert a new Dashboard section. Rename it Strings Dashboard.**

Refer to“Creating a New Dashboard Section” on page 67 for instructions on renaming a section.

2 **Drag a command button from the Elements pane to the Content pane.**

3 **Select the command button and choose Dashboard, then Scripts.**

4 **Use the Object browser to navigate to Application, then Documents, then Methods and then double-click Open.**

The Description pane shows that the arguments for the Open method are strings: `Document Open(String Filename, [optional] String DisplayName)`. The second argument is not required and this script does not include it. step 6 on page 83 through step 7 on page 83 adds the `String Filename` argument to the `Open` method.

Caution! Strings must be quoted! The Documents.Open() method requires string arguments.

5 **Switch to the desktop and find and copy the path (not the file name) to the file you wish to open.**

For the `What's New.bqy` document, the default path is `C:\Program Files\Brio \BrioQuery\Samples`.

6 **Add the path to the file inside the parentheses.**

a. Click inside the parentheses, type quotes, and paste the path plus an ending slash between the quote marks.

b. Type a backslash (\) in front of each slash in the file path.

The script should look similar to

```
Documents.Open("C:\\Program Files\\Brio\\BrioQuery\\Samples\\")
```

7 **Copy the file name and paste it at the end of the path in the current script.**

a. Switch back to the desktop and copy the exact file name.

b. Return to Intelligence Clients and open the Script Editor on the command button.

c. Click after the last slash in the path, before the ending quote mark, and paste the file name.

The script should look similar to:

```
Documents.Open("C:\\Program Files\\Brio\\BrioQuery\\Samples\\What's New.bqy")
```

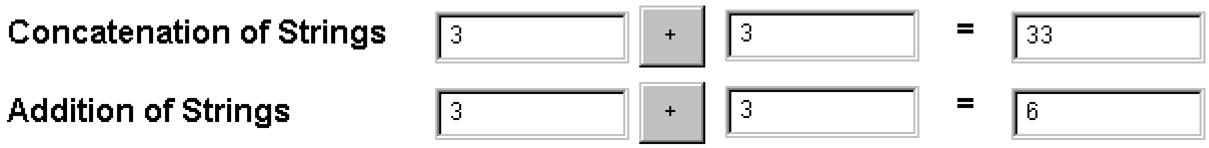
8 Click **OK** to save the script and close the Script Editor.

9 Toggle to Run mode and click the command button to open the file.

Concatenating versus Adding

JavaScript recognizes several types of data including: strings of characters (letters and numbers) and real or integer numbers. The data type affects the results of expressions using the + and += operators. If all the values in the expression are numeric, + performs addition. If one value is a string value, + concatenates.

Figure 2 Concatenation and addition of strings



Concatenation of Strings 3 + 3 = 33

Addition of Strings 3 + 3 = 6

Text boxes, list boxes, and drop-down boxes return string values, not numbers. If these strings are to be treated as numbers, JavaScript needs to be told to “parse” (change the value of) the string into a number.

JavaScript has two methods for parsing strings into numbers:

- `parseInt()` converts a string into an integer
- `parseFloat()` converts a string into a floating point number

Note: An Interactive Reporting document can be used for the following exercise.

Example

In Design mode, add three text boxes and a command button to a new or existing Dashboard section in `Sample3.bqy`, similar to Addition of Strings in Figure 2. Script the command button to add the values of the first and second text boxes, returning the result in the third text box.

Enter numbers in both the first and second text box and click the button. What is the result?

Exercise: Concatenating Values

➤ To concatenate the values of two text boxes to a third text box:

1 In Design mode, drag a text box from the **Elements** pane to the Content pane of a new or existing Dashboard section.

2 Double-click the text box and change the Name to **operand1**.

3 **Add a command button to the right of operand1.**

4 **Double-click the command button and change the Title to +.**

5 **Copy and paste the operand1 text box, move it to the right of the + button.**

The new text box is automatically renamed operand2 by Interactive Reporting.

Tip: Interactive Reporting allows copying and pasting of control objects only when the Console window is closed, and only within the same document section.

6 **Copy and paste the operand1 text box again and move it to the right of operand2.**

Interactive Reporting automatically renames the new text box operand3.

7 **Change the Name of operand3 to txt_result.**

8 **Select the + button and choose Dashboard, then Scripts**

9 **Add the following JavaScript:**

```
txt_result.Text=operand1.Text+operand2.Text
```

To avoid typing errors, use the Object browser to navigate to Dashboard Objects, and then select the Text property for each text box. Type only the = and + operators.

10 **Click OK to save the script and close the Script Editor.**

In Run mode, type numbers in operand1 and operand2. They concatenate to txt_result after you click the + button.

Note: The following exercise continues from the previous one, but changes the script to add instead of concatenate strings.

Exercise: Summing Values

➤ To sum the values in two text boxes to a third text box:

1 **In Design mode, select the + button and choose Dashboard, then Scripts.**

2 **Add the parseInt() method around operand1.Text.**

```
txt_result.Text=parseInt(operand1.Text)+operand2.Text
```

3 **Add the parseInt() method around operand2.Text.**

```
txt_result.Text=parseInt(operand1.Text)+parseInt(operand2.Text)
```

Caution! The method `parseInt` is lowercase, with the *I* capitalized.

4 **Click OK to save the script and close the Script Editor.**

In Run mode, the sum of the numbers in operand1 and operand2 is displayed in txt_result after you click the + button.

Variables

Variables are user-defined names that temporarily store data such as numbers, strings, or other objects (a string, a limit, a chart, a pivot, and so on). Variables can be created and defined when the variable is needed, or formally declared at the beginning of the script. Variables can be either local or global and have two important characteristics:

- **Name**—Word used to identify the variable. A variable name must not be a reserved word and must start with a letter. Letters, numbers, or an underscore can be used in the name. Do not use periods, spaces or hyphens.
- **Data Type**—Type of information stored in the variable, including:
  - **Numbers**—For example, 1 or 6.5777
  - **Booleans**—True or false
  - **Strings**—For example, Hyperion Solutions
  - **null**—Keyword which denotes a null value. The *null* value is also a primitive value.
  - **undefined**—Top-level property whose value is undefined. The undefined value is also a primitive value.

> **Note:** There is no method to distinguish explicitly between integer values (for example, 2) and real (floating point) numbers (for example, 3.14.). In addition, there is no `date` data type. However, you can use the Date object to handle date manipulations.

Chapter 5, "JavaScript Basics" introduces the use of variables in JavaScript scripts.

Declaring Local Variables

Use the term `var` to declare a new *local* variable.

```
var variable
```

A local variable is available only to the function or event handler script that defines it, and cannot be accessed by another function or event. Once a name is declared, it is assigned a value of *null* or *undefined* unless you assign a specific value when declaring the variable name.

In the Intelligence Client object model, it is helpful to adopt a naming convention that starts with the type of object and includes the action or value. For example, a list box containing StoreType data values could be named `Lbx_storeType`.

Caution! JavaScript is case-sensitive. A variable named `Lbx_StoreType` is not the same as `lbx_StoreType`.

Declaring Global Variables

Global variables are available throughout Intelligence Clients and include document and custom scripts, all Dashboard control and section scripts, the Report Designer section, the computed items features of the Results, Chart, and Pivot sections, and other BQY documents opened in the same instance of Intelligence Clients.

Note: Global variables are not available between BQY documents when using the web client because Web browsers do not support them. Cookies should be used to write variables shared between documents.

A global variable is defined outside of a function or event and once defined, can be accessed by other functions and events. You must assign a specific value to global variable when you declare it. If you type:

```
gvariable
```

you get a run-time "not defined" error because global variables can not have a value of null or undefined.

Note: There is no way to bypass a "not defined" error, not even by using try-catch syntax since it is a run-time error. The JavaScript engine halts execution before "try" has a change to "catch" an exception. This is expected behavior of the JavaScript language.

To prevent this error, assign a specific value to the global variable when declaring it:

```
gvariable=25
```

You should consider some naming convention to distinguish global variables from local variables, such as an extra "g" at the beginning of the global variable name.

Dynamically Declaring Variables

You can also dynamically declare a variable by creating a new property on an object, for example:

```
ActiveDocument.MyName = "Dan"
```

These variables are similar to global variables but they can be seen only within the scope of the object with which they are associated, and they exist only as long as the object exists. To access this variable you need to include the object name as well as the variable name, for example:

```
Console.Writeln(ActiveDocument.MyName)
```

Assigning Values

The JavaScript assignment operator (=) assigns a value to a variable. The type of data can be a number (the number of items or the result of a calculation), an object (a string or an object path), or a boolean (true or false).

```
var Result = 15   // The value 15 is assigned to
                     the variable named Result
var Result = Result + 2 // The variable Result is incremented by 2
```

The data type can be changed at any time. For local variables, the data type is null or undefined until a value is assigned. Once a value is assigned, the variable's data type defines whether the + operator concatenates or adds; to add, all values must be numeric. See "Concatenating versus Adding" on page 84 for converting a string to a number.

JavaScript is a dynamically typed language. This means that you do not need to specify the data type of a variable when you declare it. Data types are converted automatically as needed during script execution. This allows you to reuse variables with different data types. For example, if you define a variable, such as:

```
var version = 6.5
```

Later, you can assign a string value to the same variable:

```
version = "Brio Intelligence 6.6"
```

Since JavaScript is dynamically typed, this assignment does not cause an error message.

In expressions involving numeric and string values with the + operator, JavaScript converts numeric values to strings. For example, consider the following statements:

```
// returns The version is 6.5
x = "The version is " + 6.5
```

In statements that involve other operators, JavaScript does not convert numeric values to strings, for example:

```
"50" - 5 // returns 45
"60" + 5 // returns 65
```

Reserved Words

JavaScript sets aside certain words that have a unique meaning and cannot be used as function or variable names, or method and object names. Some of the reserved words are in use in the current version of JavaScript or are intended for use in a future version.

The following table lists JavaScript's reserved words.

Reserved Words				
abstract	default	goto	null	this
add	do	if	package	throw
application	double	implements	private	throws
boolean	else	imports	protected	transient
break	extends	in	public	true

Reserved Words				
byte	false	instanceof	return	try
case	final	int	short	var
catch	finally	interface	static	void
char	float	long	super	while
class	for	native	switch	with
const	function	new	synchronized	
continue				

Special characters to avoid when naming sections and variables:

Table 31 Reserved Characters

Reserved Characters	
<space>	'
<comma>	!
<period>	&
/	*
\	%
“	@
|	?
(euro symbol)	$

5 JavaScript Basics

In This Chapter

Using Drop-Down Boxes

Drop-down boxes are typically used to list multiple values from which users can make one selection. You can use limits that were set in other sections to limit the values available in drop-down boxes.

For example, Activate the Limits Results section of `Sample1mod.bqy` and double-click Territory on the Limit line. There are seven *available* Territory values in the database, but only three are selected in the Show Values list (Asia, North America, and South America). This means the Results section displays only products sold in Asia, North America, and South America.

In the Limits Dashboard section, the territories shown in the drop-down box also are limited to these three selected values.

This section shows you how to create a script that allows the user to view the pie chart according by territory as selected from the drop-down box.

The exercises in this section add a script to a drop-down box to display an alert. The exercises introduce two concepts:

- Accessing a Drop-Down Selection
- Using a Variable for the Selection

Note: Use the file `Sample1mod.bqy` to add a script to DropDown1 in the Limits Dashboard section.

Accessing a Drop-Down Selection

➤ To add a JavaScript script to the drop-down box to display the selection in an alert:

1 Open **Sample1mod.bqy**, activate the **Limits Dashboard** section, and change to Design mode.

2 **Double-click DropDown1 to display the Properties dialog box, change the Name field to drp_Territory, and then click OK.**

Tip: To clarify the purpose of control objects when viewed through the object model, adopt a naming convention that includes the type of object (`drp_` for a drop-down box) and the type of information or action associated with the object (`Territory` for database Territory options).

3 **Open the Script Editor for the drop-down box.**

Right-click drp_Territory (changed from DropDown1)—which should still have handles around the box—and select Scripts from the shortcut menu.

Note: This rest of this exercise shows how to click through the object model to associate a script to drp_Territory, a Limits Dashboard object. The script will apply the `Item` method, using a pointer to the selected item (SelectedIndex) as the method argument.

4 **Expand Methods and Properties for Limits DashboardDashboard Objects., then drp_Territory.**

To see both methods and properties for drp_Territory, scroll down, or click the striped arrow below the scroll bar to expand the **Elements** pane.

5 **Double-click the Item method to enter drp_Territory.Item() in the Script Editor.**

```
drp_Territory.Item()
```

6 **Place the cursor inside the parentheses of the Item method, and then double-click SelectedIndex from the Properties folder.**

7 **Type a semicolon (;) and a return [Enter] at the end of the line.**

The line now reads:

```
drp_Territory.Item(drp_Territory.SelectedIndex);
```

The `SelectedIndex` property of a drop-down box is the index number (or position number) of the selected value. `Item()` uses this argument to return the text the user selected.

8 **Add a new statement using the Alert method with the drop-down selection as its argument.**

a. Type **Alert**() or double-click **Alert** from within **Application**, then **Methods**.

b. Copy the first JavaScript statement and paste it inside the Alert method's argument parentheses.

c. Delete the semicolon from within the argument.

d. Type a semicolon (;) and return [**Enter**] at the end of the statement.

9 **Click OK to save the current script and close the Script Editor.**

10 **Toggle to Run mode, click the arrow in the drp_Territory drop-down box and select an item.**

Selecting an item causes a popup message to is displayed.

The Alert method displays a popup message that contains the name of the item selected from the drop-down box.

Using a Variable for the Selection

A *variable* is a temporary holder of information, such as the user's selection in a drop-down box. Using variables clarifies the programming logic, making it easier to troubleshoot. The table below lists the characteristics of a variable.

Characteristic	Explanation
Name	Names must start with a letter or underscore. Subsequent characters may include letters, numbers, or the underscore (`drp_Territory` is a legal name). A variable name cannot be a reserved word (`new` is a reserved word, but `newVariable` is not). Use a JavaScript `var` statement to declare a variable name before it is used. For example: `var newVariable;` In the above statement, JavaScript reserves the name `newVariable`, with a value of null or undefined.

Value	Values are assigned to variables with the JavaScript assignment operator (=). Any type of data can be assigned to a variable: an object (a string, a user selection, an object path), a boolean (true or false), or a number. For example:
	`var newVariable;`
	`newVariable=true;`
	The value can be assigned in a separate statement like above, or in the same statement that declares the variable name. For example:
	`var newVariable=true;`
	`newVariable` (and its value) is available only within the script or function that declares it, preventing accidental changes from other scripts using the same variable name.

This exercise uses a variable to hold the user's selection (the first line of the script from the preceding exercise). A variable can be declared in one statement and the value assigned in another:

```
var Selection;
Selection=Territory.Item(Territory.SelectedIndex);
```

or the statements can be combined:

```
var Selection=Territory.Item(Territory.SelectedIndex);
```

Note: The following exercise continues from the previous section. This exercise alters the script in the drop-down box (renamed drp_Territory) in the Limits Dashboard section of `Sample1mod.bqy`.

Exercise: Declaring a Variable

➤ To declare a variable and assign the drop-down selection:

1 **Toggle to Design mode, open the Script Editor on drp_Territory.**

2 **Type var Selection= at the beginning of the first line of the existing script.**

```
var Selection=drp_Territory.Item(drp_Territory.SelectedIndex);
```

This declares a new variable *Selection*, and assigns the user's selection to it.

3 **Replace the argument for Alert with the variable Selection.**

```
var Selection=drp_Territory.Item(drp_Territory.SelectedIndex);
Alert(Selection)
```

4 **Click OK to save the script and close the Script Editor.**

5 **Test the script by toggling to Run mode and selecting an item from the drop-down box.**

An alert containing the name of the selected item should is displayed.

If the alert displays "undefined", the alert argument has not been assigned a value. This is most likely a typo: If the variable is defined with a capital "S" for *Selection*, the variable in the alert argument must also have a capital "S".

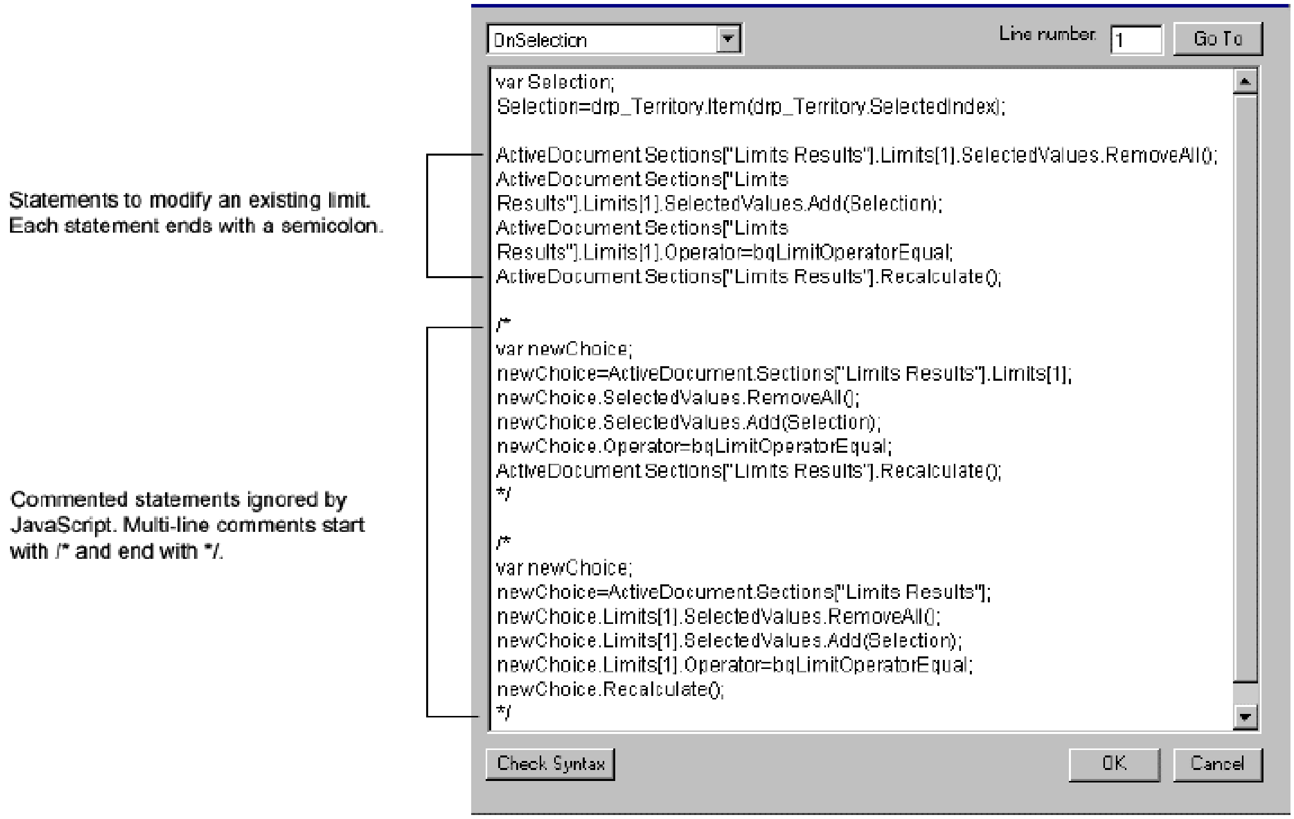
The exercises in this section used JavaScript and the Intelligence Client object model to access a user's selection in a drop-down box. The `Item` method, with `SelectedIndex` as the argument can be acted on directly, or a variable can hold the selection, clarifying the JavaScript logic: get the selection, act on it.

Modifying Filters

The drop-down box in the Limits Dashboard section of `Sample1mod.bqy` displays three Territory options and a pie chart of total unit sales. Continuing from the previous exercises, the exercises in this section change the drop-down script so the user's selection modifies the Limit line in the Limits Results section instead of displaying an alert. When the Limits Results section recalculates, the pie chart shows data for the selected Territory.

Study the JavaScript below. The basic steps to modify an existing limit are:

1. Remove the existing value.
2. Add a value.
3. Assign a "limit operator".
4. Recalculate (results and tables), or Process (queries).



Statements to modify an existing limit. Each statement ends with a semicolon.

Commented statements ignored by JavaScript. Multi-line comments start with /* and end with */.

```
var Selection;
Selection=drp_Territory.Item(drp_Territory.SelectedIndex);

ActiveDocument.Sections["Limits Results"].Limits[1].SelectedValues.RemoveAll();
ActiveDocument.Sections["Limits
Results"].Limits[1].SelectedValues.Add(Selection);
ActiveDocument.Sections["Limits
Results"].Limits[1].Operator=bqLimitOperatorEqual;
ActiveDocument.Sections["Limits Results"].Recalculate();

/*
var newChoice;
newChoice=ActiveDocument.Sections["Limits Results"].Limits[1];
newChoice.SelectedValues.RemoveAll();
newChoice.SelectedValues.Add(Selection);
newChoice.Operator=bqLimitOperatorEqual;
ActiveDocument.Sections["Limits Results"].Recalculate();
*/

/*
var newChoice;
newChoice=ActiveDocument.Sections["Limits Results"];
newChoice.Limits[1].SelectedValues.RemoveAll();
newChoice.Limits[1].SelectedValues.Add(Selection);
newChoice.Limits[1].Operator=bqLimitOperatorEqual;
newChoice.Recalculate();
*/
```

Modifying a Results Filter

The advantage of modifying a Results section filter is that it excludes data from the display without affecting the local data set and without a database connection. Processing Query section limits requires a database connection.

Caution! There must be a limit on the Limit Line for the script to execute. If there is no existing limit, an "uncaught exception" error will be recorded in the Console window and the rest of the script will not execute.

Note: This exercise assumes that the Script Editor for the drop-down box drp_Territory is open. If it is not, open the file `Sample1mod.bqy` and activate the Limits Dashboard section. In Design mode, select the drop-down box drp_Territory, and then right-click to select Scripts.

➤ To modify an existing limit with the drop-down selection:

1 **Delete the Alert line in the existing JavaScript.**

There should be one statement left in the scripting frame.

```
var Selection=drp_Territory.Item(drp_Territory.SelectedIndex);
```

2 **Use the Object browser to navigate to Application), then ActiveDocument, then Sections, then Limits Results, then Limits, then 1, then SelectedValues, then Methods and then double-click RemoveAll.**

The Limits Results section is low in the object model hierarchy because it is the result of a second query (every section that pertains to the first query comes before it).

The `RemoveAll` method deletes any current values, making “room” for the new selection.

3 **Type a semicolon (;) at the end of the statement, and then press [Enter].**

```
ActiveDocument.Sections["Limits Results"].Limits[1].SelectedValues.RemoveAll();
```

JavaScript recognizes an end-of-statement when it sees a return [Enter] or a semicolon (;). It is good practice to end statements with both, especially when a long line wraps and starts looking like several lines in the Script Editor.

4 **Add the user’s selection as the next statement in the script.**

a. Double-click the **Add** method (in the same Methods folder as RemoveAll).

b. Type the variable **Selection** as the argument for **Add** (inside the parentheses).

c. Type a semicolon (;) at the end of the statement, and then press [**Enter**].

```
ActiveDocument.Sections["Limits
Results"].Limits[1].SelectedValues.Add(Selection);
```

This step adds the selection the user chose as the value for the limit.

5 **For the next statement, add the limit operator.**

a. Navigate up the object model to object **1**, expand the **Properties** folder, and then double-click **Operator**.

b. Type = after Operator.

The Intelligence Client object model includes a collection of Constants. Use the `BqLimitOperator` constant to set the Operator value in the next step.

c. In the object model Constants collection, open **`BqLimitOperator`**, and double-click **`bqLimitOperatorEqual`**.

You may want to close all expanded folders to access Constants more easily.

d. End the statement with a semicolon (;) and a return [**Enter**].

```
ActiveDocument.Sections["Limits
Results"].Limits[1].Operator=bqLimitOperatorEqual;
```

6 **Double-click the Recalculate method from ActiveDocument, then Sections, then Limit Results, then Methods.**

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

The `Recalculate` method instructsInteractive Reporting to recalculate the results data. It takes no arguments, but still has parentheses because it is a method.

7 **Click OK to save the script and close the Script Editor.**

8 **Toggle to Run mode and select a different item in the drop-down box.**

In Run mode, selecting an item in the drop-down box changes the limit value and operator, and recalculates the results to include data for a Territory equal to the selection. The pie chart updates with the current results data.

Using a Variable for an Object

The JavaScript script for modifying a limit with the drop-down selection is:

```
var Selection=drp_Territory.Item(drp_Territory.SelectedIndex);
ActiveDocument.Sections["Limits Results"].Limits[1].SelectedValues.RemoveAll();
ActiveDocument.Sections["Limits Results"].Limits[1].SelectedValues.Add(Selection);
ActiveDocument.Sections["Limits Results"].Limits[1].Operator=bqLimitOperatorEqual;
ActiveDocument.Sections["Limits Results"].Recalculate()
```

Several words in the last four statements are repeated in each line—the path to Limits[1] and the path to Limits Results section. A variable can hold these objects (the repeated words), which makes the JavaScript logic easier to read.

Note: This following exercise edits the JavaScript associated with drp_Territory from the previous exercise (in the Limits Dashboard section of `Sample1mod.bqy`) to use a variable for the object path.

Example

Refer to figure in the and study the last two sets of statements—using a variable for the object (path). Select one of the sets to enter in Design mode.

Start by commenting out the last four statements in your current script by typing `/*` before the first `ActiveDocument.Sections` statement and `*/` after the fourth one. This will make the four lines *comment* lines that JavaScript does not execute.

Enter one set of statements with a variable *newChoice* holding the object path. Do not enclose these statements in comments.

Test the new statements in Run mode. Selecting an item in the drop-down box should still change the pie chart results.

Caution! Do not end an object path value with a period. The syntax error “missing name after. operator” refers to an incomplete object model path.

The first set of commented statements in the figure below creates a variable equal to navigating from ActiveDocument to Limits[1].

```
var newChoice;
newChoice=ActiveDocument.Sections["Limits Results"].Limits[1];
newChoice.SelectedValues.RemoveAll();
newChoice.SelectedValues.Add(Selection);
newChoice.Operator=bqLimitOperatorEqual;
ActiveDocument.Sections["Limits Results"].Recalculate()
```

The second set creates a variable equal to navigating from ActiveDocument to the Limits Results section (one step up in the hierarchy).

```
var newChoice;
newChoice=ActiveDocument.Sections["Limits Results"];
newChoice.Limits[1].SelectedValues.RemoveAll();
newChoice.Limits[1].SelectedValues.Add(Selection);
newChoice.Limits[1].Operator=bqLimitOperatorEqual
```

Modifying a Query Filter

The same JavaScript logic modifies a Results and a Query filter, except a query uses Process instead of Recalculate to update the data with the new filter value. Processing a query requires a database connection.

Example

Change the existing limit in the Limits Query section based on the drop-down selection. Use the `Process` method instead of the `Recalculate` method to update the query results.

```
ActiveDocument.Sections["Limits Query"].Process()
```

To test the script, a database connection is needed. Use `Brio 6.0 Sample 1.oce`. Leave the Host Name and Host Password blank.

The exercises in this section modified a limit according to a user's selection. Variables were used to hold the user's choice and to hold the object model path to the limit line object.

Finishing the Interactive Reporting Document File

The Limits Dashboard section contains an active pie chart and a drop-down box that allows the user to choose limit options. To "finish" this Interactive Reporting document file as a user interface, set Limits Dashboard as the section that displays when the Interactive Reporting document file is opened. Additional features to add are:

- Setting a Chart Fact
- Hiding Toolbars

Setting a Chart Fact

A chart in an Dashboard section can be passive (View-only), can activate the chart section when clicked (Hyperlink), or can access drill down options when right-clicked (Active). The pie chart in the Limits Dashboard section is set to Active in the Properties dialog box, which allows direct access to underlying Product Line data.

When the user drills down into underlying data, the chart section's X-Categories are changed to reflect this action. With JavaScript, the chart can be returned to the top level (Product Line).

The following script executes when the Limits Dashboard section is activated.

Note: This following exercise sets the XCategory of the Limits Chart section to display the top level fact: Product Line. The script is added to the Limits Dashboard section of `Sample1mod.bqy`.

➤ To script an Dashboard section to set a chart to a specific fact:

1 Activate the **Limits Dashboard** section by clicking the title in the **Section** frame.

2 Toggle to Design mode and choose **Dashboard** , then **Scripts** to open the Script Editor on the active section.

3 Declare a variable **topDrill** and assign the string **Product Line.**

```
var topDrill="Product Line";
```

The variable *topDrill* now holds the chart fact Product Line.

4 **RemoveAll** XCategories from the **Limits Chart** section.

```
ActiveDocument.Sections["Limits Chart"].XCategories.RemoveAll();
```

5 **Add topDrill** to the XCategories of the chart section.

```
ActiveDocument.Sections["Limits Chart"].XCategories.Add(topDrill);
```

6 Click **OK** to save the script and close the Script Editor.

In Run mode, when the user selects Limits Dashboard in the Section frame, or the Interactive Reporting document file automatically activates this section, the chart shows Product Line data.

The same JavaScript can be added to a command button, so the user can choose to return the chart to Product Line at any time.

Hiding Toolbars

JavaScript scripts can be added to the Interactive Reporting document file itself, executing *OnStartup* or *OnShutdown*. Add scripts to these events to perform any startup and shutdown tasks. The following exercise hides the application Status bar, the Format toolbar, and the dInteractive Reporting document file Section/Element pane. There are other toolbars accessible in each of these areas of the object model. Use this exercise as a starting point to accessing the various toolbars.

➤ To hide toolbars:

1 Choose **File**, then **Document Scripts** to open the Script Editor on the Interactive Reporting document file.

2 Use the Object browser to navigate the object model, and set the property for the Status bar, the Formatting toolbar, and the Section/Element panes to **false.**

a. In **Application**, then **Properties**, then double-click **ShowStatusBar**, and then type =**false**. End the statement with a semicolon and a return.

```
Application.ShowStatusBar=false;
```

The menu bar can be accessed at this level of the object model, with `ShowMenuBar` property.

b. In **Application**, then **Toolbars**, then **Formatting**, then **Properties**, then double-click **Visible**, and then type =**false**. End the statement with a semicolon and a return.

```
Toolbars["Formatting"].Visible=false;
```

Several other toolbars are accessible under Application, then Toolbars.

c. In **Application**, then **ActiveDocument**, then **Properties**, then double-click **ShowCatalog**, and then type =**false**. End the statement with a semicolon and a return.

```
ActiveDocument.ShowCatalog=false;
```

The Section title bar can be accessed at this level of the object model with the `ShowSectionTitleBar` property.

6

JavaScript Control Structures

In This Chapter

Understanding Control Structure Syntax

The basic syntax of a control structure is:

type of control (control statement) { block of statements to execute, based on the value of the control statement ; }

- Parentheses hold the control statement and are a required part of the control structure syntax.
- Curly brackets delineate the control statement block or control body.
- Each statement in the body of the control structure ends with a semicolon.

The result of the control statement defines whether or not the statements in the control block are executed. Statements outside the control block are always executed. The following table describes three JavaScript control structures and their syntax.

Control	Explanation	Syntax

If	An *if* tests the condition of a control statement, using the comparison or logical operators in the JavaScript Operators table.
	The body statements execute only if the condition tests true.
		`if (`*`condition returning true or false`*`)`
		`{`
		  `statements;`
		`}`
		`statements executed after control;`

Control	Explanation	Syntax

If...else	An *if...else* tests the condition of a control statement, using the comparison or logical operators in the JavaScript Operators table.
	The body of the *if* executes only when the condition tests true. The body of the *else* executes if the condition tests false.
		if (*condition returning true or false*)
		{
		  statements;
		}
		else
		{
		  statements;
		}
		statements executed after control;

Switch	A *switch* compares an expression to multiple *case* values. Statements within a case execute only if the case value matches the expression value.
	Each case can end with an optional *break* statement which breaks out of the switch control block and continues execution with statements that follow the end of switch.
	An optional *default* statement executes only if none of the case values match the expression value.
	If there is no default statement, and no matching case value is found, execution continues with the statement that follows the end of switch.
		switch (*expression returning a value*)
		{
		case *value* :
		  statements;
		break;
		
		case *value* :
		  statements;
		break;
		
		.
		.
		.
		
		default :
		   statements;
		}
		statements executed after control;

About if...else Statements

The JavaScript logic to set the ChartType to a Line if the check box is checked is:

```
if (the checked property of the Check Box==true)
{
set the chart type to a line chart
}
```

JavaScript tests whether the Checked property of the check box matches *true*. When the condition test returns true (yes, the Checked property matches true), it executes the body of the *if* statement block. When the condition test returns false (no, the Checked property does not match true), it skips the *if* statement block.

With this JavaScript logic, when the check box is selected, the pie chart changes to a line chart. When the check box is cleared, the chart does not change because the condition test (chk_ChartType.Checked==true) is false.

Use an *if...else* statement to expand the *if* to change the chart back to a pie chart when the condition test is false. JavaScript tests the Checked property of the check box. When the check box is checked, the body of the *if* executes. When the check box is not checked, the body of the *else* executes.

Note: The following exercise adds scripts to new control objects in the Limits Dashboard section of `Sample1mod.bqy`.

Exercise: Using an if...else Statement to Change Chart Types

➤ To display a line chart *if* the check box is checked, otherwise (*else*) display a pie chart:

1 Add a new check box to **Limits Dashboard,** change the Name to **chk_ChartType,** and change the Title to **Show Chart As A Line Chart** (use the Properties dialog box).

2 Open the Script Editor on the new **Show Chart As A Line Chart** check box.

3 Type **if (),** a return **[Enter]** an open curly bracket (**{**), two returns **[Enter]** and a close curly bracket (**}**).

```
if ()
{
}
```

The parentheses hold the controlling condition test. The curly brackets are for the body of the *if*, executed only when the condition tests to true.

4 Click inside the control part of the *if*, then use the Object browser to navigate to **Limits Dashboard Objects** , then **chk_ChartType** , then **Properties** and then double-click **Checked.**

```
if (chk_ChartType.Checked)
{
}
```

The Description pane shows Property Checked as Boolean. There are two Boolean values: *true* and *false*.

5 After the chk_ChartType.Checked property, type **==true.**

```
if (chk_ChartType.Checked==true)
{
}
```

Verify that there are two equal signs, meaning *match true*, not *assign the value* true. Condition statements use the comparison or logical operators "Logical Operators" on page 116.

6 Add a statement in the body of the *if* statement to change the Limits Chart to type Line.

a. Click inside the body of the *if* (on the blank line between the curly brackets), then use the Object browser to navigate to **Application** , then **ActiveDocument**, then **Sections**, then **Limits Chart**, then **Properties** and then double-click **ChartType.**

```
if (chk_ChartType.Checked==true)
{
ActiveDocument.Sections["Limits Chart"].ChartType
}
```

The Description Pane shows Property ChartType as BqChartType. Find the collection for BqChartType in the object model under Constants.

b. After **ChartType**, type an equal sign (=), navigate to **Constants**, then **BqChartType**, and then double-click **bqChartTypeLine**.

c. Type a semicolon (;) at the end of the statement.

```
if (chk_ChartType.Checked==true)
{
ActiveDocument.Sections["Limits Chart"].ChartType=bqChartTypeLine;
}
```

The semicolon clarifies where the statement ends and is recommended practice.

7 **On a new line, after the close curly bracket for the *if*, type else, a return [Enter], an open curly bracket ({), two returns [Enter], and a close curly bracket (}).**

```
}
else
{
}
```

The curly brackets are for the body of the else, executed only when the condition tests false.

8 **Click in the body of the *else* and use the Object browser to add a statement to change the Limits Chart to a pie chart.**

```
else
{
ActiveDocument.Sections["Limits Chart"].ChartType=bqChartTypePie;
}
```

The statement should end with a semicolon.

9 **Click OK to save the script and close the Script Editor.**

10 **Toggle to Run mode to test the script.**

When Show Chart As Line Chart is checked, the chart type changes to Line. When it is not checked, the chart type is Pie. The on/off (checked/unchecked) states of the check box controls two chart type options.

Exercise

Add a new check box to the Limits Dashboard section of `Sample1mod.bqy`. Add an if statement that shows the chart legend when the check box is selected, and hides it when the check box is not selected.

ShowLegend requires a Boolean (true or false) assignment. The statement to show the chart legend is:

```
ActiveDocument.Sections["Limits Chart"].ShowLegend=true;
```

The JavaScript script to show a chart legend if chk_ShowLegend is *true* is:

```
if (chk_ShowLegend.Checked==true)
{
ActiveDocument.Sections["Limits Chart"].ShowLegend=true;
}
else
{
ActiveDocument.Sections["Limits Chart"].ShowLegend=false;
}
```

Caution! Use one equal sign when assigning a value, use two equal signs when testing if the value matches true.

Tip: The check box and its script show the chart legend when the check box is selected and its state becomes “checked.” Since the chart legend is already showing, the check box must be cleared to hide the legend the first time the Dashboard section is used. The initial state of the chart and the check box can be set with JavaScript statements associated with the OnActivate event of the Dashboard section:

```
ActiveDocument.Sections["Limits Chart"].ShowLegend=true;
chk_ShowLegend.Checked=true;
```

Tip: Set both ShowLegend and the check box Checked properties to true when the section is activated.

About switch Statements

switch statements use expressions, or cases, to control statement execution. Each case holds one possible value and includes the statements to execute when the value matches the expression result. The following table compares the control logic of a *switch* to an *if...else*.

switch	if...else

switch (*CheckBox*.Checked)
{
case true :
  set the chart type to a line chart
case false :
  set the chart type to a pie chart
}
	if (*CheckBox*.Checked==true)
	{
	  set the chart type to a line chart
	}
	else
	{
	  set the chart type to a pie chart
	}

JavaScript evaluates the expression in a *switch*, then compares the expression value to each case until it finds a matching value. The statements in the matching case are executed and the next case is compared. If the matching case ends with a *break* statement, JavaScript skips the rest of the cases (conserving execution time).

Note: Add a check box to the Limits Dashboard section of `Sample1mod.bqy`. Use *switch* logic instead of *if...else* logic to change the chart type.

Exercise: Using a switch Statement to Change Chart Types

➤ To switch to a line chart when Checked is true, or to a pie chart when Checked is false:

1 Add a new check box to **Limits Dashboard**, and use the Properties dialog box to change the Title to **Switch Chart To Line Chart** and the Name to **chk_ChartType2**.

2 Open the Script Editor on the new **Switch Chart To Line Chart** check box. [**F8**]

3 Type **switch ()** a return **[Enter]**, an open curly bracket (**{**), two returns **[Enter]**, and a close curly bracket (**}**).

```
switch ()
{
}
```

The parentheses are for the expression. The curly brackets are for the body of the switch.

4 Click inside the expression parentheses, and then use the Object browser to navigate to **Limits Dashboard Objects**, then **chk_ChartType2 Properties** and then double-click **Checked**.

```
switch (chk_ChartType2.Checked)
{
}
```

The Description pane shows Property Checked as Boolean. Since there are two Boolean values (*true* and *false*), we will provide two case values.

5 In the body of the *switch*, add the case for a value of true, and the statement to change the Limits Chart to a Line chart.

a. Click inside the body of the switch (on the blank line between the curly brackets), type **case true :** and a return [**Enter**].

b. Navigate to **Application**, then **ActiveDocument**, then **Sections**, then **Limits Chart** then **Properties** and then double-click **ChartType**.

The Description pane shows Property ChartType as BqChartType. Find the collection for BqChartType in the object model under Constants.

c. After **ChartType**, type an equal sign (=), then navigate to **Constants**, then **BqChartType**, then and double-click **bqChartTypeLine**.

d. Type a semicolon (;) at the end of the statement, and a return [**Enter**].

e. Type **break**; and two returns [**Enter**].

```
switch (chk_ChartType2.Checked)
{
case true :
ActiveDocument.Sections["Limits Chart"].ChartType=bqChartTypeLine;
break;
}
```

The case for true changes the chart to a line chart and ends with a break statement so other cases are ignored. The extra return is for readability.

6 **Add the case for a value of false, and the statement to change the Limits Chart to a pie chart.**

a. Click inside the body of the switch (on the blank line above the closing curly bracket), type **case false :** and a return [**Enter**].

b. Use the Object browser to navigate to **Application**, then **ActiveDocument**, then **Sections**, then **Limit Charts**, and then double-click **ChartType**.

c. After **ChartType**, type an equal sign (=), and then navigate to **Constants**, then **BqChartType**, and then double-click **bqChartTypePie**.

d. Type a semicolon (;) at the end of the statement, and a return [**Enter**].

e. Type **break**; and a return [**Enter**].

```
switch (chk_ChartType2.Checked)
{
case true :
ActiveDocument.Sections["Limits Chart"].ChartType=bqChartTypeLine;
break;
case false :
ActiveDocument.Sections["Limits Chart"].ChartType=bqChartTypePie;
break;
}
```

Verify that there is a close curly bracket after the last case.

7 **Click OK to save the script and close the Script Editor.**

8 **Toggle to Run mode to test the script.**

The chart should work the same with the switch as with the if...else logic. When the check box is selected, the chart is a line chart; when the check box is cleared, the chart is a pie chart.

Example

DropDown1 (under Select View) of The *Plan and Actual* section of `Sample2mod.bqy` allows the user to change the Costs, Sold, and Revenue charts to display results in terms of Planned vs. Actual, Planned, or Actual.

The OnClick event for DropDown1 creates a variable for the user choice. Then, depending on the value of *choice,* the JavaScript goes through each chart and removes all facts, and adds the appropriate facts. This is done with an if...else control structure.

In Design mode, with the Console window closed, copy and paste DropDown1 and rename the new one DropDown1_switch. Change the if...else control structure to a switch. (See "Controlling Chart Facts with if...else" on page 107 and "Controlling Chart Facts with switch" on page 108 for the finished JavaScript scripts.)

Controlling Chart Facts with if...else

The JavaScript script for DropDown1, *Plan and Actual* section of `Sample2mod.bqy` is:

```
var choice=ActiveDocument.Sections["Plan and
Actual"].Shapes.DropDown1[DropDown1.SelectedIndex];
if (choice=='Planned vs. Actual')
{
ActiveDocument.Sections["PlanActualCostsChart"].Facts.RemoveAll();
ActiveDocument.Sections["PlanActualCostsChart"].Facts.Add('Costs Plan');
ActiveDocument.Sections["PlanActualCostsChart"].Facts.Add('Costs Actual');
ActiveDocument.Sections["PlanActualSoldChart"].Facts.RemoveAll();
ActiveDocument.Sections["PlanActualSoldChart"].Facts.Add('Units Sold Plan');
ActiveDocument.Sections["PlanActualSoldChart"].Facts.Add('Units Sold Actual');
ActiveDocument.Sections["PlanActualRevenueChart"].Facts.RemoveAll();
ActiveDocument.Sections["PlanActualRevenueChart"].Facts.Add('Revenue Plan');
ActiveDocument.Sections["PlanActualRevenueChart"].Facts.Add('Revenue Actual');
}
else
if (choice=='Planned')
{
ActiveDocument.Sections["PlanActualCostsChart"].Facts.RemoveAll();
ActiveDocument.Sections["PlanActualCostsChart"].Facts.Add('Costs Plan');
ActiveDocument.Sections["PlanActualSoldChart"].Facts.RemoveAll();
ActiveDocument.Sections["PlanActualSoldChart"].Facts.Add('Units Sold Plan');
ActiveDocument.Sections["PlanActualRevenueChart"].Facts.RemoveAll();
ActiveDocument.Sections["PlanActualRevenueChart"].Facts.Add('Revenue Plan');
}
else
if (choice=='Actual')
{
ActiveDocument.Sections["PlanActualCostsChart"].Facts.RemoveAll();
ActiveDocument.Sections["PlanActualCostsChart"].Facts.Add('Costs Actual');
ActiveDocument.Sections["PlanActualSoldChart"].Facts.RemoveAll();
ActiveDocument.Sections["PlanActualSoldChart"].Facts.Add('Units Sold Actual');
ActiveDocument.Sections["PlanActualRevenueChart"].Facts.RemoveAll();
ActiveDocument.Sections["PlanActualRevenueChart"].Facts.Add('Revenue Actual');
}
```

Controlling Chart Facts with switch

The JavaScript for DropDown1_switch, *Plan and Actual* section of `Sample2mod.bqy` is:

```
var choice=ActiveDocument.Sections["Plan and
Actual"].Shapes.DropDown1_switch[DropDown1_switch.SelectedIndex];
switch (choice)
{
  case 'Planned vs. Actual':
ActiveDocument.Sections["PlanActualCostsChart"].Facts.RemoveAll();
ActiveDocument.Sections["PlanActualCostsChart"].Facts.Add('Costs Plan');
ActiveDocument.Sections["PlanActualCostsChart"].Facts.Add('Costs Actual');
ActiveDocument.Sections["PlanActualSoldChart"].Facts.RemoveAll();
ActiveDocument.Sections["PlanActualSoldChart"].Facts.Add('Units Sold Plan');
ActiveDocument.Sections["PlanActualSoldChart"].Facts.Add('Units Sold Actual');
ActiveDocument.Sections["PlanActualRevenueChart"].Facts.RemoveAll();
ActiveDocument.Sections["PlanActualRevenueChart"].Facts.Add('Revenue Plan');
ActiveDocument.Sections["PlanActualRevenueChart"].Facts.Add('Revenue Actual');
   break;
   case 'Planned':
ActiveDocument.Sections["PlanActualCostsChart"].Facts.RemoveAll();
ActiveDocument.Sections["PlanActualCostsChart"].Facts.Add('Costs Plan');
ActiveDocument.Sections["PlanActualSoldChart"].Facts.RemoveAll();
ActiveDocument.Sections["PlanActualSoldChart"].Facts.Add('Units Sold Plan');
ActiveDocument.Sections["PlanActualRevenueChart"].Facts.RemoveAll();
ActiveDocument.Sections["PlanActualRevenueChart"].Facts.Add('Revenue Plan');
   break;
   case 'Actual' :
ActiveDocument.Sections["PlanActualCostsChart"].Facts.RemoveAll();
ActiveDocument.Sections["PlanActualCostsChart"].Facts.Add('Costs Actual');
ActiveDocument.Sections["PlanActualSoldChart"].Facts.RemoveAll();
ActiveDocument.Sections["PlanActualSoldChart"].Facts.Add('Units Sold Actual');
ActiveDocument.Sections["PlanActualRevenueChart"].Facts.RemoveAll();
ActiveDocument.Sections["PlanActualRevenueChart"].Facts.Add('Revenue Actual');
   break;
}
```

7

JavaScript Operators

In This Chapter

This section provides detailed information on JavaScript operators and operator precedence.

Arithmetic Operators

Arithmetic operators, described in the following table, take numerical values (either literals or variables) as their operands and return a single numerical value.

Operator	Description
+	(Addition) Adds 2 numbers.
++	(Increment) Adds one to a variable representing a number (returning either the new or old value of the variable). The increment operator is used as follows: `var++ or ++var` The increment operator increments (adds one to) its operand and returns a value. If it is used postfix, with operator after operand (for example, x++), then it returns the value before incrementing. If it is used prefix with operator before operand (for example, ++x), then it returns the value after incrementing. For example, if x is three, then the statement y = x++ sets y to 3 and increments x to 4. If x is 3, then the statement y = ++x increments x to 4 and sets y to 4.

Operator	Description
--	(Decrement) Subtracts one from a variable representing a number (returning either the new or old value of the variable). The decrement operator is used as follows: `var-- or --var` The decrement operator decrements (subtracts one from) its operand and returns a value. If it is used postfix (for example, x--), then it returns the value before decrementing. If it is used prefix (for example, --x), then it returns the value after decrementing. For example, if x is three, then the statement y = x-- sets y to 3 and decrements x to 2. If x is 3, then the statement y = --x decrements x to 2 and sets y to 2.
-	(Unary negation, subtraction) As a unary operator, negates the value of its argument. As a binary operator, subtracts two numbers. The unary negation operator precedes its operand and negates it. For example, `y = -x` negates the value of `x` and assigns that to `y`; that is, if `x` were 3, `y` would get the value -3 and `x` would retain the value 3.
*	(Multiplication) Multiplies two numbers.
/	(Division) Divides two numbers.
%	(Modulus) Computes the integer remainder of dividing two numbers. The modulus operator is used as follows: `var1 % var2` The modulus operator returns the first operand modulo the second operand, that is, `var1` modulo `var2`, in the preceding statement, where `var1` and `var2` are variables. The modulo function is the integer remainder of dividing `var1` by `var2`. For example, 12 % 5 returns 2.

Assignment Operators

Assignment operators assign a value to a left operand based on the value of a right operand. The basic assignment operators are described in the Assignment Operators table. The other assignment operators, described in the Shorthand Assignment Operators table, are shorthand for standard operations.

Operator	Description
=	Assigns the value of the second operand to the first operand.
+=	Adds two numbers and assigns the result to the first.
-=	Subtracts two numbers and assigns the result to the first.
*=	Multiplies two numbers and assigns the result to the first.
/=	Divides two numbers and assigns the result to the first.
%=	Computes the modulus of two numbers and assigns the result to the first.
&=	Performs a bitwise AND and assigns the result to the first operand.
^=	Performs a bitwise XOR and assigns the result to the first operand.

Operator	Description
|=	Performs a bitwise OR and assigns the result to the first operand.
>>=	Performs a sign-propagating right shift and assigns the result to the first operand.
>>>=	Performs a zero-fill right shift and assigns the result to the first operand.

Shorthand Operator	Meaning
x += y	x = x + y
x -= y	x = x – y
x *= y	x = x * y
x /= y	x = x / y
x %= y	x = x % y
x <<= y	x = x << y
x >>= y	x = x >> y
x >>>= y	x = x >>> y
x &= y	x = x & y
x ^= y	x = x ^ y
x |= y	x = x | y

Bitwise Operators

Bitwise operators, described in the following table, treat their operands as a set of bits (zeros and ones), rather than as decimal, hexadecimal, or octal numbers. For example, the decimal number nine has a binary representation of 1001. Bitwise operators perform their operations on such binary representations, but they return standard JavaScript numerical values.

Operator	Description
&	(Bitwise AND) Returns a one in each bit position if bits of both operands are ones. The Bitwise AND operator is used as follows: `a & b` Returns a one in each bit position if bits of both operands are ones.
^	(Bitwise XOR) Returns a one in a bit position if bits of one, but not if both operands are one. The bitwise XOR operator is used as follows: `a ^ b` Returns a one in a bit position if bits of one, but not both operands are one.

Operator	Description
|	(Bitwise OR) Returns a one in a bit if bits of either operand is one. The Bitwise OR operator is used as follows: `a | b` Returns a one in a bit if bits of either operand is one.
~	(Bitwise NOT) Flips the bits of its operand. The Bitwise NOT operator is used as follows: `~ a` Flips the bits of its operand.
<<	(Left shift) Shifts its first operand in binary representation the number of bits to the left specified in the second operand, shifting in zeros from the right. The Left shift operator is used as follows: `a << b` Shifts a in binary representation b bits to left, shifting in zeros from the right.
>>	(Sign-propagating right shift) Shifts the first operand in binary representation the number of bits to the right specified in the second operand, discarding bits shifted off. The Sign-propagating right shift operator is used as follows: `a >> b` Shifts a in binary representation b bits to right, discarding bits shifted off.
>>>	(Zero-fill right shift) Shifts the first operand in binary representation the number of bits to the right specified in the second operand, discarding bits shifted off, and shifting in zeros from the left. The zero-fill right shift operator is used as follows: `a >>> b` Shifts a in binary representation b bits to the right, discarding bits shifted off, and shifting in zeros from the left.

Bitwise Logical Operators

Conceptually, the bitwise logical operators work as follows:

1. The operands are converted to thirty-two-bit integers and expressed by a series of bits (zeros and ones).
2. Each bit in the first operand is paired with the corresponding bit in the second operand: first bit to first bit, second bit to second bit, and so on.
3. The operator is applied to each pair of bits, and the result is constructed bitwise.

For example, the binary representation of nine is 1001, and the binary representation of fifteen is 1111. So, when the bitwise operators are applied to these values, the results are as follows:

```
15 & 9 yields 9 (1111 & 1001 = 1001)
15 | 9 yields 15 (1111 | 1001 = 1111)
15 ^ 9 yields 6 (1111 ^ 1001 = 0110)
```

Bitwise Shift Operators

The bitwise shift operators, described in the following table, take two operands: the first is a quantity to be shifted, and the second specifies the number of bit positions by which the first operand is to be shifted. The operator used controls the direction of the shift operation

Shift operators convert their operands to thirty-two-bit integers and return a result of the same type as the left operator.

Operator	Description
<< (Left Shift)	This operator shifts the first operand the specified number of bits to the left. Excess bits shifted off to the left are discarded. Zero bits are shifted in from the right. For example, `9<<2` yields thirty-six, because 1001 shifted two bits to the left becomes 100100, which is thirty-six.
>> (Sign-Propagating Right Shift)	This operator shifts the first operand the specified number of bits to the right. Excess bits shifted off to the right are discarded. Copies of the leftmost bit are shifted in from the left. For example, `9>>2` yields two, because 1001 shifted two bits to the right becomes 10, which is two. Likewise, `-9>>2` yields -3, because the sign is preserved.
>>> (Zero-Fill Right Shift)	This operator shifts the first operand the specified number of bits to the right. Excess bits shifted off to the right are discarded. Zero bits are shifted in from the left. For example, `19>>>2` yields four, because 10011 shifted two bits to the right becomes 100, which is four. For non-negative numbers, zero-fill right shift and sign-propagating right shift yield the same result.

Comparison Operators

A comparison operator compares its operands and returns a logical value based on whether the comparison is true or not. The operands can be numerical or string values. When used on string values, the comparisons are based on the standard lexicographical ordering.

The following table describes the comparison operators. It assume `var1` has been assigned the value 3 and `var2` has been assigned the value 4.

Operator	Description
==	(Equal) Returns true if the operands are equal. For example: `3 == var1`
!=	(Not equal) Returns true if the operands are not equal. For example: `var1 != 4`
>	(Greater than) Returns true if left operand is greater than right operand. For example: `var2 > var1`

>=	(Greater than or equal) Returns true if left operand is greater than or equal to right operand. For example:
	`var2 >= var1`
	`var1 >= 3`

Operator	Description
<	(Less than) Returns true if left operand is less than right operand. For example: `var1 < var2`

<=	(Less than or equal) Returns true if left operand is less than or equal to right operand. For example:
	`var1 <= var2`
	`var2 <= 5`

Logical Operators

Logical operators, described in the following, take Boolean (logical) values as operands and return a Boolean value.

Operator	Description
&&	(Logical AND) Returns true if both logical operands are true. Otherwise, returns false. The Logical AND operator is used as follows: `expr1 && expr2` Returns `expr1` if it converts to false. Otherwise, returns `expr2`.
||	(Logical OR) Returns true if either logical expression is true. If both are false, returns false. The Logical OR operator is used as follows: `expr1 || expr2` Returns `expr1` if it converts to true. Otherwise, returns `expr2`.
!	(Logical negation) If its single operand is true, returns false; otherwise, returns true.

Example

Consider the following script:

```
v1 = "Cat";
v2 = "Dog";
v3 = false;
Console.Write("t && t returns " + (v1 && v2));
Console.Write("f && t returns " + (v3 && v1));
Console.Write("t && f returns " + (v1 && v3));
Console.Write("f && f returns " + (v3 && (3 == 4)));
Console.Write("t || t returns " + (v1 || v2));
Console.Write("f || t returns " + (v3 || v1));
Console.Write("t || f returns " + (v1 || v3));
Console.Write("f || f returns " + (v3 || (3 == 4)));
Console.Write("!t returns " + (!v1));

Console.Write("!f returns " + (!v3));
```

This script displays the following:

```
t && t returns Dog
```

```
f && t returns false
t && f returns false
f && f returns false
t || t returns Cat
f || t returns Cat
t || f returns Cat
f || f returns false
!t returns false
!f returns true
```

Short-Circuit Evaluation

As logical expressions are evaluated left to right, they are tested for possible "short-circuit" evaluation using these rules:

```
false && anything is short-circuit evaluated to false.
true || anything is short-circuit evaluated to true.
```

The rules of logic guarantee that these evaluations are always correct. Note that the `anything` part of the above expressions is not evaluated, so any side effects of doing so do not take effect.

String Operators

Use the concatenation operator (+) to concatenate two string values together and a return another string that is the union of the two operand strings. For example, `"my " + "string"` returns the string `"my string"`.

The shorthand assignment operator += can also be used to concatenate strings. For example, if the variable *mystring* has the value "alpha," then the expression `mystring += "bet"` evaluates to "alphabet" and assigns this value to *mystring*.

The following table describes the string operators.

Operator	Description
+	(String addition) Concatenates two strings.
+=	Concatenates two strings and assigns the result to the first operand.

Special Operators

This section explains the syntax, parameters, and descriptions for the special operators used in JavaScript, which are listed in the following table

Operator	Description
?: (Conditional operator)	Lets you perform a simple "if...then...else"
, (comma operator)	Evaluates two expressions and returns the result of the second expression.

Operator	Description
delete	Lets you delete an object property or an element at a specified index in an array.
new	Lets you create an instance of a user-defined object type or of one of the built-in object types.
this	Keyword that you can use to refer to the current object.
typeof	Returns a string indicating the type of the unevaluated operand.
void	Specifies an expression to be evaluated without returning a value.

?: (Conditional operator)

The `conditional` operator is the only JavaScript operator that takes three operands. This operator is frequently used as a shortcut for the *if* statement.

Syntax

```
condition ? expr1 : expr2
```

Parameters

`Condition`—An expression that evaluates to either true or false.

`expr1, expr2`—Expressions with values of any type.

Description

If condition is true, the operator returns the value of `expr1`; otherwise, it returns the value of `expr2`. For example, to display a different message based on the value of the *isMember* variable, you could use this statement:

```
Console.Write ("The fee is " + (isMember ? "$2.00" : "$10.00"))
```

, (comma operator)

The `comma` operator evaluates both of its operands and returns the value of the second operand.

Syntax

```
expr1, expr2
```

Parameters

`expr1, expr2`—Any expressions

Description

You can use the comma operator when you want to include multiple expressions in a location that requires a single expression. The most common usage of this operator is to supply multiple parameters in a for loop.

For example, if a is a two-dimensional array with 10 elements on a side, the following code uses the comma operator to increment two variables at once. The code prints the values of the diagonal elements in the array:

```
  for (var i=0, j=10; i <= 10; i++, j--)
Console.Write("a["+i+","+j+"]= " + a[i,j])
```

delete

The delete operator deletes an object's property or an element at a specified index in an array.

Syntax

```
delete objectName.property
delete objectName[index]
delete property
```

Parameters

objectName—The name of an object.

property—An existing property.

index—An integer representing the location of an element in an array.

Description

The third form is legal only within a with statement.

If the deletion succeeds, the delete operator sets the property or element to undefined. delete always returns undefined.

new

The new operator lets you create an instance of a user-defined object type or of one of the built-in object types that has a constructor function.

Syntax

```
objectName = new objectType (param1 [,param2] ...[,paramN])
```

Arguments

objectName—Name of the new object instance.

objectType—Must be a function that defines an object type.

param1...paramN—Property values for the object. These properties are parameters defined for the objecType function.

Description

Creating a user-defined object type requires two steps:

1. Define the object type by writing a function.
2. Create an instance of the object with `new`.

To define an object type, create a function for the object type that specifies its name, properties, and methods. An object can have a property that is itself another object. See the examples that follow.

You can always add a property to a previously defined object. For example, the statement `car1.color = "black"` adds a property `color` to `car1`, and assigns it a value of *black*. However, this does not affect any other objects. To add the new property to all objects of the same type, you must add the property to the definition of the `car` object type.

You can add a property to a previously defined object type by using the `Function.prototype` property. This defines a property that is shared by all objects created with that function, rather than by just one instance of the object type. The following code adds a `color` property to all objects of type `car`, and then assigns a value to the `color` property of the object **car1**.

```
Car.prototype.color=null
car1.color="black"
birthday.description="The day you were born"
```

Examples

Example1: object type and object instance. Suppose you want to create an object type for cars. You want this type of object to be called `car`, and you want it to have properties for make, model, and year. To do this, you would write the following function:

```
function car(make, model, year) {
  this.make = make
  this.model = model
  this.year = year
}
```

Now you can create an object called `mycar` as follows:

```
mycar = new car("Eagle", "Talon TSi", 1993)
```

This statement creates `mycar` and assigns it the specified values for its properties. Then the value of `mycar.make` is the string “`Eagle,`” `mycar.year` is the integer `1993`, and so on.

You can create any number of `car` objects by calls to `new`. For example,

```
kenscar = new car("Nissan", "300ZX", 1992)
```

Example2: object property that is itself another object. Suppose you define an object called person as follows:

```
function person(name, age, sex) {
  this.name = name
  this.age = age
  this.sex = sex
}
```

And then instantiate two new `person` objects as follows:

```
rand = new person("Rand McNally", 33, "M")
ken = new person("Ken Jones", 39, "M")
```

Then you can rewrite the definition of car to include an owner property that takes a person object, as follows:

```
function car(make, model, year, owner) {
  this.make = make;
  this.model = model;
  this.year = year;
  this.owner = owner;
}
```

To instantiate the new objects, you then use the following:

```
car1 = new car("Eagle", "Talon TSi", 1993, rand);
car2 = new car("Nissan", "300ZX", 1992, ken)
```

Instead of passing a literal string or integer value when creating the new objects, the above statements pass the objects `rand` and `ken` as the parameters for the owners. To find out the name of the owner of `car2`, you can access the following property:

```
car2.owner.name
```

this

A keyword that you can use to refer to the current object. In general, in a method `this` refers to the calling object.

Syntax

```
this[.propertyName]
```

Examples

Suppose a function called `validate` validates an object's value property, given the object and the high and low values:

```
function validate(obj, lowval, hival) {
  if ((obj.value < lowval) || (obj.value > hival))
    Alert("Invalid Value!")
}
```

typeof

The typeof operator is used in either of the following ways:

- `typeof operand`
- `typeof (operand)`

The `typeof` operator returns a string indicating the type of the unevaluated operand. operand is the string, variable, keyword, or object for which the type is to be returned. The parentheses are optional.

Suppose you define the following variables:

```
var myFun = new Function("5+2")
var shape="round"
var size=1
var today=new Date()
```

The `typeof` operator returns these results:

```
typeof myFun is object
typeof shape is string
typeof size is number
typeof today is object
typeof dontExist is undefined
```

For the keywords `true` and `null`, the `typeof` operator returns these results:

```
typeof true is boolean
typeof null is object
```

For a number or string, the `typeof` operator returns these results:

```
typeof 62 is number
typeof 'Hello world' is string
```

For property values, the `typeof` operator returns the type of value the property contains:

```
typeof document.lastModified is string
typeof window.length is number
typeof Math.LN2 is number
```

For methods and functions, the `typeof` operator returns results as follows:

```
typeof blur is function
typeof eval is function
typeof parseInt is function
typeof shape.split is function
```

For predefined objects, the `typeof` operator returns results as follows:

```
typeof Date is function
typeof Function is function
typeof Math is function
typeof Option is function
typeof String is function
```

void

The void operator is used in either of the following ways:

- `void (expression)`
- `void expression`

The void operator specifies an expression to be evaluated without returning a value. `expression` is a JavaScript expression to evaluate. The parentheses surrounding the expression are optional, but it is good style to use them.

8 Core Objects

In This Chapter

This section provides detailed descriptions of the JavaScript core objects, which are summarized in the following table.

Object	Description
Array	Represents an array.
Boolean	Represents a Boolean value.
Date	Represents a date.
Function	Specifies a string of JavaScript code to be compiled as a function.
Math	Provides basic math constants and functions; for example, its PI property contains the value of pi.
Number	Represents primitive numeric values.
Object	Contains the base functionality shared by all JavaScript objects.

Object	Description
String	Represents a JavaScript string.
Regular Expression	Represents a regular expression; also contains static properties that are shared among all regular expression objects.

Array

Function

An array allows you to store a list of common elements in a variable as shown in the following example:

```
var models = new Array("Ford", "Mazda", "Honda");
```

You can easily access the elements of an array by using the index number assigned to each element. Elements are stored in sequential order beginning with index number 0, proceeding with index number 1, and so on. Since the index numbering begins with 0, the array's item count will always be one higher than the highest value of the array. The element's index number is enclosed in square brackets and constitutes its location in the array. The Array is a core object.

To set the first element of the array in the example shown above, you would type:

```
models[0];
```

When you execute the JavaScript, the variable will contain the "Ford" string.

Created by

The Array object constructor:

```
new Array(arrayLength);
new Array(element0, element1, ..., elementN);
```

Parameters

`arrayLength`

(Optional) The initial length of the array. You can access this value using the `length` property.

`element`

(Optional) A list of values for the array's elements. When this form is specified, the array is initialized with the specified values as its elements, and the array's length property is set to the number of arguments.

Description

An array's length increases if you assign a value to an element higher than the current length of the array. The following code creates an array of length 0, then assigns a value to element 99. This changes the length of the array to 100.

```
colors = new Array()
colors[99] = "midnightblue"
```

You can construct a *dense* array of two or more elements starting with index 0 if you define initial values for all elements. A dense array is one in which each element has a value. The following code creates a dense array with three elements:

```
myArray = new Array("Hello", myVar, 3.14159)
```

The result of a match between a regular expression and a string can create an array. This array has properties and elements that provide information about the match. An array is the return value of `RegExp.exec, String.match`, and `String.replace`.

To help explain these properties and elements, look at the following example and then refer to the table below:

```
//Match one d followed by one or more b's followed by one d
//Remember matched b's and the following d
//Ignore case
myRe=/d(b+)(d)/i;
myArray = myRe.exec("cdbBdbsbz");
```

The following table lists the properties and elements returned from this match.

Property/Element	Description	Example
Input	A read-only property that reflects the original string against which the regular expression was matched.	CdbBdbsbz
Index	A read-only property that is the zero-based index of the match in the string.	1
[0]	A read-only element that specifies the last matched characters.	DbBd
[1], ...[n]	Read-only elements that specify the parenthesized substring matches, if included in the regular expression. The number of possible parenthesized substrings is unlimited.	[1]=bB [2]=d

Examples

The following example creates an array, `msgArray`, with a length of 0, then assigns values to `msgArray[0]` and `msgArray[99]`, changing the length of the array to 100.

```
msgArray = new Array()
msgArray [0] = "Hello"
msgArray [99] = "world"
// The following statement is true,
// because defined msgArray [99] element.
if (msgArray.length == 100)
      Console.Write("The length is 100.")
```

The following code creates a two-dimensional array and displays the results.

```
a = new Array(4)
for (i=0; i < 4; i++) {a[i] = new Array(4)
for (j=0; j < 4; j++)
{a[i][j] = "["+i+","+j+"]"}
}
```

```
for (i=0; i < 4; i++)
{str = "\r\nRow "+i+":"
for (j=0; j < 4; j++)
{str += a[i][j]}
Console.Write(str)
}
```

This example displays the following results:

```
Multidimensional array test
Row 0:[0,0][0,1][0,2][0,3]
Row 1:[1,0][1,1][1,2][1,3]
Row 2:[2,0][2,1][2,2][2,3]
Row 3:[3,0][3,1][3,2][3,3]
```

Array Properties

The following table displays a summary of the array properties. Detailed descriptions of each property follow the table.

Property	Description
index	For an array created by a regular expression match, the zero-based index of the match in the string.
input	For an array created by a regular expression match, reflects the original string against which the regular expression was matched.
length	Reflects the number of elements in an array.
prototype	Allows the addition of properties to an `Array` object.

index

Property of

`Array`

Description

For an array created by a regular expression match, the zero-based index of the match in the string. The `index` property is *static*.

input

Property of

`Array`

Description

For an array created by a regular expression mathc, reflects the original string against which the regular expression was matched. The `input` property is *static*.

length

Property of

Array

Description

An integer that specifies the number of elements in an array. You can set the length property to truncate an array at any time. You cannot extend an array; for example, if you set length to 3 when it is currently 2, the array will still contain only 2 elements. The `length` property is *static*.

Examples

In the following example, the `getChoice` function uses the `length` property to iterate over every element in the `musicType` `array. musicType` is a select element on the `musicForm` form.

```
function getChoice() {
      for (var i = 0; i < document.musicForm.musicType.length; i++) {
            if (document.musicForm.musicType.options[i].selected == true) {
                  return document.musicForm.musicType.options[i].text
            }
      }
}
```

The following example shortens the array `statesUS` to a length of 50 if the current length is greater than 50.

```
if (statesUS.length > 50) {
      statesUS.length=50
      alert("The U.S. has only 50 states. New length is " + statesUS.length)
}
```

prototype

Property of

Array

Description

Represents the prototype for this class. You can use the prototype to add properties or methods to all instances of a class.

Array Methods

The following table displays a summary of the array methods. Detailed descriptions of each method follow the table.

Method	Description
concat	Joins two arrays and returns a new array.
join	Joins all elements of an array into a string.
pop	Removes the last element from an array and returns that element.
push	Adds one or more elements to the end of an array and returns that last element added.
reverse	Transposes the elements of an array: the first array element becomes the last and the last becomes the first.
shift	Removes the first element from an array and returns that element.
slice	Extracts a section of an array and returns a new array.
splice	Adds and/or removes elements from an array.
sort	Sorts the elements of an array.
toString	Returns a string representing the specified object.
unshift	Adds one or more elements to the front of an array and returns the new length of the array.

concat

Joins two arrays and returns a new array.

Applies to

`Array`

Syntax

```
concat(arrayName2)
```

Parameters

`ArrayName2`

Name of the array to concatenate to this array.

Description

`concat` does not alter the original arrays, but returns a *one level deep* copy that contains copies of the same elements combined from the original arrays. Elements of the original arrays are copied into the new array as follows:

Object references (and not the actual object) — `concat` copies object references into the new array. Both the original and new array refer to the same object. If a referenced object changes, the changes are visible to both the new and original arrays.

Strings and numbers (not `String` and `Number` objects) — `concat` copies strings and numbers into the new array. Changes to the string or number in one array do not affect the other arrays.

If a new element is added to either array, the other array is not affected.

join

Joins all elements of an array into a string.

Applies to

`Array`

Syntax

`join(separator)`

Parameters

`separator`

Specifies a string to separate each element of the array. The separator is converted to a string if necessary. If omitted, the array elements are separated with a comma.

Description

The string conversion of all array elements are joined into one string.

Examples

The following example creates an array with three elements, then joins the array three times: using the default separator, then a comma and a space, and then a plus.

```
a = new Array("Wind","Rain","Fire")
Console.Write(a.join())
Console.Write(a.join(", "))
Console.Write(a.join(" + "))
```

This code produces the following output:

```
Wind,Rain,Fire
Wind, Rain, Fire
Wind + Rain + Fire
```

See also

`Array: reverse`

pop

Removes the last element from an array and returns that element. This method changes the length of the array.

Applies to

`Array`

Syntax

```
pop()
```

Parameters

None

Example

The following code displays the `myFish` array before and after removing its last element. It also displays the removed element:

```
myFish = ["angel", "clown", "mandarin", "surgeon"];
Console.Write("\r\nmyFish before: " + myFish);
popped = myFish.pop();
Console.Write("\r\nmyFish after: " + myFish);
Console.Write("\r\npopped this element: " + popped);
```

This example displays the following:

```
myFish before: ["angel", "clown", "mandarin", "surgeon"]
myFish after: ["angel", "clown", "mandarin"]
popped this element: surgeon
```

See also

`Array: push`, `Array: shift`, `Array: unshift`

push

Adds one or more elements to the end of an array and returns that last element added. This method changes the length of the array.

Applies to

`Array`

Syntax

```
push(elt1, ..., eltN)
```

Parameters

`elt1,...eltN`

The elements to add to the end of the array.

Description

The behavior of the `push` method is analogous to the push function in Perl 4. Note that this behavior is different in Perl 5.

Example

The following code displays the `myFish` array before and after adding elements to its end. It also displays the last element added:

```
myFish = ["angel", "clown"];
Console.Write("myFish before: " + myFish);
pushed = myFish.push("drum", "lion");
Console.Write("myFish after: " + myFish);
Console.Write("pushed this element last: " + pushed);
```

This example displays the following:

```
myFish before: ["angel", "clown"]
myFish after: ["angel", "clown", "drum", "lion"]
pushed this element last: lion
```

See also

`Array: pop, Array: shift, Array: unshift`

reverse

Transposes the elements of an array: the first array element becomes the last and the last becomes the first.

Applies to

`Array`

Syntax

`reverse()`

Parameters

None

Description

The reverse method transposes the elements of the calling array object.

Examples

The following example creates an array `myArray`, containing three elements, then reverses the array.

```
myArray = new Array("one", "two", "three")
myArray.reverse()
```

The output is as follows

```
myArray[0] is "three"
myArray[1] is "two"
myArray[2] is "one"
```

See also

`Array: join, Array: sort`

shift

Removes the first element from an array and returns that element. This method changes the length of the array.

Applies to

`Array`

Syntax

`shift()`

Parameters

None

Example

The following code displays the `myFish` array before and after removing its first element. It also displays the removed element:

```
myFish = ["angel", "clown", "mandarin", "surgeon"];
Console.Write("myFish before: " + myFish);
shifted = myFish.shift();
Console.Write("myFish after: " + myFish);
Console.Write("Removed this element: " + shifted);
```

This example displays the following:

```
myFish before: ["angel", "clown", "mandarin", "surgeon"]
myFish after: ["clown", "mandarin", "surgeon"]
Removed this element: angel
```

See also

`Array: pop, Array: push, Array: unshift`

slice

Extracts a section of an array and returns a new array.

Applies to

`Array`

Syntax

`slice(begin,end)`

Parameters

`Begin`

Zero-based index at which to begin extraction.

`End`

(Optional) Zero-based index at which to end extraction. `slice` extracts up to but not including `end. slice(1,4)` extracts the second element through the fourth element (elements indexed 1, 2, and 3). As a negative index, end indicates an offset from the end of the sequence. `slice(2,-1)` extracts the third element through the second to last element in the sequence. If end is omitted, `slice` extracts to the end of the sequence.

Description

`slice` does not alter the original array, but returns a new "one level deep" copy that contains copies of the elements sliced from the original array. Elements of the original array are copied into the new array as follows:

Object references (and not the actual object) -- `slice` copies object references into the new array. Both the original and new array refer to the same object. If a referenced object changes, the changes are visible to both the new and original arrays.

Strings and numbers (not `String` and `Number` objects)-- `slice` copies strings and numbers into the new array. Changes to the string or number in one array does not affect the other array.

If a new element is added to either array, the other array is not affected.

Example

In the following example, `slice` creates a new array, `newCar`, from `myCar`. Both include a reference to the object `myHonda`. When the color of `myHonda` is changed to `purple`, both arrays reflect the change.

```
//Using slice, create newCar from myCar.
myHonda = {color:"red",wheels:4,engine:{cylinders:4,size:2.2}}
myCar = [myHonda, 2, "cherry condition", "purchased 1997"]
newCar = myCar.slice(0,2)
//Write the values of myCar, newCar, and the color of myHonda
// referenced from both arrays.
Console.Write("myCar = " + myCar)
Console.Write("newCar = " + newCar)
Console.Write("myCar[0].color = " + myCar[0].color)
Console.Write("newCar[0].color = " + newCar[0].color)
//Change the color of myHonda.
myHonda.color = "purple"
Console.Write("The new color of my Honda is " + myHonda.color)
//Write the color of myHonda referenced from both arrays.
Console.Write("myCar[0].color = " + myCar[0].color)
Console.Write("newCar[0].color = " + newCar[0].color)
```

This script writes:

```
myCar = [{color:"red", wheels:4, engine:{cylinders:4, size:2.2}}, 2
      "cherry condition", "purchased 1997"]
newCar = [{color:"red", wheels:4, engine:{cylinders:4, size:2.2}}, 2]
myCar[0].color = red newCar[0].color = red
The new color of my Honda is purple
myCar[0].color = purple
newCar[0].color = purple
```

splice

Changes the content of an array, adding new elements while removing old elements.

Applies to

`Array`

Syntax

`splice(index, howMany, newElt1, ..., newElt`*`N`*`)`

Parameters

`index`

Index at which to start changing the array.

`howMany`

An integer indicating the number of old array elements to remove. If howMany is 0, no elements are removed. In this case, you should specify at least one new element.

`newElt1...newEltN`

(Optional) The elements to add to the array. If you don't specify any elements, splice simply removes elements from the array.

Description

If you specify a different number of elements to insert than the number you're removing, the array will have a different length at the end of the call. If `howMany` is 1, this method returns the single element that it removes. If `howMany` is more than 1, the method returns an array containing the removed elements.

Examples

The following script illustrates the use of the `splice`:

```
myFish = ["angel", "clown", "mandarin", "surgeon"];
Console.Write("myFish: " + myFish);
removed = myFish.splice(2, 0, "drum");
Console.Write("After adding 1: " + myFish);
Console.Write("removed is: " + removed);
removed = myFish.splice(3, 1)
Console.Write("After removing 1: " + myFish);
Console.Write("removed is: " + removed);
removed = myFish.splice(2, 1, "trumpet")
Console.Write("After replacing 1: " + myFish);
Console.Write("removed is: " + removed);
removed = myFish.splice(0, 2, "parrot", "anemone", "blue")
Console.Write("After replacing 2: " + myFish);
Console.Write("removed is: " + removed);
```

This script displays:

```
myFish: ["angel", "clown", "mandarin", "surgeon"]
After adding 1: ["angel", "clown", "drum", "mandarin", "surgeon"]
removed is: undefined
After removing 1: ["angel", "clown", "drum", "surgeon"]
removed is: mandarin
After replacing 1: ["angel", "clown", "trumpet", "surgeon"]
removed is: drum
After replacing 2: ["parrot", "anemone", "blue", "trumpet", "surgeon"]
removed is: ["angel", "clown"]
```

sort

Sorts the elements of an array.

Applies to

`Array`

Syntax

`sort(compareFunction)`

Parameters

`compareFunction`

Specifies a function that defines the sort order. If omitted, the array is sorted lexicographically (in dictionary order) according to the string conversion of each element.

Description

If `compareFunction` is not supplied, elements are sorted by converting them to strings and comparing strings in lexicographic ("dictionary" or "telephone book," *not* numerical) order. For example, "80" comes before "9" in lexicographic order, but in a numeric sort 9 comes before 80.

If `compareFunction` is supplied, the array elements are sorted according to the return value of the compare function. If a and b are two elements being compared, then:

- If `compareFunction(a, b)` is less than 0, sort `b` to a lower index than `a`.

- If `compareFunction(a, b)` returns 0, leave `a` and `b` unchanged with respect to each other, but sorted with respect to all different elements.
- If `compareFunction(a, b)` is greater than 0, sort `b` to a higher index than `a`.

So, the compare function has the following form:

```
function compare(a, b) {
      if (a is less than b by some ordering criterion)
            return -1
      if (a is greater than b by the ordering criterion)
            return 1
      // a must be equal to b
      return 0
}
```

To compare numbers instead of strings, the compare function can simply subtract b from a:

```
function compareNumbers(a, b) {
      return a - b
}
```

JavaScript uses a stable sort: the index partial order of a and b does not change if a and b are equal. If a's index was less than b's before sorting, it will be after sorting, no matter how a and b move due to sorting.

```
a = new Array();
a[0] = "Ant";
a[5] = "Zebra";
function writeArray(x) {
      for (i = 0; i < x.length; i++) {
            Console.Write(x[i]);
            if (i < x.length-1) Console.Write(", ");
      }
}
writeArray(a);
a.sort();
Console.Write();
writeArray(a);
ant, undefined, undefined, undefined, undefined, zebra
ant, zebra, undefined, undefined, undefined, undefined
```

Examples

The following example creates four arrays and displays the original array, then the sorted arrays. The numeric arrays are sorted without, then with, a compare function.

```
stringArray = new Array("Blue","Humpback","Beluga")
numericStringArray = new Array("80","9","700")
numberArray = new Array(40,1,5,200)
mixedNumericArray = new Array("80","9","700",40,1,5,200)
function compareNumbers(a, b) {
      return a - b
}
Console.Write("stringArray:" + stringArray.join())
Console.Write("Sorted:" + stringArray.sort())
Console.Write("numberArray:" + numberArray.join())
Console.Write("Sorted without a compare function:" + numberArray.sort())
Console.Write("Sorted with compareNumbers:" + numberArray.sort(compareNumbers))
Console.Write("numericStringArray:" + numericStringArray.join())
Console.Write("Sorted without a compare function:" + numericStringArray.sort())
Console.Write("Sorted with compareNumbers:" + numericStringArray.sort(compareNumbers))
Console.Write("mixedNumericArray:" + mixedNumericArray.join())
Console.Write("Sorted without a compare function:" + mixedNumericArray.sort())
Console.Write("Sorted with compareNumbers: " + mixedNumericArray.sort(compareNumbers))
```

This example produces the following output. As the output shows, when a compare function is used, numbers sort correctly whether they are numbers or numeric strings.

```
stringArray: Blue,Humpback,Beluga
Sorted: Beluga,Blue,Humpback
numberArray: 40,1,5,200
Sorted without a compare function: 1,200,40,5
Sorted with compareNumbers: 1,5,40,200
numericStringArray: 80,9,700
Sorted without a compare function: 700,80,9
Sorted with compareNumbers: 9,80,700
mixedNumericArray: 80,9,700,40,1,5,200
Sorted without a compare function: 1,200,40,5,700,80,9
Sorted with compareNumbers: 1,5,9,40,80,200,700
```

See also

`Array: join`, `Array: reverse`

toString

Returns a string representing the specified object.

Applies to

`Array`

Syntax

```
toString()
```

Parameters

None

Description

Every object has a `toString` method that is automatically called when it is to be represented as a text value or when an object is referred to in a string concatenation.

You can use `toString` within your own code to convert an object into a string, and you can create your own function to be called in place of the default `toString` method.

For `Array` objects, the built-in `toString` method joins the array and returns one string containing each array element separated by commas. For example, the following code creates an array and uses `toString` to convert the array to a string while writing output.

```
var monthNames = new Array("Jan","Feb","Mar","Apr")
Console.Write("monthNames.toString() is " + monthNames.toString())
```

The output is as follows:

```
monthNames.toString() is Jan,Feb,Mar,Apr
```

For information on defining your own `toString` method, see the `Object: toString` method.

unshift

Adds one or more elements to the beginning of an array and returns the new length of the array.

Applies to

`Array`

Syntax

`arrayName.unshift(elt1,..., eltN)`

Parameters

`elt1...eltN`

The elements to add to the front of the array.

Example

The following code displays the `myFish` array before and after adding elements to it.

```
myFish = ["angel", "clown"];
Console.Write("myFish before: " + myFish);
unshifted = myFish.unshift("drum", "lion");
Console.Write("myFish after: " + myFish);
Console.Write("New length: " + unshifted);
```

This example displays the following:

```
myFish before: ["angel", "clown"]
myFish after: ["drum", "lion", "angel", "clown"]
New length: 4
```

See also

`Array: pop`, `Array: push`, `Array: shift`

Boolean

The Boolean object is an object wrapper for a boolean value. The Boolean object is a core object.

Created by

The `Boolean` constructor:

```
new Boolean(value)
```

Parameters

`value`

The initial value of the Boolean object. The value is converted to a boolean value, if necessary. If value is omitted or is 0, null, false, or the empty string (""), the object has an initial value of false. All other values, including the string "false", create an object with an initial value of true.

Description

Use a `Boolean` object when you need to convert a non-boolean value to a boolean value. You can use the `Boolean` object any place JavaScript expects a primitive boolean value. JavaScript returns the primitive value of the `Boolean` object by automatically invoking the `valueOf` method.

Examples

The following examples create `Boolean` objects with an initial value of false:

```
bNoParam = new Boolean()
bZero = new Boolean(0)
bNull = new Boolean(null)
bEmptyString = new Boolean("")
bfalse = new Boolean(false)
```

The following examples create `Boolean` objects with an initial value of true:

```
btrue = new Boolean(true)
btrueString = new Boolean("true")
bfalseString = new Boolean("false")
bSuLin = new Boolean("Su Lin")
```

Boolean Properties

The following table displays the boolean property. A detailed description of the property follows the table.

Property	Description
Prototype	Defines a property that is shared by all Boolean objects.

prototype

Property of

`Boolean`

Description

Represents the prototype for this class. You can use the prototype to add properties or methods to all instances of a class.

Boolean Methods

The following table displays the boolean method. A detailed description of the method follows the table.

Method	Description
toString	Returns a string representing the specified object.

toString

Returns a string representing the specified object.

Applies to:

`Boolean`

Syntax

`toString()`

Parameters

None

Description

Every object has a `toString` method that is automatically called when it is to be represented as a text value or when an object is referred to in a string concatenation.

You can use `toString` within your own code to convert an object into a string, and you can create your own function to be called in place of the default `toString` method.

For Boolean objects and values, the built-in toString method returns "true" or "false" depending on the value of the boolean object.

```
flag.toString returns "true".
flag = new Boolean(true)
Console.Write("flag.toString() is " + flag.toString())
```

For information on defining your own `toString` method, see the `Object:` `toString` method.

Date

Lets you work with dates and times. Date is a core object.

Created by

The Date constructor:

`new Date()`

`new Date("month day, year hours:minutes:seconds")`

```
new Date(yr_num, mo_num, day_num)
```

```
new Date(yr_num, mo_num, day_num, hr_num, min_num, sec_num)
```

Parameters

`month, day, year, hours, minutes, seconds`

String values representing part of a date.

`yr_num, mo_num, day_num, hr_num, min_num, sec_num`

Integer values representing part of a date. As an integer value, the month is represented by 0 to 11 with 0=January and 11=December.

Description

If you supply no arguments, the constructor creates a `Date object` for today's date and time. If you supply some arguments, but not others, the missing arguments are set to 0. If you supply any arguments, you must supply at least the year, month, and day. You can omit the hours, minutes, and seconds.

The way JavaScript handles dates is very similar to the way Java handles dates: both languages have many of the same date methods, and both store dates internally as the number of milliseconds since January 1, 1970 00:00:00. Dates prior to 1970 are not allowed.

Examples

The following examples show several ways to assign dates:

```
today = new Date()
birthday = new Date("December 17, 1995 03:24:00")
birthday = new Date(95,11,17)
birthday = new Date(95,11,17,3,24,0)
```

Date Properties

The following table displays the date property. A detailed description of the property follows the table.

Property	Description
constructor	A reference to the function which created the object.
prototype	Enables you to add properties and methods to the object.

Date Methods

The following table displays a summary of the date methods. Detailed descriptions of each method follow the table.

Method	Description
getDate	Returns the day of the month for the specified date.
getDay	Returns the day of the week for the specified date.
getHours	Returns the hour in the specified date.
getMinutes	Returns the minutes in the specified date.
getMonth	Returns the month in the specified date.
getSeconds	Returns the seconds in the specified date.
getTime	Returns the numeric value corresponding to the time for the specified date.
getTimezoneOffset	Returns the time-zone offset in minutes for the current locale.
getFullYear	Returns the year in the specified date.
parse	Returns the number of milliseconds in a date string since January 1, 1970, 00:0 0:00, local time.
setDate	Sets the day of the month for a specified date.
setHours	Set the hours for a specified date.
setMinutes	Sets the minutes for a specified date.
setMonth	Sets the month for a specified date.
setSeconds	Sets the seconds for a specified date.
toGMTString	Converts a date to a string, using the Internet GMT conventions.

Method	Description
toLocaleString	Converts a data to a string, using the current locale's conventions.
UTC	Returns the number of milliseconds in a Date object since January 1, 1970.

getDate

Returns the day of the month for the specified date.

Applies to:

`Date`

Syntax

```
getDate()
```

Parameters

None

Description

The value returned by `getDate` is an integer between 1 and 31.

Examples

The second statement below assigns the value 25 to the variable `day`, based on the value of the `Date` object `Xmas95`.

```
Xmas95 = new Date("December 25, 1995 23:15:00")
day = Xmas95.getDate()
```

See also

`Date: setDate`

getDay

Returns the day of the week for the specified date.

Applies to

`Date`

Syntax

```
getDay()
```

Parameters

None

Description

The value returned by `getDay` is an integer corresponding to the day of the week: 0 for Sunday, 1 for Monday, 2 for Tuesday, and so on.

Examples

The second statement below assigns the value 1 to `weekday`, based on the value of the `Date` object `Xmas95. December 25, 1995, is a Monday`.

```
Xmas95 = new Date("December 25, 1995 23:15:00")
weekday = Xmas95.getDay()
```

getHours

Returns the hour for the specified date.

Applies to

`Date`

Syntax

`getHours()`

Parameters

None

Description

The value returned by getHours is an integer between 0 and 23.

Examples

The second statement below assigns the value 23 to the variable hours, based on the value of the `Date` object `Xmas95`.

```
Xmas95 = new Date("December 25, 1995 23:15:00")
hours = Xmas95.getHours()
```

See also

`Date: setHours`

getMinutes

Returns the minutes in the specified date.

Applies to

`Date`

Syntax

```
getMinutes()
```

Parameters

None

Description

The value returned by `getMinutes` is an integer between 0 and 59.

Examples

The second statement below assigns the value 15 to the variable `minutes`, based on the value of the `Date` object `Xmas95`.

```
Xmas95 = new Date("December 25, 1995 23:15:00")
minutes = Xmas95.getMinutes()
```

See also

`Date: setMinutes`

getMonth

Returns the month in the specified date.

Applies to

`Date`

Syntax

```
getMonth()
```

Parameters

None

Description

The value returned by `getMonth` is an integer between 0 and 11. 0 corresponds to January 1 to February, and so on.

Examples

The second statement below assigns the value 11 to the variable `month`, based on the value of the `Date` object `Xmas95`.

```
Xmas95 = new Date("December 25, 1995 23:15:00")
```

```
month = Xmas95.getMonth()
```

See also

Date: setMonth

getSeconds

Returns the seconds in the current time.

Applies to

Date

Syntax

```
getSeconds()
```

Parameters

None

Description

The value returned by getSeconds is an integer between 0 and 59.

Examples

The second statement below assigns the value 30 to the variable secs, based on the value of the Date object Xmas95.

```
Xmas95 = new Date("December 25, 1995 23:15:30")
secs = Xmas95.getSeconds()
```

See also

Date: setSeconds

getTime

Returns the numeric value corresponding to the time for the specified date.

Applies to

Date

Syntax

```
getTime()
```

Parameters

None

Description

The value returned by the `getTime` method is the number of milliseconds since 1 January 1970 00:00:00. You can use this method to help assign a date and time to another `Date` object.

Examples

The following example assigns the date value of `theBigDay` to `sameAsBigDay`:

```
theBigDay = new Date("July 1, 1999")
sameAsBigDay = new Date()
sameAsBigDay.setTime(theBigDay.getTime())
```

See also

`Date: setTime`

getTimezoneOffset

Returns the time-zone offset in minutes for the current locale.

Applies to

`Date`

Syntax

```
getTimezoneOffset()
```

Parameters

None

Description

The time-zone offset is the difference between local time and Greenwich Mean Time (GMT). Daylight savings time prevents this value from being a constant.

Examples

```
x = new Date()
currentTimeZoneOffsetInHours = x.getTimezoneOffset()/60
```

getFullYear

Returns the year in the specified date.

Applies to

`Date`

Syntax

```
getFullYear()
```

Parameters

None

Description

The value returned by `getFullYear` is the four-digit year. For example, if the year is 1856, the value returned is 1856. If the year is 2026, the value returned is 2026.

Examples

The second statement assigns the value 1995 to the variable `year`.

```
Xmas = new Date("December 25, 1995 23:15:00")
year = Xmas.getFullYear()
```

The second statement assigns the value 2000 to the variable `year`.

```
Xmas = new Date("December 25, 2000 23:15:00")
year = Xmas.getFullYear()
```

The second statement assigns the value 95 to the variable `year`, representing the year 1995.

```
Xmas.setYear(95)
year = Xmas.getFullYear()
```

See also

`Date: setYear`

parse

Returns the number of milliseconds in a date string since January 1, 1970, 00:00:00, local time. The `parse` method is static, read only.

Applies to:

`Date`

Syntax

`Date.parse(dateString)`

Parameters

`dateString`

A string representing a date.

Description

The `parse` method takes a date string (such as `"Dec 25, 1995"`) and returns the number of milliseconds since January 1, 1970, 00:00:00 (local time). This function is useful for setting date values based on string values, for example in conjunction with the `setTime` method and the `Date` object.

Given a string representing a time, `parse` returns the time value. It accepts the IETF standard date syntax: `"Mon, 25 Dec 1995 13:30:00 GMT."` It understands the continental US time-zone abbreviations, but for general use, use a time-zone offset, for example, `"Mon, 25 Dec 1995 13:30:00 GMT+0430"` (4 hours, 30 minutes west of the Greenwich meridian). If you do not specify a time zone, the local time zone is assumed. GMT and UTC are considered equivalent.

Because `parse` is a static method of `Date`, you always use it as `Date.parse()`, rather than as a method of a `Date object` you created.

Examples

If `IPOdate` is an existing `Date` object, then you can set it to August 9, 1995 as follows:

```
IPOdate.setTime(Date.parse("Aug 9, 1995"))
```

setDate

Sets the day of the month for a specified date.

Applies to:

`Date`

Syntax

```
setDate(dayValue)
```

Parameters

`datValue`

An integer from 1 to 31, representing the day of the month.

Examples

The second statement below changes the day for `theBigDay` to July 24 from its original value.

```
theBigDay = new Date("July 27, 1962 23:30:00"
theBigDay.setDate(24)
```

See also

`Date: getDate`

setHours

Sets the hours for a specified date.

Applies to:

`Date`

Syntax

```
setHours(hoursValue)
```

Parameters

```
hoursValue
```

An integer between 0 and 23, representing the hour.

Examples

```
theBigDay.setHours(7)
```

setMinutes

Sets the minutes for a specified date.

Applies to:

```
Date
```

Syntax

```
setMinutes(minutesValue)
```

Parameters

```
mintuesValue
```

An integer between 0 and 59, representing the minutes.

Examples

```
theBigDay.setMinutes(45)
```

See also

`Date:` `getMinutes`

setMonth

Sets the month for a specified date.

Applies to:

```
Date
```

Syntax

```
setMonth(monthValue)
```

Parameters

```
monthValue
```

An integer between 0 and 11, representing the months January through December.

Examples

```
theBigDay.setMonth(6)
```

See also

`Date:` getMonth

setSeconds

Sets the seconds for a specified date.

Applies to:

`Date`

Syntax

```
setSeconds(secondsValue)
```

Parameters

`secondsValue`

An integer between 0 and 59.

Examples

```
theBigDay.setSeconds(30)
```

See also

`Date:` getSeconds

setTime

Sets the value of a `Date` object.

Applies to:

`Date`

Syntax

```
setTime(timevalue)
```

Parameters

`timevalue`

An integer representing the number of milliseconds since 1 January 1970 00:00:00.

Description

Use the `setTime` method to help assign a date and time to another Date object.

Examples

```
theBigDay = new Date("July 1, 1999")
sameAsBigDay = new Date()
sameAsBigDay.setTime(theBigDay.getTime())
```

setYear

Sets the year for a specified date.

Applies to

`Date`

Syntax

```
setYear(yearValue)
```

Parameters

`yearValue`

An integer.

Description

If `yearValue` is a number between 0 and 99 (inclusive), then the year for `dateObjectName` is set to 1900 + `yearValue`. Otherwise, the year for `dateObjectName` is set to `yearValue`.

Examples

Note that there are two ways to set years in the 20th century.

- The year is set to 1996.

  ```
  theBigDay.setYear(96)
  ```

- The year is set to 1996.

  ```
  theBigDay.setYear(1996)
  ```

- The year is set to 2000.

  ```
  theBigDay.setYear(2000)
  ```

See also

`Date: getFullYear`

toGMTString

Converts a date to a string, using the Internet GMT conventions.

Applies to:

`Date`

Syntax

```
toGMTString()
```

Parameters

None

Description

The exact format of the value returned by `toGMTString` varies according to the platform.

Examples

In the following example, `today` is a `Date` object:

```
today.toGMTString()
```

In this example, the `toGMTString` method converts the date to GMT (UTC) using the operating system's time-zone offset and returns a string value that is similar to the following form. The exact format depends on the platform.

```
Mon, 18 Dec 1995 17:28:35 GMT
```

See also

`Date: toLocaleString`

toLocaleString

Converts a date to a string, using the current locale's conventions.

Applies to:

`Date`

Syntax

```
toLocaleString()
```

Parameters

None

Description

If you pass a date using `toLocaleString`, be aware that different platforms assemble the string in different ways. Using methods such as `getHours`, `getMinutes`, and `getSeconds` gives more portable results.

Examples

In the following example, `today` is a `Date` object:

```
today = new Date(95,11,18,17,28,35) //months are represented by 0 to 11
today.toLocaleString()
```

In this example, `toLocaleString` returns a string value that is similar to the following form. The exact format depends on the platform.

```
12/18/95 17:28:35
```

See also

`Date:` `toGMTString`

UTC

Returns the number of milliseconds in a `Date` object since January 1, 1970, 00:00:00, Universal Coordinated Time (GMT). `UTC` is static, read only.

Applies to

`Date`

Syntax

```
Date.UTC(year, month, day, hrs, min, sec)
```

Parameters

`year`

A year after 1900.

`month`

A month between 0 and 11.

`date`

A day of the month between 1 and 31.

`hrs`

(Optional) A number of hours between 0 and 23.

`min`

(Optional) A number of minutes between 0 and 59.

`sec`

(Optional) A number of seconds between 0 and 59.

Description

`UTC` takes comma-delimited date parameters and returns the number of milliseconds since January 1, 1970, 00:00:00, Universal Coordinated Time (GMT).

Because `UTC` is a static method of `Date`, you always use it as `Date.UTC()`, rather than as a method of a `Date` object you created.

Examples

The following statement creates a `Date` object using GMT instead of local time:

```
gmtDate = new Date(Date.UTC(96, 11, 1, 0, 0, 0))
```

prototype

Property	Description
Prototype	Allows the addition of properties to a `Date` object.

Property of

`Date`

Description

Represents the prototype for this class. You can use the prototype to add properties or methods to all instances of a class.

Function

Specifies a string of JavaScript code to be compiled as a function. Function is a core object.

Created by

The `Function` constructor:

```
new Function (arg1, arg2, ... argN, functionBody)
```

Parameters

`arg1, arg2,...argn`

(Optional) Names to be used by the function as formal argument names. Each must be a string that corresponds to a valid JavaScript identifier; for example "x" or "theForm".

`functionBody`

A string containing the JavaScript statements comprising the function definition.

Description

`Function` objects are evaluated each time they are used. This is less efficient than declaring a function and calling it within your code, because declared functions are compiled.

In addition to defining functions as described here, you can also use the `function` statement, as described in the JavaScript Guide.

Examples

The following code assigns a function to the variable `activeSection.name`. This function sets the current document's section name.

```
var changeName = new Function("activeSection.name='sales'")
```

To call the `Function object`, you can specify the variable name as if it were a function. The following code executes the function specified by the `changeName` variable:

```
var newName="sales"
if (newName=="sales") {newName()}
function changeName() {
  activeSection.name='sales'
}
```

Assigning a function to a variable is similar to declaring a function, but they have differences:

When you assign a function to a variable using `var changeName = new Function("...")`, `changeName` is a variable for which the current value is a reference to the function created with `new Function()`.

When you create a function using `function changeName() {...}`, `changeName` is not a variable, it is the name of a function

Specifying arguments in a Function object

The following code specifies a `Function` object that takes two arguments.

```
var multFun = new Function("x", "y", "return x * y")
```

The string arguments `"x"` and `"y"` are formal argument names that are used in the function body, `"return x * y"`.

The following code shows a way to call the function `multFun`:

```
var theAnswer = multFun(7,6)
Console.Write("15*2 = " + multFun(15,2))
```

Function Properties

The following table displays a summary of the function properties. Detailed descriptions of each property follow the table.

Property	Description
arguments	An array corresponding to the arguments passed to a function.
arity	Indicates the number of arguments expected by the function.
caller	Specifies which function called the current function.

Property	Description
prototype	Allows the addition of properties to a `Function` object.

arguments

An array corresponding to the arguments passed to a function.

Property of

`Function`

Description

You can call a function with more arguments than it is formally declared to accept by using the `arguments` array. This technique is useful if a function can be passed a variable number of arguments. You can use `arguments.length` to determine the number of arguments passed to the function, and then treat each argument by using the `arguments` array.

The `arguments` array is available only within a function declaration. Attempting to access the `arguments` array outside a function declaration results in an error.

The `this` keyword does not refer to the currently executing function, so you must refer to functions and `Function` objects by name, even within the function body.

In JavaScript 1.2, `arguments` includes these additional properties:

- formal arguments—Each formal argument of a function is a property of the `arguments` array.
- local variables—Each local variable of a function is a property of the `arguments` array.
- `caller`—A property whose value is the `arguments` array of the outer function. If there is no outer function, the value is undefined.
- `callee` —A property whose value is the function reference.

For example, the following script demonstrates several of the `arguments` properties:

```
function b(z) {
      Console.Write(arguments.z)
      Console.Write (arguments.caller.x)
      return 99
}
function a(x, y) {
      return  b(534)
}
Console.Write (a(2,3))
This displays:
534
2
99
```

534 is the actual parameter to b, so it is the value of `arguments.z`. *2* is a's actual x parameter, so (viewed within b) it is the value of `arguments.caller.x`. *99* is what `a(2,3)` returns.

Examples

This example defines a function that creates test lists. The only formal argument for the function is a string that changes the appearance of the list. To create a bullet list (also called an "unordered list"), use `"U"`. To create a numbered list (also called an "ordered list"), use `"O"`. The function is defined as follows:

```
function list(type) {
      Console.Write(type)
      for (var i=1; i<list.arguments.length; i++) {
            Console.Write(list.arguments[i])
            Console.Write(type)
      }
}
```

You can pass any number of arguments to this function, and it displays each argument as an item in the type of list indicated. For example, the following call to the function:

```
list("U", "One", "Two", "Three")
results in this output:
One
Two
Three
```

arity

Indicates the number of arguments expected by the function.

Description

`arity` is external to the function, and indicates how many arguments the function expects. By contrast, `arguments.length` provides the number of arguments actually passed to the function.

Example

The following example demonstrates the use of `arity` and `arguments.length`.

```
function addNumbers(x,y){
      Console.Write("length = " + arguments.length)
      z = x + y
}
Console.Write("arity = " + addNumbers.arity)
addNumbers(3,4,5)
```

This script writes:

```
arity = 2
length = 3
```

caller

Returns the name of the function that invoked the currently executing function.

Property of

`Function`

Description

In JavaScript 1.4, the `caller` property is available only within the body of a function. If used outside a function declaration, the `caller` property is null.

If the currently executing function was invoked by the top level of a JavaScript program, the value of `caller` is null.

The `this` keyword does not refer to the currently executing function, so you must refer to functions and `Function` objects by name, even within the function body.

The `caller` property is a reference to the calling function, so if you use it in a string context, you get the result of calling `functionName.toString`. That is, the decompiled canonical source form of the function.

You can also call the calling function, if you know what arguments it might want. Thus, a called function can call its caller without knowing the name of the particular caller, provided it knows that all of its callers have the same form and fit, and that they will not call the called function again unconditionally (which would result in infinite recursion).

In JavaScript 1.4 arguments.caller was an intrinsic object available to any function. Using Arguments you could examine such properties as the parameters you were called with and who called you. For example, you might have added:

```
function myFunc() {
   if (arguments.caller == null) {
      return ("The function was called from the top!");
   } else
      return ("This function's caller was " +
arguments.caller.callee.toString().split("(")[0].replace("function ",""))
}
```

In JavaScript 1.5 this has been changed implementing of a direct function property that is more directly accessible. Now you would need to replace the above script by adding:

```
function myFunc() {
   if (myFunc.caller == null) {
      return ("The function was called from the top!");
   } else
      return ("This function's caller was " + myFunc.caller.toString().split("(")
[0].replace("function ",""))
}
```

Calling the script has not changed in JavaScript 1.5:

```
Console.Writeln(myFunc())
function localFunction(){
   Console.Writeln(myFunc())
}
localFunction()
```

Additionally, prior to the introduction of JavaScript 1.5, access to your caller and its callers was through the arguments.caller and arguments.caller.callee objects. For example, assume the following scenario:

```
function a (){ b() }
function b (){ c() }
function c (){ d() }
function d (){getStack() }
a()
```

to print a list of the names of all the functions from the original caller to the current point of execution was achieved by writing the getStack function as follows

```
 function getStack(){
      var f = arguments.caller
      var stack = "Stack trace:";
      while (f != null){
        stack += "\r\n" + f.callee.toString().split("{")[0];
        f = f.caller
      }
      Console.Writeln(stack)
   }
```

With JavaScript 1.5 arguments are no longer available and a "name" property has been added. The equivalent function is now:

```
function getStack(){
      var f = getStack.caller
      var stack = "Stack trace:";
      while (f != null){
        stack += "\r\n function " + f.name + "()";
        f = f.caller;
      }
      Console.Writeln(stack)
   }
```

The Mozilla web site also refers to the fact that where there is recursion in functions, then an infinite loop occurs with this code. See http://developer.mozilla.org/en/docs/Core_JavaScript_1.5_Reference:Global_Objects:Function:caller. If the original environment showed:

```
var go = true
function a (){ b() }
function b (){
   if (go){ go = false; a() }
   else{ c() }
}
function b (){ c() }
function c (){ d() }
function d (){getStack() }
a()
```

then an infinite loop resulted in the function trying to print the stack trace. To avoid this, add script that stops after a certain amount of loops

Examples

The following code checks the value of a function's `caller` property.

```
function myFunc() {
      if (myFunc.caller == null) {
            alert("The function was called from the top!")
      } else alert("This function's caller was " + myFunc.caller)
}
```

See also

`Function: arguments`

prototype

A value from which instances of a particular class are created. Every object that can be created by calling a constructor function has an associated `prototype` property.

Property of

`Object`

Description

You can add new properties or methods to an existing class by adding them to the prototype associated with the constructor function for that class. The syntax for adding a new property or method is:

`fun.prototype.name = value`

where

Property	Description
fun	The name of the constructor function object you want to change.
name	The name of the property or method to be created.
value	The value initially assigned to the new property or method.

If you add a new property to the prototype for an object, then all objects created with that object's constructor function will have that new property, even if the objects existed before you created the new property. For example, assume you have the following statements:

```
var array1 = new Array();
var array2 = new Array(3);
Array.prototype.description=null;
array1.description="Contains some stuff"
array2.description="Contains other stuff"
```

After you set a property for the prototype, all subsequent objects created with Array will have the property:

```
anotherArray=new Array()
anotherArray.description="Currently empty"
```

Example

The following example creates a method, str_rep, and uses the statement String.prototype.rep = str_rep to add the method to all String objects. All objects created with new String() then have that method, even objects already created. The example then creates an alternate method and adds that to one of the String objects using the statement s1.rep = fake_rep. The str_rep method of the remaining String objects is not altered.

```
var s1 = new String("a")
var s2 = new String("b")
var s3 = new String("c")
// Create a repeat-string-N-times method for all String objects
function str_rep(n) {
var s = "", t = this.toString()
while (--n >= 0) s += t
return s
}
String.prototype.rep = str_rep
// Display the results
Console.Write("s1.rep(3) is " + s1.rep(3)) // "aaa"
Console.Write("s2.rep(5) is " + s2.rep(5)) // "bbbbb"
Console.Write("s3.rep(2) is " + s3.rep(2)) // "cc"
// Create an alternate method and assign it to only one String variable
function fake_rep(n) {
   return "repeat " + this + n + " times."
}
s1.rep = fake_rep
Console.Write("s1.rep(1) is " + s1.rep(1)) // "repeat a 1 times."
Console.Write("s2.rep(4) is " + s2.rep(4)) // "bbbb"
Console.Write("s3.rep(6) is " + s3.rep(6)) // "cccccc"
```

This example produces the following output:

```
s1.rep(3) is aaa
s2.rep(5) is bbbbb
s3.rep(2) is cc
s1.rep(1) is repeat a1 times.
s2.rep(4) is bbbb
s3.rep(6) is cccccc
```

The function in this example also works on String objects not created with the String constructor. The following code returns "zzz".

```
"z".rep(3)
```

Function Methods

The following displays the function method. A detailed description of the method follows the table.

Method	Description
toString	Returns a string representing the specified object.

toString

Returns a string representing the specified object.

Applies to

`Function`

Syntax

`toString()`

Parameters

None

Description

Every object has a `toString` method that is automatically called when it is to be represented as a text value or when an object is referred to in a string concatenation.

You can use `toString` within your own code to convert an object into a string, and you can create your own function to be called in place of the default `toString` method.

For `Function` objects, the built-in `toString` method decompiles the function back into the JavaScript source that defines the function This string includes the `function` keyword, the argument list, curly braces, and function body.

For example, assume you have the following code that defines the `Dog` object type and creates `theDog,` an object of `type Dog`:

```
function Dog(name,breed,color,sex) {
      this.name=name
      this.breed=breed
      this.color=color
      this.sex=sex
}
theDog = new Dog("Gabby","Lab","chocolate","girl")
```

Any time `Dog` is used in a string context, JavaScript automatically calls the `toString` function, which returns the following string:

```
function Dog(name, breed, color, sex) { this.name = name; this.breed = breed; this.color
= color; this.sex = sex; }
```

For information on defining your own `toString` method, see the `Object: toString` method.

Math

A built-in object that has properties and methods for mathematical constants and functions. For example, the Math object's PI property has the value of pi. Math is a core object.

Created by

The Math object is a top-level, predefined JavaScript object. You can automatically access it without using a constructor or calling a method.

Description

All properties and methods of Math are static. You refer to the constant PI as Math.PI and you call the sine function as Math.sin(x), where x is the method's argument. Constants are defined with the full precision of real numbers in JavaScript.

It is often convenient to use the with statement when a section of code uses several Math constants and methods, so you don't have to type "Math" repeatedly. For example:

```
with (Math) {
      a = PI * r*r
      y = r*sin(theta)
      x = r*cos(theta)
}
```

Math Properties

The following table displays a summary of the math properties. Detailed descriptions of each property follow the table.

Properties	Descriptions
E	Euler's constant and the base of natural logarithms, approximately 2.718.
LN10	Natural logarithm of 10, approximately 2.302.
LN2	Natural logarithm of 2, approximately 0.693.
LOG10E	Base 10 logarithm of E (approximately 0.434).
LOG2E	Base 2 logarithm of E (approximately 1.442).
PI	Ratio of the circumference of a circle to its diameter, approximately 3.14159.

Properties	Descriptions
SQRT1_2	Square root of v; equivalently, 1 over the square root of 2, approximately 0.707.
SQRT2	Square root of 2, approximately 1.414.

E

Euler's constant and the base of natural logarithms, approximately 2.718. `Math` is static, read-only.

Property of

`Math`

Examples

The following function returns Euler's constant:

```
function getEuler() {
      return Math.E
}
```

Description

Because `E` is a static property of `Math`, you always use it as `Math.E`, rather than as a property of a `Math` object you created.

LN10

The natural logarithm of 10, approximately 2.302. `LN10` is static, read-only.

Property of

`Math`

Example

The following function returns the natural log of 10:

```
function getNatLog10() {
      return Math.LN10
}
```

Description

Because `LN10` is a static property of `Math`, you always use it as `Math.LN10`, rather than as a property of a `Math` object you created.

LN2

The natural logarithm of 2, approximately 0.693. `LN2` is static, read-only.

Property of

`Math`

Examples

The following function returns the natural log of 2:

```
function getNatLog2() {
      return Math.LN2
}
```

Description

Because `LN2` is a static property of `Math`, you always use it as `Math.LN2`, rather than as a property of a `Math` object you created.

LOG10E

The base 10 logarithm of E (approximately 0.434). `LOG10E` is static, read-only.

Property of

`Math`

Example

The following function returns the base 10 logarithm of E:

```
function getLog10e() {
      return Math.LOG10E
}
```

Description

Because `LOG10E` is a static property of `Math`, you always use it as `Math.LOG10E`, rather than as a property of a `Math` object you created.

LOG2E

The base 2 logarithm of E (approximately 1.442). `LOG2E` is static, read-only.

Property of

`Math`

Examples

The following function returns the base 2 logarithm of E:

```
function getLog2e() {
      return Math.LOG2E
}
```

Description

Because `LOG2E` is a static property of `Math`, you always use it as `Math.LOG2E`, rather than as a property of a `Math` object you created.

PI

The ratio of the circumference of a circle to its diameter, approximately 3.14159. `PI` is static, read-only.

Property of

`Math`

Examples

The following function returns the value of pi:

```
function getPi() {
      return Math.PI
}
```

Description

Because `PI` is a static property of `Math`, you always use it as `Math.PI`, rather than as a property of a `Math` object you created.

SQRT1_2

The square root of Ú; equivalently, 1 over the square root of 2, approximately 0.707. `SQRT1_2` is static, read-only.

Property of

`Math`

Example

The following function returns 1 over the square root of 2:

```
function getRoot1_2() {
      return Math.SQRT1_2
}
```

Description

Because `SQRT1_2` is a static property of `Math`, you always use it as `Math.SQRT1_2`, rather than as a property of a `Math` object you created.

SQRT2

The square root of 2, approximately 1.414. `SQRT2` is static, read-only.

Property of

`Math`

Example

The following function returns the square root of 2:

```
function getRoot2() {
      return Math.SQRT2
}
```

Description

Because `SQRT2` is a static property of `Math`, you always use it as `Math.SQRT2`, rather than as a property of a `Math` object you created.

Math Methods

The following displays a summary of the math methods. Detailed descriptions of each method follow the table.

Method	Description
abs	Returns the absolute value of a number.
acos	Returns the arccosine (in radians) of a number.
asin	Returns the arcsine (in radians) of a number.
atan	Returns the arctangent (in radians) of a number.
atan2	Returns the arctangent of the quotient of its arguments.
ceil	Returns the smallest integer greater than or equal to a number.
cos	Returns the cosine of a number.
exp	Returns Enumber, where number is the argument, and E is Euler's constant, the base of the natural logarithms.

Method	Description
floor	Returns the largest integer less than or equal to a number.
log	Returns the natural logarithm (base E) of a number.
max	Returns the greater of two numbers.
min	Returns the lesser of two numbers.
pow	Returns base to the exponent power, that is, baseexponent.
random	Returns a pseudo-random number between 0 and 1.
round	Returns the value of a number rounded to the nearest integer.
sin	Returns the sine of a number.
sqrt	Returns the square root of a number.
tan	Returns the tangent of a number.

abs

Returns the absolute value of a number.

Applies to

`Math`

Syntax

```
abs(x)
```

Parameters

`x`

A number.

Example

The following function returns the absolute value of the variable x:

```
function getAbs(x) {
      return Math.abs(x)
}
```

Description

abs is a static method of Math. As a result, you always use it as Math.abs(), rather than as a method of a Math object you create.

acos

Returns the arccosine (in radians) of a number.

Applies to

Math

Syntax

acos(x)

Parameters

x

A number.

Description

The acos method returns a numeric value between 0 and pi radians. If the value of number is outside this range, it returns 0.

acos is a static method of Math. As a result, you always use it as Math.acos(), rather than as a method of a Math object you create.

Example

The following function returns the arccosine of the variable x:

```
function getAcos(x) {
      return Math.acos(x)
}
```

If you pass -1 to getAcos, it returns 3.141592653589793; if you pass 2, it returns 0 because 2 is out of range.

See also

Math:asin, Math:atan, Math:atan2, Math:cos, Math:sin, Math:tan

asin

Returns the arcsine (in radians) of a number.

Applies to

Math

Syntax

asin(x)

Parameters

x

A number.

Description

The asin method returns a numeric value between -pi/2 and pi/2 radians. If the value of number is outside this range, it returns 0.

asin is a static method of Math. As a result, you always use it as Math.asin(), rather than as a method of a Math object you create.

Examples

The following function returns the arcsine of the variable x:

```
function getAsin(x) {
      return Math.asin(x)
}
```

If you pass getAsin the value 1, it returns 1.570796326794897 (pi/2); if you pass it the value 2, it returns 0 because 2 is out of range.

See also

Math:acos, Math:atan, Math:atan2, Math:cos, Math:sin, Math:tan

atan

Returns the arctangent (in radians) of a number.

Applies to

Math

Syntax

atan(x)

Parameters

x

A number.

Description

The `atan` method returns a numeric value between -pi/2 and pi/2 radians.

`atan` is a static method of `Math`. As a result, you always use it as `Math.atan()`, rather than as a method of a `Math` object you create.

Example

The following function returns the arctangent of the variable `x`:

```
function getAtan(x) {
      return Math.atan(x)
}
```

If you pass `getAtan` the value 1, it returns 0.7853981633974483; if you pass it the value .5, it returns 0.4636476090008061.

See also

`Math.acos`, `Math.asin`, `Math.atan2`, `Math.cos`, `Math.sin`, `Math.tan`

atan2

Returns the arctangent of the quotient of its arguments.

Applies to

`Math`

Syntax

`atan2(y, x)`

Parameters

`y,x`

A number.

Description

The `atan2` method returns a numeric value between -pi and pi representing the angle theta of an (`x,y`) point. This is the counterclockwise angle, measured in radians, between the positive X axis, and the point (`x,y`). Note that the arguments to this function pass the y-coordinate first and the x-coordinate second.

`atan2` is passed separate `x` and `y` arguments, and `atan` is passed the ratio of those two arguments.

`atan2` is a static method of `Math`. As a result, you always use it as `Math.atan2()`, rather than as a method of a `Math` object you create.

Example

The following function returns the angle of the polar coordinate:

```
function getAtan2(x,y) {
      return Math.atan2(x,y)
}
```

If you pass `getAtan2` the values (90,15), it returns 1.4056476493802699; if you pass it the values (15,90), it returns 0.16514867741462683.

See also

`Math.acos`, `Math.asin`, `Math.atan`, `Math.cos`, `Math.sin`, `Math.tan`

ceil

Returns the smallest integer greater than or equal to a number.

Applies to

`Math`

Syntax

`ceil(x)`

Parameters

`x`

A number.

Description

`ceil` is a static method of `Math`. As a result, you always use it as `Math.ceil()`, rather than as a method of a `Math` object you create.

Example

The following function returns the ceil value of the variable `x`:

```
function getCeil(x) {
      return Math.ceil(x)
}
```

If you pass 45.95 to `getCeil`, it returns 46; if you pass -45.95, it returns -45.

See also

`Math:floor`

cos

Returns the cosine of a number.

Applies to

Math

Syntax

cos(x)

Parameters

x

A number.

Description

The cos method returns a numeric value between -1 and 1, which represents the cosine of the angle.

cos is a static method of Math. As a result, you always use it as Math.cos(), rather than as a method of a Math object you create.

Examples

The following function returns the cosine of the variable x:

```
function getCos(x) {
      return Math.cos(x)
}
```

If x equals Math.PI/2, getCos returns 6.123031769111886e-017; if x equals Math.PI, getCos returns -1.

See also

Math:acos, Math.asin, Math.atan, Math.atan2, Math.sin, Math.tan

exp

Returns Ex, where x is the argument, and E is Euler's constant, the base of the natural logarithms.

Applies to

Math

Syntax

exp(x)

Parameters

x

A number.

Description

exp is a static method of Math. As a result, you always use it as Math.exp(), rather than as a method of a Math object you create.

Examples

The following function returns the exponential value of the variable x:

```
function getExp(x) {
      return Math.exp(x)
}
```

If you pass getExp the value 1, it returns 2.718281828459045.

See also

Math:E, Math:log, Math:pow

floor

Returns the largest integer less than or equal to a number.

Applies to

Math

Syntax

floor(x)

Parameters

x

A number.

Description

floor is a static method of Math. As a result, you always use it as Math.floor(), rather than as a method of a Math object you create.

Examples

The following function returns the floor value of the variable x:

```
function getFloor(x) {
      return Math.floor(x)
}
```

If you pass 45.95 to getFloor, it returns 45; if you pass -45.95, it returns -46.

See also

Math:ceil

log

Returns the natural logarithm (base E) of a number.

Applies to

Math

Syntax

log(x)

Parameters

x

A number.

Description

If the value of number is outside the suggested range, the return value is always -1.797693134862316e+308.

log is a static method of Math. As a result, you always use it as Math.log(), rather than as a method of a Math object you create.

Examples

The following function returns the natural log of the variable x:

```
function getLog(x) {
      return Math.log(x)
}
```

If you pass getLog the value 10, it returns 2.302585092994046; if you pass it the value 0, it returns -1.797693134862316e+308 because 0 is out of range.

See also

Math.exp, Math.pow

max

Returns the larger of two numbers.

Applies to

Math

Syntax

max(x,y)

Parameters

x,y

Numbers.

Description

`max` is a static method of `Math`. As a result, you always use it as `Math.max()`, rather than as a method of a `Math` object you create.

Examples

The following function evaluates the variables `x` and `y`:

```
function getMax(x,y) {
      return Math.max(x,y)
}
```

If you pass `getMax` the values 10 and 20, it returns 20; if you pass it the values -10 and -20, it returns -10.

See also

`Math.min`

min

Returns the smaller of two numbers.

Applies to

`Math`

Syntax

`min(x,y)`

Parameters

`x,y`

Numbers.

Description

`min` is a static method of `Math`. As a result, you always use it as `Math.min()`, rather than as a method of a `Math` object you create.

Examples

The following function evaluates the variables *x* and *y*:

```
function getMin(x,y) {
      return Math.min(x,y)
}
```

If you pass `getMin` the values 10 and 20, it returns 10; if you pass it the values -10 and -20, it returns -20.

See also

Math.max

pow

Returns base to the exponent power, that is, baseexponent.

Applies to

Math

Syntax

```
pow(x,y)
```

Parameters

base

The base number.

exponent

The exponent to which to raise base.

Description

pow is a static method of Math. As a result, you always use it as Math.pow(), rather than as a method of a Math object you create.

Examples

```
function raisePower(x,y) {
      return Math.pow(x,y)
}
```

If x is 7 and y is 2, raisePower returns 49 (7 to the power of 2).

See also

Math.exp, Math.log

random

Returns a pseudo-random number between 0 and 1. The random number generator is seeded from the current time, as in Java.

Applies to

Math

Syntax

```
random()
```

Parameters

None

Description

`random` is a static method of `Math`. As a result, you always use it as `Math.random()`, rather than as a method of a `Math` object you create.

Examples

```
//Returns a random number between 0 and 1
function getRandom() {
      return Math.random()
}
```

round

Returns the value of a number rounded to the nearest integer.

Applies to

`Math`

Syntax

`round(x)`

Parameters

`x`

A number.

Description

If the fractional portion of number is .5 or greater, the argument is rounded to the next highest integer. If the fractional portion of number is less than .5, the argument is rounded to the next lowest integer.

`round` is a static method of `Math`. As a result, you always use it as `Math.round()`, rather than as a method of a `Math` object you create.

Examples

```
//Displays the value 20
Console.Write("The rounded value is " + Math.round(20.49))
//Displays the value 21
Console.Write("The rounded value is " + Math.round(20.5))
//Displays the value -20
Console.Write("The rounded value is " + Math.round(-20.5))
//Displays the value -21
Console.Write("The rounded value is " + Math.round(-20.51))
```

sin

Returns the sine of a number.

Applies to

Math

Syntax

```
sin(x)
```

Parameters

x

A number.

Description

The sin method returns a numeric value between -1 and 1, which represents the sine of the argument.

sin is a static method of Math. As a result, you always use it as Math.sin(), rather than as a method of a Math object you create.

Examples

The following function returns the sine of the variable *x*:

```
function getSine(x) {
      return Math.sin(x)
}
```

If you pass getSine the value Math.PI/2, it returns 1.

See also

Math:acos, Math:asin, Math:atan, Math:atan2, Math:cos, Math:tan

sqrt

Returns the square root of a number.

Applies to

Math

Syntax

```
sqrt(x)
```

Parameters

x

A number.

Description

If the value of number (x) is outside the required range,sqrt returns 0.

sqrt is a static method of Math. As a result, you always use it as Math.sqrt(), rather than as a method of a Math object you create.

Examples

The following function returns the square root of the variable *x*:

```
function getRoot(x) {
      return Math.sqrt(x)
}
```

If you pass getRoot the value 9, it returns 3; if you pass it the value 2, it returns 1.414213562373095.

tan

Returns the tangent of a number.

Applies to

Math

Syntax

tan(x)

Parameters

x

A number.

Description

The tan method returns a numeric value that represents the tangent of the angle.

tan is a static method of Math. As a result, you always use it as Math.tan(), rather than as a method of a Math object you create.

Examples

The following function returns the tangent of the variable *x*:

```
function getTan(x) {
      return Math.tan(x)
}
```

If you pass Math.PI/4 to getTan, it returns 0.9999999999999999.

Number

Lets you work with numeric values. The `Number` object is an object wrapper for primitive numeric values and a core object.

Created by

The `Number` constructor.

Syntax

```
new Number(value);
```

Parameters

`value`

The numeric value of the object being created.

Description

The primary uses for the `Number` object are:

- To access its constant properties, which represent the largest and smallest representable numbers, positive and negative infinity, and the Not-a-Number value
- To create numeric objects that you can add properties to. Most likely, you will rarely need to create a `Number` object.

The properties of `Number` are properties of the class itself, not of individual `Number` objects.

`Number(x)` now produces `NaN` rather than an error if *x* is a string that does not contain a well-formed numeric literal. For example:

```
x=Number("three");
Console.Write(x);
prints NaN
```

Examples

The following example uses the `Number` object's properties to assign values to several numeric variables:

```
biggestNum = Number.MAX_VALUE
smallestNum = Number.MIN_VALUE
infiniteNum = Number.POSITIVE_INFINITY
negInfiniteNum = Number.NEGATIVE_INFINITY
notANum = Number.NaN
```

The following example creates a `Number` object, `myNum`, then adds a `description` property to all `Number` objects. Then a value is assigned to the `myNum` object's `description` property.

```
myNum = new Number(65)
Number.prototype.description=null
myNum.description="wind speed"
```

Number Properties

The following displays a summary of the number properties. Detailed descriptions of each property follow the table.

Property	Description
MAX_VALUE	The largest representable number.
MIN_VALUE	The smallest representable number.
NaN	Special "not a number" value.
NEGATIVE_INFINITY	Special infinite value; returned on overflow.
POSITIVE_INFINITY	Special negative infinite value; returned on overflow.
Prototype	Allows the addition of properties to a Number object.

MAX_VALUE

The maximum numeric value representable in JavaScript.

Property of

Number

Description

The MAX_VALUE property has a value of approximately 1.79E+308. Values larger than MAX_VALUE are represented as Infinity.

Description

The MAX_VALUE property has a value of approximately 1.79E+308. Values larger than MAX_VALUE are represented as Infinity.

MAX_VALUE is a static, read-only property of Number. As a result, you always use it as Number.MAX_VALUE, rather than as a property of a Number object you create.

Example

The following code multiplies two numeric values. If the result is less than or equal to MAX_VALUE, the func1 function is called; otherwise, the func2 function is called.

```
if (num1 * num2 <= Number.MAX_VALUE)
```

```
      func1()
else
      func2()The smallest positive numeric value that can be represented in JavaScript.
```

MIN_VALUE

The smallest positive numeric value representable in JavaScript.

Property of

Number

Description

The MIN_VALUE property is the number closest to 0, not the most negative number, that JavaScript can represent.

MIN_VALUE has a value of approximately 2.22E-308. Values smaller than MIN_VALUE ("underflow values") are converted to 0.

MIN_VALUE is a static, read-only property of Number. As a result, you always use it as Number.MIN_VALUE, rather than as a property of a Number object you create.

Example

The following code divides two numeric values. If the result is greater than or equal to MIN_VALUE, the func1 function is called; otherwise, the func2 function is called.

```
if (num1 / num2 >= Number.MIN_VALUE)
      func1()
else
      func2()
```

NaN

A special value representing Not-A-Number. This value is represented as the unquoted literal NaN. NaN is a read-only property.

Property of

Number

Description

JavaScript prints the value Number.NaN as NaN.

NaN is always unequal to any other number, including NaN itself; you cannot check for the not-a-number value by comparing to Number.NaN. Use the isNaN function instead.

You might use the NaN property to indicate an error condition for a function that should return a valid number.

Example

In the following example, if month has a value greater than 12, it is assigned NaN, and a message is displayed indicating valid values.

```
var month = 13
if (month < 1 || month > 12) {
      month = Number.NaN
      alert("Month must be between 1 and 12.")
}
```

NEGATIVE_INFINITY

A special numeric value representing negative infinity. This value is displayed as -Infinity.

Property of:

Number

Description

This value behaves mathematically like infinity; for example, anything multiplied by infinity is infinity, and anything divided by infinity is 0.

NEGATIVE_INFINITY is a static, read-only property of Number. As a result, you always use it as Number.NEGATIVE_INFINITY, rather than as a property of a Number object you create.

Examples

In the following example, the variable smallNumber is assigned a value that is smaller than the minimum value. When the if statement executes, smallNumber has the value -Infinity, so the func1 function is called.

```
var smallNumber = -Number.MAX_VALUE*10
if (smallNumber == Number.NEGATIVE_INFINITY)
      func1()
else
      func2()
```

POSITIVE_INFINITY

A special numeric value representing infinity. This value is displayed as Infinity.

Property of

Number

Description

This value behaves mathematically like infinity; for example, anything multiplied by infinity is infinity, and anything divided by infinity is 0.

JavaScript does not have a literal for Infinity.

POSITIVE_INFINITY is a static, read-only property of Number. As a result, you always use it as Number.POSITIVE_INFINITY, rather than as a property of a Number object you create.

Example

In the following example, the variable bigNumber is assigned a value that is larger than the maximum value. When the if statement executes, bigNumber has the value Infinity, so the func1 function is called.

```
var bigNumber = Number.MAX_VALUE * 10
if (bigNumber == Number.POSITIVE_INFINITY)
      func1()
else
      func2()
```

Prototype

Description

Represents the prototype for this class. You can use the prototype to add properties or methods to all instances of a class. For information on prototypes, see Function.prototype.

Property of

Number

Number Methods

The following table displays the method for Number. A detailed description of this method follows the table.

Method	Description
tostring	Returns a string representing the specified object.

toString

Returns a string representing the specified object.

Applies to

Number

Syntax

```
toString()
```

```
toString(radix)I I
```

Parameters

radix

(Optional) An integer between 2 and 16 specifying the base to use for representing numeric values.

Description

Every object has a toString method that is automatically called when it is to be represented as a text value or when an object is referred to in a string concatenation.

You can use toString within your own code to convert an object into a string, and you can create your own function to be called in place of the default toString method.

You can use toString on numeric values, but not on numeric heliterals:

```
// The next two lines are valid
var howMany=10
 ("howMany.toString() is " + howMany.toString() )
// The next line causes an error
 ("45.toString() is " + 45.toString() )
```

For information on defining your own toString method, see the Object.toString method.

Object

Object is the primitive JavaScript object type. All JavaScript objects are descended from Object. That is, all JavaScript objects have the methods defined for Object.

Created by

The Object constructor

Syntax

```
new Object();
```

Parameters

None

Object Properties

The following table displays a summary of the object properties. Detailed descriptions of each property follow the table.

Property	Description
constructor	Specifies the function that creates an object's prototype.
Prototype	Allows the addition of properties to all objects.

constructor

Specifies the function that creates an object's prototype. Note that the value of this property is a reference to the function itself, not a string containing the function's name.

Property of:

Object

Description

All objects inherit a constructor property from their prototype:

```
o = new Object  // or o = {}
o.constructor == Object
a = new Array   // or a = []
a.constructor == Array
n = new Number(3)
n.constructor == Number
```

Example

The following example creates a prototype, Tree, and an object of that type, theTree. The example then displays the constructor property for the object theTree.

```
function Tree(name) {
      this.name=name
}
theTree = new Tree("Redwood")
Console.Write("theTree.constructor is" +     theTree.constructor)
```

This example displays the following output:

```
theTree.constructor is function Tree(name) { this.name = name; }
```

Prototype

Represents the prototype for this class. You can use the prototype to add properties or methods to all instances of a class.

Property of

Object

Object Methods

The following table displays a summary of the object methods. Detailed descriptions of each method follow the table.

Method	Description
eval	Evaluates a string of JavaScript code in the context of the specified object.
toString	Returns a string representing the specified object.
unwatch	Removes a watchpoint from a property of the object.
valueOf	Returns the primitive value of the specified object.
watch	Adds a watchpoint to a property of the object.

eval

Evaluates a string of JavaScript code in the context of this object.

Property of

`Object`

Syntax

```
eval(string)
```

Parameters

`string`

Any string representing a JavaScript expression, statement, or sequence of statements. The expression can include variables and properties of existing objects.

Description

The argument of the `eval` method is a string. If the string represents an expression, `eval` evaluates the expression. If the argument represents one or more JavaScript statements, `eval` performs the statements. Do not call `eval` to evaluate an arithmetic expression; JavaScript evaluates arithmetic expressions automatically.

If you construct an arithmetic expression as a string, you can use `eval` to evaluate it at a later time. For example, suppose you have a variable *x*. You can postpone evaluation of an expression involving *x* by assigning the string value of the expression, say `"3 * x + 2"`, to a variable, and then calling `eval` at a later point in your script.

`eval` is also a global function, not associated with any object.

Examples

The following example creates `breed` as a property of the object `myDog`, and also as a variable. The first `write` statement uses `eval('breed')` without specifying an object; the string “`breed`” is evaluated without regard to any object, and the `write` method displays `Shepherd`, which is the value of the `breed` variable.

The second `write` statement uses `myDog.eval('breed')` which specifies the object `myDog;` the string “`breed"` is evaluated with regard to the `myDog` object, and the `write` method displays `"Lab"`, which is the value of the `breed` property of the `myDog` object.

```
function Dog(name,breed,color) {
      this.name=name
      this.breed=breed
      this.color=color
}
myDog = new Dog("Gabby")
myDog.breed="Lab"
var breed='Shepherd'
Console.Write(eval('breed'))
Console.Write(myDog.eval('breed'))
```

The following example uses `eval` within a function that defines an object type, `stone`. The statement `flint = new stone("x=42")` creates the object `flint` with the properties `x`, `y`, `z`, and `z2`. The `write` statements display the values of these properties as 42, 43, 44, and 45, respectively.

```
function stone(str) {
      this.eval("this."+str)
      this.eval("this.y=43")
      this.z=44
      this["z2"] = 45
}
flint = new stone("x=42")
Console.Write(flint.x is " + flint.x)
Console.Write(flint.y is " + flint.y)
Console.Write(flint.z is " + flint.z)
Console.Write(flint.z2 is " + flint.z2)
```

toString

Returns a string representing the specified object.

Applies to

`Object`

Syntax

`toString()`

`toString(radix)`

Parameters

`radix`

(Optional) An integer between 2 and 16 specifying the base to use for representing numeric values.

Description

Every object has a `toString` method that is automatically called when it is to be represented as a text value or when an object is referred to in a string concatenation. For example, the following examples require `theDog` to be represented as a string:

```
Console.Write(theDog)
Console.Write("The dog is " + theDog)
```

You can use `toString` within your own code to convert an object into a string, and you can create your own function to be called in place of the default `toString` method.

- **Built-in toString methods**

  Every object type has a built-in `toString` method, which JavaScript calls whenever it needs to convert an object to a string. If an object has no string value and no user-defined `toString` method, `toString` returns `[object type]`, where `type` is the object type or the name of the constructor function that created the object.

  Some built-in classes have special definitions for their toString methods. See the descriptions of this method for these objects:

- **User-defined toString methods**

  You can create a function to be called in place of the default `toString` method. The `toString` method takes no arguments and should return a string. The `toString` method you create can be any value you want, but it will be most useful if it carries information about the object.

  The following code defines the `Dog` object type and creates `theDog,` an object of type `Dog`:

  ```
  function Dog(name,breed,color,sex) {
        this.name=name
        this.breed=breed
        this.color=color
        this.sex=sex
  }
  theDog = new Dog("Gabby","Lab","chocolate","girl")
  ```

  The following code creates `dogToString,` the function that will be used in place of the default `toString` method. This function generates a string containing each property, of the form `property = value;`.

  ```
  function dogToString() {
        var ret = "Dog " + this.name + " is ["
        for (var prop in this)
              ret += "  " + prop + " is " + this[prop] + ";"
        return ret + "]"
  }
  ```

  The following code assigns the user-defined function to the object's `toString` method:

  ```
  Dog.prototype.toString = dogToString
  ```

With the preceding code in place, any time `theDog` is used in a string context, JavaScript automatically calls the `dogToString` function, which returns the following string:

```
    Dog Gabby is [ name is Gabby; breed is Lab; color is chocolate; sex is girl;
toString is function dogToString() { var ret = "Object " + this.name + " is ["; for
(var prop in this) { ret += " " + prop + " is " + this[prop] + ";"; } return ret +
"]"; } ;]
```

An object's `toString` method is usually invoked by JavaScript, but you can invoke it yourself as follows:

```
    alert(theDog.toString())
```

Examples

The following example prints the string equivalents of the numbers 0 through 9 in decimal and binary.

```
for (x = 0; x < 10; x++) {
      ("Decimal: ", x.toString(10), " Binary: ",
Console.write
            x.toString(2))
}
```

The preceding example produces the following output:

```
Decimal: 0 Binary: 0
Decimal: 1 Binary: 1
Decimal: 2 Binary: 10
Decimal: 3 Binary: 11
Decimal: 4 Binary: 100
Decimal: 5 Binary: 101
Decimal: 6 Binary: 110
Decimal: 7 Binary: 111
Decimal: 8 Binary: 1000
Decimal: 9 Binary: 1001
```

See also

`Object.valueOf`

unwatch

Removes a watchpoint set with the `watch` method.

Applies to

`Object`

Syntax

`unwatch(prop)`

Parameters

`prop`

The name of a property of the object.

Example

See: `Object:watch`

valueOf

Returns the primitive value of the specified object.

Applies to

`Object`

Syntax

```
valueOf()
```

Parameters

None

Description

Every object has a `valueOf` method that is automatically called when it is to be represented as a primitive value. If an object has no primitive value, `valueOf` returns the object itself.

You can use `valueOf` within your own code to convert an object into a primitive value, and you can create your own function to be called in place of the default `valueOf` method.

Every object type has a built-in `valueOf` method, which JavaScript calls whenever it needs to convert an object to a primitive value.

You rarely need to invoke the `valueOf` method yourself. JavaScript automatically invokes it when encountering an object where a primitive value is expected.

The following table shows object types for which the `valueOf` method is most useful. Most other objects have no primitive value.

Object Type	Value Returned by `valueOf`
Number	Primitive numeric value associated with the object.
Boolean	Primitive boolean value associated with the object.
String	String associated with the object.
Function	Function reference associated with the object. For example, `typeof funObj` returns `object`, but `typeof funObj.valueOf()` returns `function`.

You can create a function to be called in place of the default `valueOf` method. Your function must take no arguments.

Suppose you have an object type `myNumberType` and you want to create a `valueOf` method for it. The following code assigns a user-defined function to the object's `valueOf` method:

```
myNumberType.prototype.valueOf = new Function(functionText)
```

With the preceding code in place, any time an object of type myNumberType is used in a context where it is to be represented as a primitive value, JavaScript automatically calls the function defined in the preceding code.

An object's valueOf method is usually invoked by JavaScript, but you can invoke it yourself as follows:

```
myNumber.valueOf()
```

Tip: Objects in string contexts convert via the toString method, which is different from String objects converting to string primitives using valueOf. All string objects have a string conversion, if only [object type]. But many objects do not convert to number, boolean, or function.

watch

Watches for a property to be assigned a value and runs a function when that occurs.

Applies to

Object

Syntax

```
watch(prop, handler)
```

Parameters

prop

The name of a property of the object.

handler

A function to call.

Description

Watches for assignment to a property named prop in this object, calling handler(prop, oldval, newval) whenever prop is set and storing the return value in that property. A watchpoint can filter (or nullify) the value assignment, by returning a modified newval (or oldval).

If you delete a property for which a watchpoint has been set, that watchpoint does not disappear. If you later recreate the property, the watchpoint is still in effect.

To remove a watchpoint, use the unwatch method.

Example

```
o = {p:1}
o.watch("p",
      function (id,oldval,newval) {
            Console.Write("o." + id + " changed from "
                  + oldval + " to " + newval)
            return newval
      })
o.p = 2
o.p =
delete o.p
o.p = 4
o.unwatch('p')
o.p = 5
```

This script displays the following:

```
o.p changed from 1 to 2
o.p changed from 2 to 3
o.p changed from 3 to 4
```

String

An object representing a series of characters in a string. String is a core object.

Created by

The String constructor:

```
new String(string);
```

Parameters

`string`

Any string.

Description

The `String` object is a built-in JavaScript object. You an treat any JavaScript string as a `String` object.

A string can be represented as a literal enclosed by single or double quotation marks; for example, "Hyperion" or 'Hyperion'.

Examples

- String Variable

  The following statement creates a string variable:

  ```
  var last_name = "Schaefer"
  ```

- String Object Properties

  The following statements evaluate to 8, `"SCHAEFER,"` and `"schaefer"`:

  ```
  last_name.length
  last_name.toUpperCase()
  last_name.toLowerCase()
  ```

- Accessing individual characters in a string

  You can think of a string as an array of characters. In this way, you can access the individual characters in the string by indexing that array. For example, the following code:

  ```
      var myString = "Hello"
      Console.Write ("The first character in the string is " +     myString[0])
  ```

  displays:

  ```
  "The first character in the string is H"
  ```

String Properties

The following example displays a summary of the string properties. Detailed descriptions of each property follow the table.

Property	Description
length	Reflects the length of the string.
prototype	Allows the addition of properties to a `String` object.

length

The length of the string. The `length` property is read-only.

Property of

`String`

Description

For a null string, length is 0.

Example

The following example displays 8 in an Alert dialog box:

```
var x="Netscape"
Alert("The string length is " + x.length)
```

prototype

Represents the prototype for this class. You can use the prototype to add properties or methods to all instances of a class. For information on prototypes, see `Function.prototype`.

Property of

`String`

String Methods

The following example displays a summary of the string methods. Detailed descriptions of each method follow the table.

Method	Description
anchor	Creates an HTML anchor that is used as a hypertext target.
big	Causes a string to be displayed in a big font as if it were in a `BIG` tag.
blink	Causes a string to blink as if it were in a `BLINK` tag.
bold	Causes a string to be displayed as if it were in a `B` tag.
charAt	Returns the character at the specified index.
charCodeAt	Returns a number indicating the ISO-Latin-1 codeset value of the character at the given index.
concat	Combines the text of two strings and returns a new string.
fixed	Causes a string to be displayed in fixed-pitch font as if it were in a `TT` tag.
fontcolor	Causes a string to be displayed in the specified color as if it were in a `<FONT COLOR=color>` tag.
fontsize	Causes a string to be displayed in the specified font size as if it were in a `<FONT SIZE=size>` tag.
fromCharCode	Returns a string from the specified sequence of numbers that are ISO-Latin-1 codeset values.
indexOf	Returns the index within the calling `String` object of the first occurrence of the specified value.
italics	Causes a string to be italic, as if it were in an `I` tag.
lastIndexOf	Returns the index within the calling `String` object of the last occurrence of the specified value.
link	Creates an HTML hypertext link that requests another URL.
match	Used to match a regular expression against a string.

Method	Description
replace	Used to find a match between a regular expression and a string, and to replace the matched substring with a new substring.
search	Executes the search for a match between a regular expression and a specified string.
slice	Extracts a section of a string and returns a new string.
small	Causes a string to be displayed in a small font, as if it were in a `SMALL` tag.
split	Splits a `String` object into an array of strings by separating the string into substrings.
strike	Causes a string to be displayed as struck-out text, as if it were in a `STRIKE` tag.
sub	Causes a string to be displayed as a subscript, as if it were in a `SUB` tag.
substr	Returns the characters in a string beginning at the specified location through the specified number of characters.
substring	Returns the characters in a string between two indexes into the string.
sup	Causes a string to be displayed as a superscript, as if it were in a `SUP` tag.
toLowerCase	Returns the calling string value converted to lowercase.
toUpperCase	Returns the calling string value converted to uppercase.

anchor

Creates an HTML anchor that is used as a hypertext target.

Applies to

`String`

Syntax

```
anchor(nameAttribute)
```

Parameters

`nameAttribute`

A string.

Description

Use the `anchor` method with `Console.Write` to programmatically create and display an anchor in a document. Create the anchor with the `anchor` method, and then call `write` to display the anchor in a document.

In the syntax, the `text` string represents the literal text that you want the user to see. The `nameAttribute` string represents the `NAME` attribute of the `A` tag.

Anchors created with the `anchor` method become elements in the `document.anchors array`.

Examples

The following example opens the msgWindow window and creates an anchor for the table of contents:

```
var myString="Table of Contents"
Write(myString.anchor("contents_anchor"))
```

The previous example produces the same output as the following HTML:

```
<A NAME="contents_anchor">Table of Contents</A>
```

See also

String:link

big

Causes a string to be displayed in a big font as if it were in a BIG tag.

Applies to

String

Syntax

```
big()
```

Parameters

None

Description

Use the big method with the Write method to format and display a string in a document.

Example

The following example uses string methods to change the size of a string:

```
var worldString="Hello, world"
Console.Write(worldString.small())
Console.Write(worldString.big())
Console.Write(worldString.fontsize(7))
```

The previous example produces the same output as the following HTML:

```
<SMALL>Hello, world</SMALL>
<BIG>Hello, world</BIG>
<FONTSIZE=7>Hello, world</FONTSIZE>
```

See also

String.fontsize, String.small

blink

Causes a string to blink as if it were in a BLINK tag.

Applies to

String

Syntax

```
blink()
```

Parameters

None

Description

Use the blink method with the Write method to format and display a string in a document.

Example

The following example uses string methods to change the formatting of a string:

```
var worldString="Hello, world"
Console.Write(worldString.blink())
Console.Write("<P>" + worldString.bold())
Console.Write("<P>" + worldString.italics())
Console.Write("<P>" + worldString.strike())
```

The previous example produces the same output as the following HTML:

```
<BLINK>Hello, world</BLINK>
<P><B>Hello, world</B>
<P><I>Hello, world</I>
<P><STRIKE>Hello, world</STRIKE>
```

See also

String.bold, String.italics, String.strike

bold

Causes a string to be displayed as bold as if it were in a B tag.

Applies to

String

Syntax

```
bold()
```

Parameters

None

Description

Use the `bold` method with the `Write` methods to format and display a string in a document.

Example

The following example uses `string` methods to change the formatting of a string:

```
var worldString="Hello, world"
Console.Write(worldString.blink())
Console.Write("<P>" + worldString.bold())
Console.Write("<P>" + worldString.italics())
Console.Write("<P>" + worldString.strike())
```

The previous example produces the same output as the following HTML:

```
<BLINK>Hello, world</BLINK>
<P><B>Hello, world</B>
<P><I>Hello, world</I>
<P><STRIKE>Hello, world</STRIKE>
```

See also

`String:blink`, `String:italics`, `String.strike`

charAt

Returns the specified character from the string.

Applies to

`String`

Syntax

`charAt(index)`

Parameters

`index`

An integer between 0 and 1 less than the length of the string.

Description

Characters in a string are indexed from left to right. The index of the first character is 0, and the index of the last character in a string called `stringName is stringName.length - 1`. If the `index` you supply is out of range, JavaScript returns an empty string.

Example

The following example displays characters at different locations in the string `"Brave new world"`:

```
var anyString="Brave new world"
Console.Write("The character at index 0 is " + anyString.charAt(0))
Console.Write("The character at index 1 is " + anyString.charAt(1))
Console.Write("The character at index 2 is " + anyString.charAt(2))
Console.Write("The character at index 3 is " + anyString.charAt(3))
Console.Write("The character at index 4 is " + anyString.charAt(4))
```

These lines display the following:

```
The character at index 0 is B
The character at index 1 is r
The character at index 2 is a
The character at index 3 is v
The character at index 4 is e
```

See also

String:indexOf, String.lastIndexOf, String.split

charCodeAt

Returns a number indicating the ISO-Latin-1 codeset value of the character at the given index.

Applies to

String

Syntax

```
charCodeAt(index)
```

Parameters

index

(Optional) An integer between 0 and 1 less than the length of the string. The default value is 0.

Description

The ISO-Latin-1 codeset ranges from 0 to 255. The first 0 to 127 are a direct match of the ASCII character set.

Example

The following example returns 65, the ISO-Latin-1 codeset value for A.

```
"ABC".charCodeAt(0)
```

concat

Combines the text of two strings and returns a new string.

Applies to

String

Syntax

```
concat(string2)
```

Parameters

`string1`

The first string.

`string 2`

The second string.

Description

`concat` combines the text from two strings and returns a new string. Changes to the text in one string do not affect the other string.

Example

The following example combines two strings into a new string.

```
str1="The morning is upon us. "
str2="The sun is bright."
str3=str1.concat(str2)
Console.Write(str1)
Console.Write(str2)
Console.Write(str3)
```

This writes:

```
The morning is upon us.
The sun is bright.
The morning is upon us. The sun is bright.
```

fixed

Causes a string to be displayed in fixed-pitch font as if it were in a `TT` tag.

Applies to

`String`

Syntax

```
fixed()
```

Parameters

None

Description

Use the `fixed` method with the `Write` method to format and display a string in a document.

Example

The following example uses the `fixed` method to change the formatting of a string:

```
var worldString="Hello, world"
 (worldString.fixed())
```

The previous example produces the same output as the following HTML:

```
<TT>Hello, world</TT>
```

fontcolor

Causes a string to be displayed in the specified color as if it were in a `<FONT COLOR=color>` tag.

Applies to

`String`

Syntax

`fontcolor(color)`

Parameters

`color`

A string expressing the color as a hexadecimal RGB triplet or as a string literal. String literals for color names are listed in Appendix B, "Color Values," in the JavaScript Guide.

Description

Use the `fontcolor` method with the `Write` method to format and display a string in a document.

If you express `color` as a hexadecimal RGB triplet, you must use the format `rrggbb`. For example, the hexadecimal RGB values for salmon are `red=FA`, `green=80`, and `blue=72`, so the RGB triplet for `salmon` is `"FA8072"`.

The `fontcolor` method overrides a value set in the `fgColor` property.

Examples

The following example uses the `fontcolor` method to change the color of a string:

```
var worldString="Hello, world"
Console.Write(worldString.fontcolor("maroon") +
      " is maroon in this line")
Console.Write("<P>" + worldString.fontcolor("salmon") +
      " is salmon in this line")
Console.Write("<P>" + worldString.fontcolor("red") +
      " is red in this line")
Console.Write("<P>" + worldString.fontcolor("8000") +
      " is maroon in hexadecimal in this line")
Console.Write("<P>" + worldString.fontcolor("FA8072") +
      " is salmon in hexadecimal in this line")
Console.Write("<P>" + worldString.fontcolor("FF00") +
      " is red in hexadecimal in this line")
```

The previous example produces the same output as the following HTML:

```
<FONT COLOR="maroon">Hello, world</FONT> is maroon in this line
<P><FONT COLOR="salmon">Hello, world</FONT> is salmon in this line
<P><FONT COLOR="red">Hello, world</FONT> is red in this line
<FONT COLOR="8000">Hello, world</FONT>
is maroon in hexadecimal in this line
<P><FONT COLOR="FA8072">Hello, world</FONT>
is salmon in hexadecimal in this line
<P><FONT COLOR="FF00">Hello, world</FONT>
is red in hexadecimal in this line
```

fontsize

Causes a string to be displayed in the specified font size as if it were in a `<FONT SIZE=size>` tag.

Applies to

`String`

Syntax

`fontsize(size)`

Parameters

`size`

An integer between 1 and 7, a string representing a signed integer between 1 and 7.

Description

Use the `fontsize` method with the `Write` method to format and display a string in a document.

When you specify `size` as an integer, you set the size of `stringName` to one of the 7 defined sizes. When you specify `size` as a string such as `"-2"`, you adjust the font size of `stringName` relative to the size set in the `BASEFONT` tag.

Example

The following example uses `string` methods to change the size of a string:

```
var worldString="Hello, world"
Console.Write(worldString.small())
Console.Write("<P>" + worldString.big())
Console.Write("<P>" + worldString.fontsize(7))
```

The previous example produces the same output as the following HTML:

```
<SMALL>Hello, world</SMALL>
<P><BIG>Hello, world</BIG>
```

```
<P><FONTSIZE=7>Hello, world</FONTSIZE>
```

See also

`String.big`, `String.small`

fromCharCode

Returns a string created by using the specified sequence ISO-Latin-1 codeset values.

Applies to

`String`

Syntax

```
fromCharCode(num1, ..., numN)
```

Parameters

`num1...numN`

A sequence of numbers that are ISO-Latin-1 codeset values.

Description

This method returns a string and not a `String` object.

`fromCharCode` is a static method of `String`. As a result, you always use it as `String.fromCharCode()`, rather than as a method of a `String` object you create.

Examples

The following example returns the string "ABC".

```
String.fromCharCode(65,66,67)
```

indexOf

Returns the index within the calling `String` object of the first occurrence of the specified value, starting the search at `fromIndex`, or -1 if the value is not found.

Applies to

`String`

Syntax

```
indexOf(searchValue, fromIndex)
```

Parameters

`searchValue`

A string representing the value for which to search.

`fromIndex`

(Optional) The location within the calling string to start the search from. It can be any integer between 0 and 1 less than the length of the string. The default value is 0.

Description

Characters in a string are indexed from left to right. The index of the first character is 0, and the index of the last character of a string called `stringName is stringName.length - 1`.

If `stringName` contains an empty string (`""`), `indexOf` returns an empty string.

The `indexOf method` is case-sensitive. For example, the following expression returns -1:

```
"Blue Whale".indexOf("blue")
```

Examples

The following example uses `indexOf` and `lastIndexOf` to locate values in the string "`Brave new world.`"

```
var anyString="Brave new world"
//Displays 8
Console.Write("<P>The index of the first w from the beginning is " +
      anyString.indexOf("w"))
//Displays 10
Console.Write("<P>The index of the first w from the end is " +
      anyString.lastIndexOf("w"))
//Displays 6
Console.Write("<P>The index of 'new' from the beginning is " +
      anyString.indexOf("new"))
//Displays 6
Console.Write("<P>The index of 'new' from the end is " +
      anyString.lastIndexOf("new"))
```

The following example defines two string variables. The variables contain the same string except that the second string contains uppercase letters. The first `write` method displays 19. But because the `indexOf` method is case-sensitive, the string `"cheddar"` is not found in `myCapString`, so the second `write` method displays -1.

```
myString="brie, pepper jack, cheddar"
myCapString="Brie, Pepper Jack, Cheddar"
Console.Write('myString.indexOf("cheddar") is ' +
      myString.indexOf("cheddar"))
Console.Write('myCapString.indexOf("cheddar") is ' +
      myCapString.indexOf("cheddar"))
```

The following example sets `count` to the number of occurrences of the letter `x` in the string `str`:

```
count = 0;
pos = str.indexOf("x");
while ( pos != -1 ) {
      count++;
      pos = str.indexOf("x",pos+1);
}
```

See also

String:charAt, String:lastIndexOf, String:split

italics

Causes a string to be italic, as if it were in an I tag.

Applies to

String

Syntax

```
italics()
```

Parameters

None

Description

Use the italics method with the Write method to format and display a string in a document.

Example

The following example uses string methods to change the formatting of a string:

```
var worldString="Hello, world"
Console.Write(worldString.blink())
Console.Write(worldString.bold())
Console.Write(worldString.italics())
Console.Write(worldString.strike())
```

The previous example produces the same output as the following HTML:

```
<BLINK>Hello, world</BLINK>
<P><B>Hello, world</B>
<P><I>Hello, world</I>
<P><STRIKE>Hello, world</STRIKE>
```

See also

String:blink, String:bold, String:strike

lastIndexOf

Returns the index within the calling String object of the last occurrence of the specified value. The calling string is searched backward, starting at fromIndex, or -1 if not found.

Applies to

String

Syntax

```
lastIndexOf(searchValue, fromIndex)
```

Parameters

`searchValue`

A string representing the value for which to search.

`fromIndex`

(Optional) The location within the calling string to start the search from. It can be any integer between 0 and 1 less than the length of the string. The default value is 1 less than the length of the string.

Description

Characters in a string are indexed from left to right. The index of the first character is 0, and the index of the last character is `stringName.length - 1`.

The `lastIndexOf` method is case-sensitive. For example, the following expression returns -1:

```
"Blue Whale, Killer Whale".lastIndexOf("blue")
```

Example

The following example uses `indexOf` and `lastIndexOf` to locate values in the string `"Brave new world."`

```
var anyString="Brave new world"
//Displays 8
Console.Write("The index of the first w from the beginning is " +
      anyString.indexOf("w"))
//Displays 10
Console.Write("The index of the first w from the end is " +
      anyString.lastIndexOf("w"))
//Displays 6
Console.Write("The index of 'new' from the beginning is " +
      anyString.indexOf("new"))
//Displays 6
Console.Write("The index of 'new' from the end is "
      anyString.lastIndexOf("new"))
```

See also

`String:charAt`, `String:indexOf`, `String:split`

link

Creates an HTML hypertext link that requests another URL.

Applies to

`String`

Syntax

`link(hrefAttribute)`

Parameters

`hrefAttribute`

Any string that specifies the HREF attribute of the A tag; it should be a valid URL (relative or absolute).

Description

Use the `link` method to programmatically create a hypertext link, and then call to the `write` method display the link in a document.

Example

The following example displays the word "Hyperion" as a hypertext link that returns the user to the Hyperion Web site:

```
var hotText="Hyperion"
var URL="http://www.hyperion.com"
Console.Write("Click to return to " + hotText.link(URL))
```

The previous example produces the same output as the following HTML:

```
Click to return to <A HREF="http://www.hyperion.com">Hyperion</A>
```

See also

`String:anchor`

match

Used to match a regular expression against a string.

Applies to

`String`

Syntax

`match(regexp)`

Parameters

`regexp`

Name of the regular expression. It can be a variable name or literal.

Description

If you want to execute a global match, or a case insensitive match, include the `g` (for global) and `i` (for ignore case) flags in the regular expression. These can be included separately or together. The following two examples below show how to use these flags with `match`.

Tip: If you execute a match simply to find true or false, use `String.search` or the regular expression `test` method.

Examples

In the following example, `match` is used to find 'Chapter' followed by 1 or more numeric characters followed by a decimal point and numeric character 0 or more times. The regular expression includes the `i` flag so that case will be ignored.

```
str = "For more information, see Chapter 3.4.5.1";
re = /(chapter \d+(\.\d)*)/i;
found = str.match(re);
Console.Write(found);
```

This returns the array containing Chapter 3.4.5.1, Chapter 3.4.5.1,.1

'Chapter 3.4.5.1' is the first match and the first value remembered from (Chapter \d+(\.\d)*).

'.1' is the second value remembered from (\.\d).

The following example demonstrates the use of the global and ignore case flags with `match`.

```
str = "abcDdcba";
newArray = str.match(/d/gi);
Console.Write(newArray);
```

The returned array contains D, d.

replace

Used to find a match between a regular expression and a string, and to replace the matched substring with a new substring.

Applies to

`String`

Syntax

```
replace(regexp, newSubStr)
```

Parameters

`regexp`

The name of the regular expression. It can be a variable name or a literal.

`newSubStr`

The string to put in place of the string found with `regexp`. This string can include the `RegExp` properties `$1, ..., $9, lastMatch`, `lastParen`, `leftContext`, and `rightContext`.

Description

This method does not change the `String` object it is called on; it simply returns a new string.

If you want to execute a global search and replace, or a case insensitive search, include the g (for global) and i (for ignore case) flags in the regular expression. These can be included separately or together. The following two examples below show how to use these flags with replace.

Examples

In the following example, the regular expression includes the global and ignore case flags which permits replace to replace each occurrence of 'apples' in the string with 'oranges.'

```
re = /apples/gi;
str = "Apples are round, and apples are juicy.";
newstr=str.replace(re, "oranges");
Console.Write(newstr)
```

This prints "oranges are round, and oranges are juicy."

In the following example, the regular expression is defined in replace and includes the ignore case flag.

```
str = "Twas the night before Xmas...";
newstr=str.replace(/xmas/i, "Christmas");
Console.Write(newstr)
```

This prints:

```
"Twas the night before Christmas..."
```

The following script switches the words in the string. For the replacement text, the script uses the values of the $1 and $2 properties.

```
re = /(\w+)\s(\w+)/;
str = "John Smith";
newstr = str.replace(re, "$2, $1");
Console.Write(newstr)
```

This prints:

```
"Smith, John"
```

search

Executes the search for a match between a regular expression and this String object.

Applies to

String

Syntax

```
search(regexp)
```

Parameters

regexp

Name of the regular expression. It can be a variable name or a literal.

Description

If successful, `search` returns the index of the match inside the string. Otherwise, it returns -1.

When you want to know whether a pattern is found in a string use `search` (similar to the regular expression `test method`); for more information (but slower execution) use `match` (similar to the regular expression `exec` method).

Example

The following example prints a message which depends on the success of the test.

```
function testinput(re, str){
      if (str.search(re) != -1)
            midstring = " contains ";
      else
            midstring = " does not contain ";
      Console.Write (str + midstring + re.source);
}
```

slice

Extracts a section of a string and returns a new string.

Applies to

`String`

Syntax

`slice(beginslice,endSlice)`

Parameters

`beginSlice`

The zero-based index at which to begin extraction.

`endSlice`

(Optional) The zero-based index at which to end extraction. If omitted, slice extracts to the end of the string.

Description

`slice` extracts the text from one string and returns a new string. Changes to the text in one string do not affect the other string.

`slice` extracts up to but not including `endSlice. string.slice(1,4)` extracts the second character through the fourth character (characters indexed 1, 2, and 3).

As a negative index, `endSlice` indicates an offset from the end of the string. `string.slice(2,-1)` extracts the third character through the second to last character in the string.

Example

The following example uses `slice` to create a new string.

```
str1="The morning is upon us. "
str2=str1.slice(3,-5)
Console.Write(str2)
```

This writes:

```
The morning is upon us
```

small

Causes a string to be displayed in a small font, as if it were in a `SMALL` tag.

Applies to

`String`

Syntax

```
small()
```

Parameters

None

Description

Use the `small` method with the `Write` method to format and display a string in a document.

Example

The following example uses `string` methods to change the size of a string:

```
var worldString="Hello, world"
Console.Write(worldString.small())
Console.Write("<P>" + worldString.big())
Console.Write("<P>" + worldString.fontsize(7))
```

The previous example produces the same output as the following HTML:

```
<SMALL>Hello, world</SMALL>
<P><BIG>Hello, world</BIG>
<P><FONTSIZE=7>Hello, world</FONTSIZE>
```

See also

`String:big`, `String:fontsize`

split

Splits a `String` object into an array of strings by separating the string into substrings.

Applies to

`String`

Syntax

`split(separator, limit)`

Parameters

`separator`

(Optional) Specifies the character to use for separating the string. The separator is treated as a string. If separator is omitted, the array returned contains one element consisting of the entire string. The separator can either be a single separator or a multi-character separators.

`limit`

(Optional) Integer specifying a limit on the number of splits to be found.

Description

The `split` method returns the new array.

When found, `separator` is removed from the string and the substrings are returned in an array. If `separator` is omitted, the array contains one element consisting of the entire string.

It can take a regular expression argument, as well as a fixed string, by which to split the object string. If `separator` is a regular expression, any included parentheses cause submatches to be included in the returned array.

It can take a limit count so that it won't include trailing empty elements in the resulting array.

Examples

The following example defines a function that splits a string into an array of strings using the specified separator. After splitting the string, the function displays messages indicating the original string (before the split), the separator used, the number of elements in the array, and the individual array elements.

```
function splitString (stringToSplit,separator) {
      arrayOfStrings = stringToSplit.split(separator)
      Console.Write ('<P>The original string is: "' + stringToSplit + '"')
      Console.Write ('<BR>The separator is: "' + separator + '"')
      Console.Write ("<BR>The array has " + arrayOfStrings.length + " elements: ")
      for (var i=0; i < arrayOfStrings.length; i++) {
            Console.Write (arrayOfStrings[i] + " / ")
      }
}
var tempestString="Oh brave new world that has such people in it."
var monthString="Jan,Feb,Mar,Apr,May,Jun,Jul,Aug,Sep,Oct,Nov,Dec"
var space=" "
var comma=","
splitString(tempestString,space)
splitString(tempestString)
splitString(monthString,comma)
```

This example produces the following output:

```
The original string is: "Oh brave new world that has such people in it."
The separator is: " "
The array has 10 elements: Oh / brave / new / world / that / has / such / people / in /
it. /
The original string is: "Oh brave new world that has such people in it."
The separator is: "undefined"
The array has 1 elements: Oh brave new world that has such people in it. /
The original string is: "Jan,Feb,Mar,Apr,May,Jun,Jul,Aug,Sep,Oct,Nov,Dec"
The separator is: ","
The array has 12 elements: Jan / Feb / Mar / Apr / May / Jun / Jul / Aug / Sep / Oct /
Nov / Dec /
```

Consider the following script:

```
str="She sells   seashells \nby   the\n seashore"
Console.Write(str )
a=str.split(" ")
Console.Write(a)
```

Using `LANGUAGE="JavaScript1.2"`, this script produces

```
"She", "sells", "seashells", "by", "the", "seashore"
```

In the following example, `split` looks for 0 or more spaces followed by a semicolon followed by 0 or more spaces and, when found, removes the spaces from the string. `nameList` is the array returned as a result of `split`.

```
names = "Harry  Trump  ;Fred Barney; Helen   Rigby ; Bill Abel ;Chris Hand ";
Console.Write (names , "    ");
re = /\s*;\s*/;
nameList = names.split (re);
Console.Write(nameList);
```

This prints two lines; the first line prints the original string, and the second line prints the resulting array.

```
Harry Trump ;Fred Barney; Helen Rigby ; Bill Abel ;Chris Hand
Harry Trump,Fred Barney,Helen Rigby,Bill Abel,Chris Hand
```

In the following example, `split` looks for 0 or more spaces in a string and returns the first 3 splits that it finds.

```
myVar = "  Hello World. How are you doing?    ";
splits = myVar.split(" ", 3);
Console.Write(splits)
```

This script displays the following:

```
["Hello", "World.", "How"]
```

See also

String.charAt, String.indexOf, String.lastIndexOf

strike

Causes a string to be displayed as struck-out text, as if it were in a `STRIKE` tag.

Applies to

String

Syntax

```
strike()
```

Parameters

None

Description

Use the strike method with the Write method to format and display a string in a document.

Examples

The following example uses string methods to change the formatting of a string:

```
var worldString="Hello, world"
Console.Write(worldString.blink())
Console.Write("<P>" + worldString.bold())
Console.Write("<P>" + worldString.italics())
Console.Write("<P>" + worldString.strike())
```

The previous example produces the same output as the following HTML:

```
<BLINK>Hello, world</BLINK>
<P><B>Hello, world</B>
<P><I>Hello, world</I>
<P><STRIKE>Hello, world</STRIKE>
```

See also

String.blink, String.bold, String.italics

sub

Causes a string to be displayed as a subscript, as if it were in a SUB tag.

Applies to

String

Syntax

```
sub()
```

Parameters

None

Description

Use the sub method with the Write method to format and display a string in a document.

Example

The following example uses the `sub` and `sup` methods to format a string:

```
var superText="superscript"
var subText="subscript"
Console.Write("This is what a " + superText.sup() + " looks like.")
Console.Write("<P>This is what a " + subText.sub() + " looks like.")
```

The previous example produces the same output as the following HTML:

```
This is what a <SUP>superscript</SUP> looks like.
<P>This is what a <SUB>subscript</SUB> looks like.
```

See also

`String:sup`

substr

Returns the characters in a string beginning at the specified location through the specified number of characters.

Applies to

`String`

Syntax

`substr(start, length)`

Parameters

`start`

Location at which to begin extracting characters.

`length`

(Optional) The number of characters to extract.

Description

`start` is a character index. The index of the first character is 0, and the index of the last character is 1 less than the length of the string. `substr` begins extracting characters at `start` and collects `length` number of characters.

If `start` is positive and is the length of the string or longer, `substr` returns no characters.

If `start` is negative, `substr` uses it as a character index from the end of the string. If `start` is negative and `abs(start)` is larger than the length of the string, `substr` uses 0 is the start index.

If `length` is 0 or negative, `substr` returns no characters. If `length` is omitted, `start` extracts characters to the end of the string.

Example

Consider the following script:

```
str = "abcdefghij"
Console.Write("(1,2): ", str.substr(1,2))
Console.Write("(-2,2): ", str.substr(-2,2))
Console.Write("(1): ", str.substr(1))
Console.Write("(-20, 2): ", str.substr(1,20))
Console.Write("(20, 2): ", str.substr(20,2))
```

This script displays:

```
(1,2): bc
(-2,2): ij
(1): bcdefghij
(-20, 2): bcdefghij
(20, 2):
```

See also

`String`: `substring`

substring

Returns a subset of a `String` object.

Applies to

String

Syntax

substring(indexA, indexB)

Parameters

`indexA`

An integer between 0 and 1 less than the length of the string.

`indexB`

An integer between 0 and 1 less than the length of the string.

Description

`substring` extracts characters from `indexA` up to but not including `indexB`. In particular:

- If `indexA` is less than 0, `indexA` is treated as if it were 0.
- If `indexB` is greater than `stringName.length`, `indexB` is treated as if it were `stringName.length`.
- If `indexA` equals `indexB`, `substring` returns an empty string.
- If `indexB` is omitted, `substring` extracts characters to the end of the string.

- If indexA is greater than indexB, JavaScript returns a substring beginning with indexB and ending with indexA - 1.

Examples

The following example uses substring to display characters from the string "Netscape":

```
var anyString="Netscape"
//Displays "Net"
Console.Write(anyString.substring(0,3))
Console.Write(anyString.substring(3,0))
//Displays "cap"
Console.Write(anyString.substring(4,7))
Console.Write(anyString.substring(7,4))
//Displays "Netscap"
Console.Write(anyString.substring(0,7))
//Displays "Netscape"
Console.Write(anyString.substring(0,8))
Console.Write(anyString.substring(0,10))
```

The following example replaces a substring within a string. It will replace both individual characters and substrings. The function call at the end of the example changes the string "Brave New World" into "Brave New Web".

```
function replaceString(oldS,newS,fullS) {
// Replaces oldS with newS in the string fullS
      for (var i=0; i<fullS.length; i++) {
            if (fullS.substring(i,i+oldS.length) == oldS) {
                  fullS = fullS.substring(0,i)+newS+fullS.substring(i
+oldS.length,fullS.length)
            }
      }
      return fullS
}
replaceString("World","Web","Brave New World")
```

sup

Causes a string to be displayed as a superscript, as if it were in a SUP tag.

Applies to

String

Syntax

sup()

Parameters

None

Description

Use the sup method with the Write method to format and display a string in a document.

Examples

The following example uses the sub and sup methods to format a string:

```
var superText="superscript"
var subText="subscript"
Console.Write("This is what a " + superText.sup() + " looks like.")
Console.Write("<P>This is what a " + subText.sub() + " looks like.")
```

The previous example produces the same output as the following HTML:

```
This is what a <SUP>superscript</SUP> looks like.
<P>This is what a <SUB>subscript</SUB> looks like.
```

See also

String.sub

toLowerCase

Returns the calling string value converted to lowercase.

Applies to

String

Syntax

toLowerCase()

Parameters

None

Description

The toLowerCase method returns the value of the string converted to lowercase. toLowerCase does not affect the value of the string itself.

Example

The following example displays the lowercase string "alphabet":

```
var upperText="ALPHABET"
Console.Write(upperText.toLowerCase())
```

See also

String:toUpperCase

toUpperCase

Returns the calling string value converted to uppercase.

Applies to

`String`

Syntax

`toUpperCase()`

Parameters

None

Description

The `toUpperCase` method returns the value of the string converted to uppercase. `toUpperCase` does not affect the value of the string itself.

Examples

The following example displays the string `"ALPHABET"`:

```
var lowerText="alphabet"
Console.Write(lowerText.toUpperCase())
```

See also

`String.toLowerCase`

Regular Expression

A regular expression object contains the pattern of a regular expression. It has properties and methods for using that regular expression to find and replace matches in strings.

In addition to the properties of an individual regular expression object that you create using the `RegExp` constructor function, the predefined `RegExp` object has static properties that are set whenever any regular expression is used. Regular expression is a core object.

Note: In JavaScript 1.5, some characters such as pipes (|) which were treated a characters in certain circumstances are now treated as regular expression symbols. See also http://developer.mozilla.org/en/docs/Core_JavaScript_1.5_Reference.

Created by

A literal text format or the `RegExp` constructor function.

The literal format is used as follows:

`/pattern/flags`

The constructor function is used as follows:

`new RegExp("`*`pattern`*`", "`*`flags`*`")`

Parameters

```
pattern
```

The text of the regular expression

```
flags
```

(Optional) If specified, flags can have one of the following 3 values:

- **G** —global match
- **i**—ignore case
- **gi**—both global match and ignore case

Notice that the parameters to the literal format do not use quotation marks to indicate strings, while the parameters to the constructor function do use quotation marks. So the following expressions create the same regular expression:

```
/ab+c/i
new RegExp("ab+c", "i")
```

Description

When using the constructor function, the normal string escape rules (preceding special characters with \ when included in a string) are necessary. For example, the following are equivalent:

```
re = new  RegExp("\\w+")
re = /\w+/
```

The following table provides a complete list and description of the special characters that can be used in regular expressions.

Character	Description
\	For characters that are usually treated literally, indicates that the next character is special and not to be interpreted literally. For example, `/b/` matches the character 'b'. By placing a backslash in front of b, that is by using `/\b/`, the character becomes special to mean match a word boundary -or- For characters that are usually treated specially, indicates that the next character is not special and should be interpreted literally. For example, * is a special character that means 0 or more occurrences of the preceding character should be matched; for example, `/a*/` means match 0 or more a's. To match * literally, precede the it with a backslash; for example, `/a\*/` matches 'a*'.
^	Matches beginning of input or line. For example, `/^A/` does not match the 'A' in "an A," but does match it in "An A."
$	Matches end of input or line. For example, `/t$/` does not match the 't' in "eater", but does match it in "eat"
*	Matches the preceding character 0 or more times. For example, `/bo*/` matches 'boooo' in "A ghost booooed" and 'b' in "A bird warbled", but nothing in "A goat grunted".
+	Matches the preceding character 1 or more times. Equivalent to {1,}. For example, `/a+/` matches the 'a' in "candy" and all the a's in "caaaaaaandy."

Character	Description
?	Matches the preceding character 0 or 1 time.For example, `/e?le?/` matches the 'el' in "angel" and the 'le' in "angle."
.	(The decimal point) matches any single character except the newline character. For example, `/.n/` matches 'an' and 'on' in "nay, an apple is on the tree", but not 'nay'.
(x)	Matches 'x' and remembers the match. For example, `/(foo)/` matches and remembers 'foo' in "foo bar." The matched substring can be recalled from the resulting array's elements [1], ..., [n], or from the predefined `RegExp` object's properties $1, ..., $9.
x|y	Matches either 'x' or 'y'. For example, `/green|red/` matches 'green' in "green apple" and 'red' in "red apple."
{n}	Where n is a positive integer. Matches exactly n occurrences of the preceding character. For example, `/a{2}/` doesn't match the 'a' in "candy," but it matches all of the a's in "caandy," and the first two a's in "caaandy."
{n,}	Where n is a positive integer. Matches at least n occurrences of the preceding character. For example, `/a{2,}` doesn't match the 'a' in "candy", but matches all of the a's in "caandy" and in "caaaaaaandy."
{n,m}	Where n and m are positive integers. Matches at least n and at most m occurrences of the preceding character. For example, `/a{1,3}/` matches nothing in "cndy", the 'a' in "candy," the first two a's in "caandy," and the first three a's in "caaaaaaandy" Notice that when matching "caaaaaaandy", the match is "aaa", even though the original string had more a's in it.
[xyz]	A character set. Matches any one of the enclosed characters. You can specify a range of characters by using a hyphen. For example, `[abcd]` is the same as `[a-c]`. They match the 'b' in "brisket" and the 'c' in "ache".
[^xyz]	A negated or complemented character set. That is, it matches anything that is not enclosed in the brackets. You can specify a range of characters by using a hyphen. For example, `[^abc]` is the same as `[^a-c]`. They initially match 'r' in "brisket" and 'h' in "chop."
[\b]	Matches a backspace. (Not to be confused with `\b.`)
\b	Matches a word boundary, such as a space. (Not to be confused with `[\b]`.).For example, `/\bn\w/` matches the 'no' in "noonday"; `/\wy\b/` matches the 'ly' in "possibly yesterday."
\B	Matches a non-word boundary. For example, `/\w\Bn/` matches 'on' in "noonday", and `/y\B\w/` matches 'ye' in "possibly yesterday."
\c*X*	Where *X* is a control character. Matches a control character in a string. For example, `/\cM/` matches control-M in a string.
\d	Matches a digit character. Equivalent to [0-9]. For example, `/\d/` or `/[0-9]/` matches '2' in "B2 is the suite number. "
\D	Matches any non-digit character. Equivalent to [^0-9]. For example, `/\D/` or `/[^0-9]/` matches 'B' in "B2 is the suite number."

Character	Description
\f	Matches a form-feed.
\n	Matches a linefeed.
\r	Matches a carriage return.
\s	Matches a single white space character, including space, tab, form feed, line feed. Equivalent to `[ \f\n\r\t\v]`. For example, `/\s\w*/` matches ' bar' in "foo bar."
\S	Matches a single character other than white space. Equivalent to `[^ \f\n\r\t\v]`. For example, `/\S/\w*` matches 'foo' in "foo bar."
\t	Matches a tab.
\v	Matches a vertical tab.
\w	Matches any alphanumeric character including the underscore. Equivalent to `[A-Za-z0-9_]`. For example, `/\w/` matches 'a' in "apple," '5' in "$5.28," and '3' in "3D."
\W	Matches any non-word character. Equivalent to `[^A-Za-z0-9_]`. For example, `/\W/` or / `[^$A-Za-z0-9_]/` matches '%' in "50%."
n	Where *n* is a positive integer. A back reference to the last substring matching the *n* parenthetical in the regular expression (counting left parentheses). For example, `/apple(,)\sorange\1/` matches 'apple, orange', in "apple, orange, cherry, peach." A more complete example follows this table. Note: If the number of left parentheses is less than the number specified in *n*, the *n* is taken as an octal escape as described in the next row.
\ooctal \xhex	Where `\ooctal` is an octal escape value or `\xhex` is a hexadecimal escape value. Allows you to embed ASCII codes into regular expressions.

The literal notation provides compilation of the regular expression when the expression is evaluated. Use literal notation when the regular expression will remain constant. For example, if you use literal notation to construct a regular expression used in a loop, the regular expression won't be recompiled on each iteration.

The constructor of the regular expression object, for example, `new RegExp("ab+c")`, provides runtime compilation of the regular expression. Use the constructor function when you know the regular expression pattern will be changing, or you don't know the pattern and are getting it from another source, such as user input. Once you have a defined regular expression, and if the regular expression is used throughout the script and may change, you can use the `compile` method to compile a new regular expression for efficient reuse.

A separate predefined RegExp object is available in each window; that is, each separate thread of JavaScript execution gets its own RegExp object. Because each script runs to completion without interruption in a thread, this assures that different scripts do not overwrite values of the RegExp object.

The predefined RegExp object contains the static properties input, multiline, lastMatch, lastParen, leftContext, rightContext, and $1 through $9. The input and multiline properties can be preset. The values for the other static properties are set after execution of the exec and test methods of an individual regular expression object, and after execution of the match and replace methods of String.

Examples

The following script uses the replace method to switch the words in the string. For the replacement text, the script uses the values of the $1 and $2 properties of the global RegExp object. Note that the RegExp object name is not be prepended to the $ properties when they are passed as the second argument to the replace method.

```
re = /(\w+)\s(\w+)/;
str = "John Smith";
newstr=str.replace(re, "$2, $1");
(newstr)
```

This displays:

```
"Smith, John".
```

In the following example, RegExp.input is set by the Change event. In the getInfo function, the exec method uses the value of RegExp.input as its argument. Note that RegExp is prepended to the $ properties.

```
function getInfo() {
      re = /(\w+)\s(\d+)/;
      re.exec();
      alert(RegExp.$1 + ", your age is " + RegExp.$2);
}
Enter your first name and your age, and then press Enter.
<FORM>
<INPUT TYPE:"TEXT" NAME="NameAge" onChange="getInfo(this);">
</FORM>
</HTML>
```

Regular Expression Properties

The following table displays a summary of the regular expression properties. Note that several of these properties have both long and short (Perl-like) names. Both names always refer to the same value. Perl is the programming language from which JavaScript modeled its regular expressions. Detailed descriptions of each property follow the table.

Property	Description
$1, ..., $9	Parenthesized substring matches, if any.

Property	Description
$_	See input.
$*	See multiline.
$&	See lastMatch.
$+	See lastParen.
$'	See leftContext.
$'	See rightContext.
global	Whether to test the regular expression against all possible matches in a string, or only against the first.
ignoreCase	Whether to ignore case while attempting a match in a string.
input	The string against which a regular expression is matched.
lastIndex	The index at which to start the next match.
lastMatch	The last matched characters.
lastParen	The last parenthesized substring match, if any.
leftContext	The substring preceding the most recent match.
multiline	Whether to search in strings across multiple lines.
rightContext	The substring following the most recent match.
source	The text of the pattern.

$1, ..., $9

Properties that contain parenthesized substring matches, if any.

Property of

`RegEx`

Description

`input` is static, read-only. As a result, it is not a property of an individual regular expression object. Instead, you always use it as `RegExp.input`.

The number of possible parenthesized substrings is unlimited, but the predefined `RegExp` object can only hold the last nine. You can access all parenthesized substrings through the returned array's indexes.

These properties can be used in the replacement text for the `String.replace` method. When used this way, do not prepend them with `RegExp`. The example below illustrates this. When parentheses are not included in the regular expression, the script interprets *$n's* literally (where *n* is a positive integer).

Example

The following script uses the replace method to switch the words in the string. For the replacement text, the script uses the values of the $1 and $2 properties of the global RegExp object. Note that the RegExp object name is not be prepended to the $ properties when they are passed as the second argument to the replace method.

```
re = /(\w+)\s(\w+)/;
str = "John Smith";
newstr=str.replace(re, "$2, $1");
Console.Write(newstr)
```

This displays:

```
"Smith, John".
```

$_

See input.

$*

See multiline.

$&

See lastMatch.

$+

See lastParen.

$'

See leftContext.

$'

See rightContext.

global

Whether the "g" flag is used with the regular expression. The global property is read-only.

Property of

RegEx

Description

global is a property of an individual regular expression object.

The value of global is true if the "g" flag is used; otherwise, it is false. The "g" flag indicates that the regular expression should be tested against all possible matches in a string.

You cannot change this property directly. However, calling the compile method changes the value of this property.

ignoreCase

Whether or not the "i" flag is used with the regular expression. The ignorecase property is read-only.

Property of

RegEx

Description

ignoreCase is a property of an individual regular expression object.

The value of ignoreCase is true if the "i" flag is used; otherwise, it is false. The "i" flag indicates that case should be ignored while attempting a match in a string.

You cannot change this property directly. However, calling the compile method changes the value of this property.

input

The string against which a regular expression is matched. $_ is another name for the same property.

Property of

RegEx

Description

Because input is static, it is not a property of an individual regular expression object. Instead, you always use it as RegExp.input.

If no string argument is provided to a regular expression's exec or test methods, and if RegExp.input has a value, its value is used as the argument to that method.

The script or the browser can preset the input property. If preset and if no string argument is explicitly provided, the value of input is used as the string argument to the exec or test methods of the regular expression object. input is set by the browser in the following cases:

When an event handler is called for a TEXT form element, input is set to the value of the contained text.

When an event handler is called for a TEXTAREA form element, input is set to the value of the contained text. Note that multiline is also set to true so that the match can be executed over the multiple lines of text.

When an event handler is called for a SELECT form element, input is set to the value of the selected text.

When an event handler is called for a Link object, input is set to the value of the text between <A HREF=...> and </A>.

The value of the input property is cleared after the event handler completes.

lastIndex

A read/write integer property that specifies the index at which to start the next match.

Property of

RegEx

Description

lastIndex is a property of an individual regular expression object.

This property is set only if the regular expression used the "g" flag to indicate a global search. The following rules apply:

- If lastIndex is greater than the length of the string, regexp.test and regexp.exec fail, and lastIndex is set to 0.
- If lastIndex is equal to the length of the string and if the regular expression matches the empty string, then the regular expression matches input starting at lastIndex.
- If lastIndex is equal to the length of the string and if the regular expression does not match the empty string, then the regular expression mismatches input, and lastIndex is reset to 0.
- Otherwise, lastIndex is set to the next position following the most recent match.

For example, consider the following sequence of statements:

Table 32 lastIndex Statements

Statement	Description
re = /(hi)?/g	Matches the empty string.
re("hi")	Returns ["hi", "hi"] with lastIndex equal to 2.
re("hi")	Returns [""], an empty array whose zeroth element is the match string. In this case, the empty string because lastIndex was 2 (and still is 2) and "hi" has length 2.

lastMatch

The last matched characters. $& is another name for the same property.

Property of

`RegEx`

Description

Because `lastMatch` is static, it is not a property of an individual regular expression object. Instead, you always use it as `RegExp.lastMatch`.

lastParen

The last parenthesized substring match, if any. `$+` is another name for the same property.

Property of

`RegEx`

Description

Because `lastParen` is static (read-only), it is not a property of an individual regular expression object. Instead, you always use it as `RegExp.lastParen`.

leftContext

The substring preceding the most recent match. `$'` is another name for the same property.

Property of

`RegEx`

Description

Because `leftContext` is static (read-only), it is not a property of an individual regular expression object. Instead, you always use it as `RegExp.leftContext`.

multiline

Reflects whether or not to search in strings across multiple lines. $* is another name for the same property.

Property of

`RegEx`

Description

Because `multiline` is static, it is not a property of an individual regular expression object. Instead, you always use it as `RegExp.multiline`.

The value of `multiline` is `true` if multiple lines are searched, `false` if searches must stop at line breaks.

rightContext

The substring following the most recent match. `$'` is another name for the same property.

Property of

`RegEx`

Description

Because `rightContext` is static (read-only), it is not a property of an individual regular expression object. Instead, you always use it as `RegExp.rightContext`.

source

A read-only property that contains the text of the pattern, excluding the forward slashes and `"g"` or `"i"` flags.

Property of

`RegEx`

Description

`source` is a property of an individual regular expression object. It is read-only. You cannot change this property directly. However, calling the `compile` method changes the value of this property.

Regular Expression Methods

The following table displays a summary of the regular expression methods. Detailed descriptions of each method follow the table.

Method	Expression
compile	Compiles a regular expression object.
exec	Executes a search for a match in its string parameter.
test	Tests for a match in its string parameter.

compile

Compiles a regular expression object during execution of a script.

Applies to

`RegExp`

Syntax

```
regexp.compile(pattern, flags)
```

Parameters

`regexp`

The name of he regular expression. It can be a variable name or a literal.

`pattern`

A string containing the text of the regular expression.

`flags`

(Optional) If specified, flags can have one of the following 3 values:

- **g**—global match
- **i**—ignore case
- **gi**—both global match and ignore case

Description

Use the `compile` method to compile a regular expression created with the `RegExp` constructor function. This forces compilation of the regular expression once only which means the regular expression isn't compiled each time it is encountered. Use the `compile` method when you know the regular expression will remain constant (after getting its pattern) and will be used repeatedly throughout the script.

You can also use the `compile` method to change the regular expression during execution. For example, if the regular expression changes, you can use the `compile` method to recompile the object for more efficient repeated use.

Calling this method changes the value of the regular expression's `source`, `global`, and `ignoreCase properties`.

exec

Executes the search for a match in a specified string. Returns a result array.

Applies to:

`RegExp`

Syntax

```
regexp.exec(str)
```

```
regexp(str)
```

Parameters

`regexp`

The name of the regular expression. It can be a variable name or a literal.

`str`

(Optional) The string against which to match the regular expression. If omitted, the value of RegExp.input is used.

Description

As shown in the syntax description, a regular expression's `exec` method call be called either directly, (with `regexp.exec(str)`) or indirectly (with `regexp(str)`).

If you are executing a match simply to find `true` or `false`, use the `test` method or the `String search` method.

If the match succeeds, the `exec` method returns an array and updates properties of the regular expression object and the predefined regular expression object, `RegExp`. If the match fails, the `exec` method returns `null`.

Note: In JavaScript version 1.5, the '|' (pipe) character is treated (when not quoted) as an alternate metacharacter . In this case, the regular expression "|aaa" means "empty string OR 'aaa'". For example "|aaa" matches the string "bbb|aaa" starting before the first character (and the matched string is empty). The same occurs with "aaabbb". It matches the empty alternative before the first character. To make an older "|aaa" regular expression work in JavaScript 1.5, place quotes around the | character with a backslash.. For example, enter "|aaa" as "\|aaa", or "|Target~". In JS1.5 as "\|Target~". Also note the JavaScript version 1.4 behavior not only occurs when '|' is located at beginning of whole regular expression, but also at the beginning of regexp group. For example, you would need to change the regular expression "aaa(|bbb)" to "aaa(\|bbb)".

Consider the following example:

```
//Match one d followed by one or more b's followed by one d
//Remember matched b's and the following d
//Ignore case
myRe=/d(b+)(d)/ig;
myArray = myRe.exec("cdbBdbsbz");
```

The following table shows the results for this script:

Object	Property/Index	Description	Example
myArray		The contents of myArray	["dbBd", "bB", "d"]
index		The 0-based index of the match in the string.	1
input		The original string	cdbBdbsbz
[0]		The last matched characters	dbBd
[1], ...[n]		The parenthesized substring matches, if any. The number of possible parenthesized substrings is unlimited.	[1] = bB [2] = d
myRe	lastIndex	The index at which to start the next match.	5

Object	Property/Index	Description	Example
ignoreCase		Indicates if the "i" flag was used to ignore case	true
global		Indicates if the "g" flag was used for a global match	true
source		The text of the pattern	d(b+)(d)
RegExp	lastMatch $&	The last matched characters	dbBd
leftContext $\Q		The substring preceding the most recent match.	c
rightContext $'		The substring following the most recent match.	bsbz
$1, ...$9		The parenthesized substring matches, if any. The number of possible parenthesized substrings is unlimited, but RegExp can only hold the last nine.	$1 = bB $2 = d
lastParen $+		The last parenthesized substring match, if any.	d

If your regular expression uses the `"g"` flag, you can use the `exec` method multiple times to find successive matches in the same string. When you do so, the `search` starts at the substring of `str` specified by the regular expression's `lastIndex` property. For example, assume you have this script:

```
myRe=/ab*/g;
str = "abbcdefabh"
myArray = myRe.exec(str);
Console.Write("\r\nFound " + myArray[0] +
". Next match starts at " + myRe.lastIndex)
mySecondArray = myRe.exec(str);
Console.Write("\r\nFound " + mySecondArray[0] +
". Next match starts at " + myRe.lastIndex)
```

This script displays the following text:

```
Found abb. Next match starts at 3
Found ab. Next match starts at 9
```

Examples

In the following example, the user enters a name and the script executes a match against the input. It then cycles through the array to see if other names match the user's name.

This script assumes that first names of registered party attendees are preloaded into the array A, perhaps by gathering them from a party database.

```
A = ["Frank", "Emily", "Jane", "Harry", "Nick", "Beth", "Rick",
            "Terrence", "Carol", "Ann", "Terry", "Frank", "Alice", "Rick",
            "Bill", "Tom", "Fiona", "Jane", "William", "Joan", "Beth"]
function lookup() {
      firstName = /\w+/i();
      if (!firstName)
                  Alert (RegExp.input + " isn't a name!");
      else {
            count = 0;
            for (i=0; i<A.length; i++)
```

```
                  if (firstName[0].toLowerCase() == A[i].toLowerCase()) count++;
            if (count ==1)
                  midstring = " other has ";
            else
                  midstring = " others have ";
            window.alert ("Thanks, " + count + midstring + "the same name!")
      }
}
Enter your first name and then press Enter.
```

test

Executes the search for a match between a regular expression and a specified string. Returns `true` or `false`.

Syntax

```
regexp.test(str)
```

Parameters

`regexp`

The name of the regular expression. It can be a variable name or a literal.

`str`

(Optional) The string against which to match the regular expression. If omitted, the value of RegExp.input is used.

Description

When you want to know whether a pattern is found in a string use the `test` method (similar to the `String.search` method); for more information (but slower execution) use the `exec` method (similar to the `String.match` method).

Example

The following example prints a message which depends on the success of the test:

```
function testinput(re, str){
      if (re.test(str))
            midstring = " contains ";
      else
            midstring = " does not contain ";
      Console.Write (str + midstring + re.source);
}
```

Part II

Scripting Reference

In Scripting Reference:

9 Dashboard Scripting

In This Chapter

About Scripts

When you use Interactive Reporting to create a dashboard, you can include one or more Interactive Reporting documents and one or more of script components:

- Startup and Shutdown Scripts—Scripts that run when documents are opened and closed (Save and Save As in Interactive Reporting Studio, Interactive Reporting Web Client, and EPM Workspace do not execute document shutdown scripts when saving documents. These scripts only execute when documents are closed. To prevent a startup script from running, press and hold Ctrl while opening the document)
- Dashboard Shapes and Controls—User-interface components that enable interaction with the application
- Computed Columns—Scripts that run within the context of a results or table section column
- Custom Menu Items—Items that enable scripts to run from any section

On Windows, you can launch script commands from the command line. The commands must include the `-jscriptcmd` flag. For example, to launch Interactive Reporting, you type

```
brioqry.exe -jscriptcmd "Application.Documents.Open ("c:\\temp\\hyperiondoc.bqy")"
```

Understanding Functions

Functions are a fundamental building block of JavaScript. A function is a JavaScript procedure: a set of statements that performs a specific task. To use a function, you must define it before your script can call it.

Defining Functions

A function definition consists of the function keyword, followed by:

- The function name
- A list of arguments, enclosed in parentheses and separated by commas
- JavaScript statements that define the function, enclosed in braces { }

For example, to define a simple function named `square`, you type

```
function square(number) {
  return number * number;
}
```

The function name is `square`; the argument is `number`. The statement is `return number * number`. The function returns the value of the argument multiplied by itself.

Changing Function Parameters and Object Properties

Parameters are passed to functions by a value. If a function changes the value of a parameter, the change is not reflected globally or in the calling function. If you pass an object as a parameter to a function and the function changes the properties of the object, the change is visible outside the function.

Example:

```
function myFunc(theObject) {
  theObject.make="Toyota"
}
mycar = {make:"Honda", model:"Accord", year:2004}
x=mycar.make // returns Honda
muffin (mycar) // pass object mycar to the function
y=mycar.make // returns Toyota (property was changed by the function)
```

Calling Functions

In Interactive Reporting, all functions defined in the current script or defined globally or at a higher level than the current script can be called.

Note: Use caution when working with global variables. Global variables are visible throughout Interactive Reporting, including to computed column calculations and report section expressions.

Defining a function names the function and specifies what happens when the function is called. Calling the function performs the specified actions with the indicated parameters. For example, to call the function `square` (see "Defining Functions" on page 242) with an argument of 5, you call `square(5)`. The function executes its statements and returns the value 25.

Function arguments can be strings, numbers, and objects. Functions can be recursive; that is, they can call themselves.

Example—Recursive function that computes factorials:

```
function factoring) {
  if ((n == 0) || (n == 1))
    return 1
  else {
    result = (n * factorial(n-1) )
  return result
  }
}
```

You can compute the factorials of 1 through 5 as follows:

```
a=factorial(1) // returns 1
b=factorial(2) // returns 2
c=factorial(3) // returns 6
d=factorial(4) // returns 24
e=factorial(5) // returns 120
```

Function Scope

Functions are accessible within the scope in which they are created, unless they are explicitly defined in a different scope. Thus, a function which is defined in the `OnClick()` event handler of a command button can only be called by other statements in that event handler.

Example—Two command buttons in a dashboard section:

```
// MyButton
function square(value)
{
return value*value;
}
Alert (‘’The square of 3 equals ‘’+ square(3))

// YourButton
var retVal = square(3)
// generates a runtime error
Alert (‘’The square of 3 equals ‘’+ retVal)
```

The `square` function is only visible in the context of MyButton. As a result, a call to the `square` function from YourButton generates a runtime error.

Defining Functions in Different Scopes

To make functions visible to scripts throughout an application, you must explicitly define the scope in which the function is visible. This objective can be accomplished in various ways:

- Using `with` statements
- Dynamically adding methods to objects
- Assigning functions to global variables

Examples: Defining Functions in Different Scopes

When using a with statement to set the script scope, functions defined within the with statement become visible for that object.

Example—Using a with statement:

```
// MyButton
With (YourButton)
{
  function square(value)
  {
    return value*value;
  }
  Alert ("The square of 3 equals "+ square(3))
}

// YourButton
var retVal = square(3)
Alert ("The square of 3 equals "+ retVal)
```

Explicitly defining the square function within the context of the YourButton object makes the function visible to scripts that are running behind the button. Any object from the object model can be used with the with statement.

Example—Dynamically adding a method to an object:

```
// MyButton
Function square(value)
  {
    return value*value;
  }
Alert ("The square of 3 equals "+ square(3))YourButton.square = square;

// YourButton
var retVal = square(3)
Alert ("The square of 3 equals "+ retVal)
```

A new method is added dynamically to the YourButton object. Scripts running in the context of this object can access the dynamically created square function.

Example—Assigning a value to a global variable:

```
// MyButton
Function square(value)
  {
    return value*value;
  }
Alert ("The square of 3 equals "+ square(3)) MyGlobalFunction = square;

// YourButton
var retVal = MyGlobalFunction(3)
Alert ("The square of 3 equals "+ retVal)
```

Creating a variable named *MyGlobalFunction* without using the var statement places the variable in the top scope. Thus, the variable is global.

Using Variables

Variables used to create function components:

- Using Global Variables
- Using Document Variables
- Using Section Variables

Using Global Variables

Functions declared at the top level of scripts are global; that is, they are always in memory, and all other functions can read and modify them.

You must carefully consider how and where to use global variables because a change to one component of a variable affects all variables that reference the changed variable.

➤ To call a function using a global variable, type

```
glMyFunction(myParam)
```

Using Document Variables

Document variables belong to and depend upon the document script in which they are included. That is, document variables are erased when their documents are closed or another variable is defined for them. From one document, you can call functions in two ways: `myFunction(myParam)` or `ActiveDocument.myFunction(myParam)`.

Using Section Variables

Section variables exist in the section and document in which they are included. You use section variables to specify elements defined from multiple dashboard sections (provided that each section declares a function of the same name).

You can call the same-named function by typing

```
ActiveDocument.Sections["Dashboard"].myFunction(myParam)
```

`(myParam)` can be defined in two ways:

- In `this.Parent.myFunction=myFunction`, `this.Parent` is the parent object of `this`, which can be a Fields or Shapes object and is the object to which the script belongs.
- In `this.myFunction=myFunction`, the object belongs to the `OnActivate` or `OnDeactivate` method of the section and `this` is the section object.

Using JavaScript Statements

JavaScript uses conditional and loop statements to change the order in which scripts execute (based on object states or user selections) and break statements to change the execution of control structures.

- Conditional Statements
- Loop Statements
- break Statements
- for...in Statements
- with Statements

Conditional Statements

Conditional statements are sets of commands that execute if specified conditions are true. JavaScript supports three conditional statements:

- if...else Statements
- Inline if Statements
- switch Statements

if...else Statements

If a logical condition is true, the `if` statement performs one or more actions. If a logical condition is false, the `else` clause, which is optional, performs one or more actions.

A typical `if` statement:

```
if (condition) {
  statements1
}
else {
  statements2
}
```

Conditions are JavaScript expressions that evaluate to true or false. Statements to be executed are JavaScript statements, including nested `if` statements. If you want to use multiple statements after an `if` or `else` statement, enclose the statements in braces {}.

Do not confuse primitive-Boolean true and false values with Boolean-object true and false values. Objects whose values are not undefined or null, including Boolean objects whose values are false, evaluate to true when passed to conditional statements, for example:

```
var b = new Boolean(false);
if (b) // this condition evaluates to true
```

Note: If you use an initial capital letter for *if* or *else* or type the word *then,* you receive an error message. A `then` statement is implied for values enclosed in braces { }.

You can use an `if...else` statement to stop a script when the script encounters a selected condition, for example:

```
if(cellvalue==0){ Alert("Cell has no value") }
else{ (execute remainder of code here) }
```

Inline if Statements

Inline `if` statements are alternatives to `if...else` statements. They use the conditional operator (`?`) to represent `if` statements and a colon (:) to represent`else` clauses. Such statements require three operands:

```
condition ? expr1 : expr2
```

- `condition`—An expression that evaluates to true or false
- `expr1, expr2`—Expressions with values of any type

If `condition` is true, the value of `expr1` is returned; if `condition` is false, the value of `expr2` is returned.

You should place the condition in parentheses and each expression in single or double quotes:

```
((condition == value)?'expr1':'expr2')
```

Note: You can safely eliminate the condition parentheses, but omitting the quotes can cause problems.

Numbers do not require quotes.

```
(condition?2:10)
```

For example, to display one message if a variable is true and another message if the variable is false, you can use this statement:

```
( isMember ? 'Member' : 'Not a member')
```

In this case, if `isMember` is true, `Member` is returned. If `isMember` is false, `Not a Member` is returned.

You can also use the comparison operator:

```
((isMember == 'Yes' ) ? 'Member' : 'Not a member')
```

In this case, if `isMember` equals the string `Yes,Member` is returned. If `isMember` does not equal the string `Yes`, `Not a Member` is returned.

If you want to nest inline `if` statements, (that is, use one inline `if` statement as an expression for another inline `if` statement), enclose the nested inline `if` statements in parentheses:

```
(1 != 1 ? 'Not Equal' : (1 < 1 ? 'Less Than': 'Equal') )
```

In this case, if 1 does not equal 1, the second inline `if` statement is evaluated as part of the `else` clause of the first inline `if` statement. If 1 evaluates as less than 1, the string `Less Than` is returned. Because 1 equals 1, the string Equal is returned from the else clause of the second inline if statement.

Note: If you open a Release 5.5 document in Interactive Reporting Release 6.x and the document contains computed columns with nested if...else statements, the Interactive Reporting JavaScript engine converts the if...else syntax to the inline if statement syntax. The conversion process does not change the meaning or value of the original if...else statement.

switch Statements

switch statements evaluate expressions and match the values of the expressions to case labels. If a match is found, the associated statement executes.

Example—switch statement:

```
switch (expression){
  case label :
    statement;
    break;
  case label :
    statement;
    break;
  ...
  default : statement;
}
```

The program searches for a label that matches the value of the expression and then executes the associated statement. If no match is found, the program searches for the optional default statement. If a match is found, the program executes the associated statement. If no default statement is found, the program executes at the statement following the end of switch.

Break statements, which are optional, are associated with case labels. Break statements ensure that, after matched statements execute, programs leave switch and continue execution at the statement following switch. If break is omitted, the program continues execution at the next statement of the switch statement.

In the following example, if expr evaluates to Bananas, the program matches the value with case Bananas and executes the associated statement. When break is encountered, the program terminates switch and executes the statement following is.

```
switch (expr) {
  case "Oranges" :
    Console.Writeln("Oranges are $0.59 a pound.");
    break;
  case "Apples" :
    Console.Writeln("Apples are $0.32 a pound.");
    break;
  case "Bananas" :
    Console.Writeln("Bananas are $0.48 a pound.");
    break;
  case "Cherries" :
    Console.Writeln("Cherries are $3.00 a pound.");
    break;
  default :
    Console.Writeln("Sorry, we are out of " + i + ".");
}
Console.Writeln("Is there anything else you'd like?");
```

Loop Statements

Loop statements are sets of commands that execute repeatedly, until a specified condition is met. JavaScript supports five loop statements:

- for Statements
- do...while Statements
- while Statements
- label Statements
- continue Statements

Note: `label`, although not a loop statement, is frequently used with loop statements. `break` and `continue` statements are also used within loop statements.

Note: `for...in` statements, which also execute statements repeatedly, are used for object manipulation. See "Manipulating Objects with JavaScript" on page 253.

for Statements

`for` loops repeat until a specified condition evaluates to false. The JavaScript `for` loop is similar to the Java and C `for` loop.

`for` loop syntax:

```
for ([initialExpression]; [condition];
[incrementExpression]) {
   statements
}
```

When a *for* loop executes, the following occurs:

1. The initializing expression `initialExpression`, if any, is executed. This expression usually initializes one or more loop counters, but the syntax accepts expressions of any degree of complexity.
2. The condition expression is evaluated. If the value of condition is true, the loop statements execute. If the value of condition is false, the `for` loop terminates.
3. The statements execute.
4. The update expression `incrementExpression` executes and control returns to step 2.

do...while Statements

do...while statements repeat until a specified condition evaluates to false.

```
do {
  statement
} while (condition)
```

The statement executes once before the condition is evaluated. If the condition returns true, the statement executes again. At the end of every execution, the condition is evaluated. When the condition returns false, execution of the statement stops, and the statement following do...while executes.

Example—do...while statement that iterates at least once and reiterates until it is greater than five:

```
do {
  i+=1;
  Console.Writeln(i);
} while (i<5);
```

while Statements

while statements execute as long as a specified condition evaluates to true:

while statement syntax:

```
while (condition) {
  statements
}
```

If the condition becomes false, the statements within the loop stop executing and control passes to the statement following the loop.

The condition test occurs before the loop statements are executed. If the condition returns true, the loop statements are executed, and the condition is re-tested. If the condition returns false, loop execution stops, and the statement following while executes.

Example—while loop that repeats as long as n < 3:

```
n = 0
x = 0
while( n < 3 ) {
  n ++
  x += n
}
```

With each iteration, the loop increments *n* and adds that value to *x*. Therefore, *x* and *n* obtain the following values:

- After the first pass: n = 1, and x = 1
- After the second pass: n = 2, and x = 3
- After the third pass: n = 3, and x = 6

After the third pass, the condition `n < 3` is no longer true, so the loop terminates.

Example—`while` loop that never terminates; that is, the loop executes forever because the condition never becomes false:

```
while (true) {
  Alert("Hello, world") }
```

label Statements

`label` statements refer to statements in other locations. For example, you can use `label` to identify a loop and then use `break` to interrupt execution of the loop.

`label` statement syntax:

```
label :
  statement
```

`label` can be any JavaScript identifier that is not a reserved word, and the statement identified by `label` can be any type.

Example—`markLoop` (the `label` identifier) identifies a `while` loop:

```
markLoop:
while (theMark == true){
  doSomething();
}
```

continue Statements

`continue` statements can be used to restart `while`, `do...while`, `for`, and `label` statements.

- For`while` and `for`, `continue` terminates the current loop and continues execution of the loop with the next iteration
- For `while`, `continue` returns to the condition
- For `for, continue` moves to the increment expression
- For `label, continue` restarts the statement or continues execution of a labeled loop with the next iteration. The `continue` statement must be in a looping statement that is identified by the label that `continue` uses.

Note: In contrast to the `break` statement, `continue` does not entirely terminate the execution of the loop.

`continue` statement syntax: `continue` or `continue [label]`

Example—A `while` loop with a `continue` statement that executes when the value of `i` is 3. Thus, `n` obtains the values 1, 3, 7, and 12.

```
i = 0
n = 0
while (i < 5) {
  i++
  if (i == 3)
    continue
  n += I
}
```

Example—A statement labeled *checkiandj* that contains a statement labeled *checkj:*

```
checkiandj :
  while (i<4) {
    Console.Writeln(i + "");
    i+=1;
    checkj :
      while (j>4) {
        Console.Writeln(j + "");
        j-=1;
        if ((j%2)==0);
          continue checkj;
        Console.Writeln(j + " is odd.");
      }
    Console.Writeln("i = " + i + "");
    Console.Writeln("j = " + j + "");
   }
```

If `continue` is encountered, the current iteration of *checkj* terminates, and the next iteration of *checkj* begins. Iterations continue until the *checkj* condition returns false. When false is returned, the remainder of *checkiandj* is complete and *checkiandj* reiterates until its condition returns false. When false is returned, the statement following *checkiandj* executes. If `continue` had a label of *checkiandj, checkiandj* would re-execute.

break Statements

`break` statements terminate `loop`, `switch`, or `label` statements.

When using `break` with a `while`, `do...while`, `for`, or `switch` statement, `break` terminates the innermost enclosing `loop` or `switch` immediately and transfers control to the following statement.

When using `break` within an enclosing `label` statement, it terminates the statement and transfers control to the following statement. If a `label` is specified when the `break` is issued, the `break` statement terminates the specified statement.

`break` statement syntax:`break` or `break [label]`

Example—Iteration through array elements until the index of an element with the value of `theValue` is found:

```
for (i = 0; i < a.length; i++) {
  if (a[i] = theValue);
    break;
}
```

Manipulating Objects with JavaScript

JavaScript uses `for...in` and `with` statements to manipulate objects.

- for...in Statements
- with Statements

for...in Statements

`for...in` statements iterate specific variables over all properties of an object. For each property, JavaScript executes the specified statements.

`for...in` statements syntax:

```
for (variable in object) {
  statements }
```

Example—An object and its name used as the argument, iteration over all properties of the object, and return of a string that lists property names and values:

```
function dump_props(obj, obj_name) {
   var result = ""
   for (var i in obj) {
      result += obj_name + "." + i + " = " + obj[i] + ""
   }
   result += "<HR>"
   return result
}
```

Example—Car (as the object) with properties make and model:

```
car.make = Ford
car.model = Mustang
```

with Statements

`with` statements establish default objects for sets of statements. JavaScript reviews unqualified names to determine whether the names are properties of the default object. If an unqualified name matches a property, the property is used in the statement; otherwise, a local or global variable is used.

`with` statement syntax:

```
with (object){
  statements
}
```

Example—Math is specified as the default object; the PI property and the cos and sin methods are referenced, but no object is specified for the references; therefore, Math is assumed to be the object of the references:

```
var a, x, y
var r=10
with (Math) {
  a = PI * r * r
  x = r * cos(PI)
  y = r * sin(PI/2)
}
```

Microsoft Automation Interfaces and the Object Model

Typically, you use JavaScript to manipulate the object model from within a dashboard section to build self-contained analytical applications. Because Interactive Reporting Studio is an OLE Automation server, on Windows systems, you can use Microsoft Automation Interfaces to work with the object model and with Interactive Reporting Studio in external applications such as Excel, Visual Basic, C++, or any application that can make OLE Automation calls. The object model is exposed through the `brioqry.tlb` file located in the `system32` directory.

OLE Automation Controller within JavaScript (JOOLE)

Interactive Reporting Studio is an OLE Automation controller. On Windows systems, Interactive Reporting Studio can control external applications (that is, programmable ActiveX objects) that are OLE Automation servers. By making OLE Automation calls, Interactive Reporting Studio can access functionality exposed by other OLE Automation Servers. Examples of OLE Automation Servers, such as Excel and Visual Basic.

➤ To define a JOOLE object reference, use `var <variableName> = new JOOLEObject(<ProgId>)`.

`<ProgID>` is a string that indicates the object to be referenced. Interactive Reporting Studio or Interactive Reporting Web Client passes the string as a reference to the object; for example, `Excel.Application`. `<ProgID>` is stored in the registry and contains a string defined as `Project.ClassName`.

JOOLE works on Windows systems only.

Note: It is recommended that JOOLE calls stored in plug-in scripts be implemented on Internet Explorer.

Tip: You cannot embed OLE objects inside Interactive Reporting documents. Likewise, Interactive Reporting Studio and Interactive Reporting Web Client are *not* OLE servers that produce OLE objects that you can embed in OLE containers.

Example—Invoking an Excel worksheet from a command button created in a dashboard section and writing “Hello World” to rows of a column.

```
Excel = new JOOLEObject("Excel.Application");
Excel.Visible = true;
Excel.Workbooks.Add;
Excel.Sheets.Item(1).Cells.Item(2).Item(2).Value = "Hello";
Excel.Sheets.Item(1).Cells.Item(2).Item(3).Value = "World";
Print(Excel.Sheets.Item(1).Cells.Item(2).Item(2).Value);
```

Example—Invoking Outlook from a command button created in a dashboard section and writing a message in the body of the e-mail message:

```
var olApp = new JOOLEObject("Outlook.Application")
var olNote = olApp.CreateItem(0)
olNote.To = "yourname@Hyperion.com"
olNote.Subject = "JOOLEObject mail Example"
olNote.Body = "This is an automatically generated note."
//olNote.Attachments.Add (filepath)
olNote.Send
```

Example—Using JOOLE to start Outlook on Windows XP:

```
var obj = Application.Shell("c:\\program Files\\outlook express\\msimn.exe")
```

Example—Invoking, displaying, and printing a Word document (Hello.doc) from a command button created in a dashboard section:

```
/Create Word Object
word = new JOOLEObject("Word.Application");
// Make is Visible
word.Visible = true;
//Open the desired file
word.Documents.Open("c:\\Hyperion\\Hello.doc");
// Set Options
word.Options.PrintBackground = false;
//Start Printing
word.ActiveDocument.PrintOut();
```

Example—Creating and writing text to a text file from an Interactive Reporting document file, appending the characters “_trace” to the document name and replacing the document extension with `.TXT`:

```
var oleApp = new JOOLEObject("Scripting.FileSystemObject")
var myPath=ActiveDocument.Path.slice(0,-4)+"_trace.txt"
var traceDoc=oleApp.CreateTextFile(myPath)
traceDoc.WriteLine("hello from Hyperion")
traceDoc.Close()
```

For example if the Interactive Reporting document is named *DashboardText.bqy*, Interactive Reporting Studio or Interactive Reporting Web Client creates the text file named *DashboardText_trace.txt*.

Exporting Scripts to Text Files

You use the Export Scripts To Text File feature to export JavaScript scripts from Interactive Reporting documents to text files. Interactive Reporting Studio categorizes text files by object name and events and includes document and custom menu-item scripts.

➤ To export scripts to text files:

1 Select **File**, then **Export**, and then **Script To Text File**, .

  Export Script is displayed.

2 Specify the file name and location, and click **Save.**

10

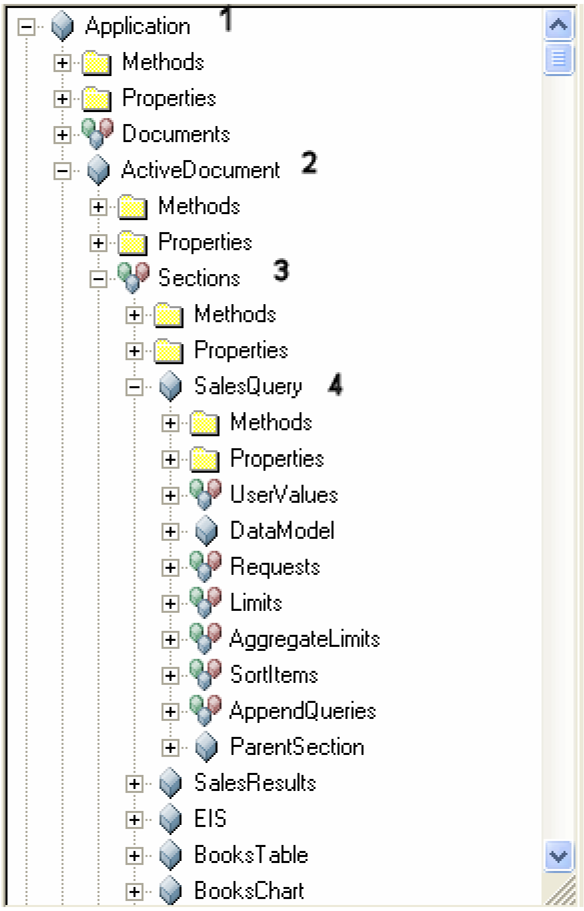
Object Model Map

In This Chapter

Object Model Hierarchy

The object model map provides an expanded view of the object model hierarchy, as seen in Script Editor. It begins at the highest level, Application, and drills down through the hierarchy. The object model hierarchy:

- Application Level Hierarchy
- ActiveDocument Level Hierarchy
- Sections



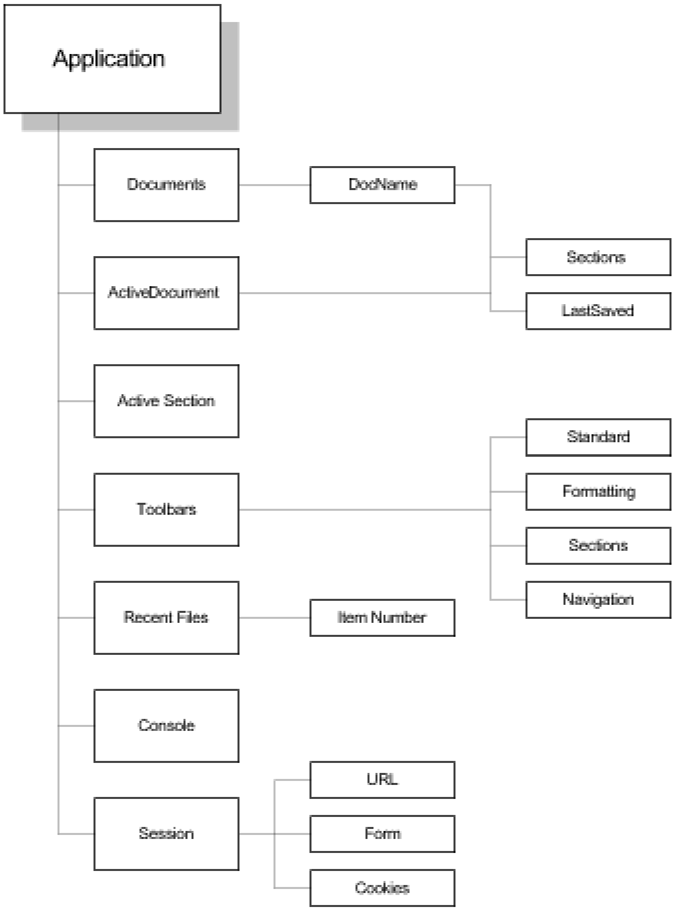
```
Application
    Methods
    Properties
    Documents
    ActiveDocument
        Methods
        Properties
        Sections
            Methods
            Properties
            SalesQuery
                Methods
                Properties
                UserValues
                DataModel
                Requests
                Limits
                AggregateLimits
                SortItems
                AppendQueries
                ParentSection
            SalesResults
            EIS
            BooksTable
            BooksChart
```

Table 33 Object Model Hierarchy

Number	Hierarchy
1	Application
2	ActiveDocument
3	Sections
4	Section specific

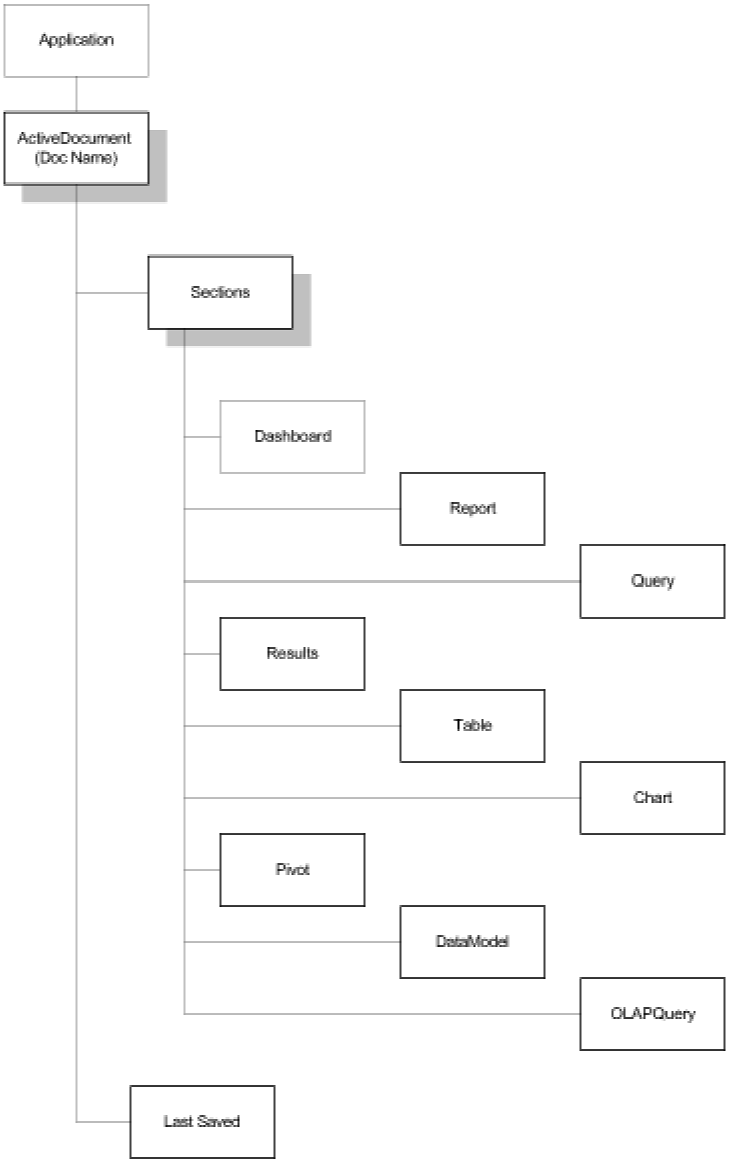
Application Level Hierarchy

Objects subordinate to the Application level:



ActiveDocument Level Hierarchy

Objects subordinate to the ActiveDocument level:



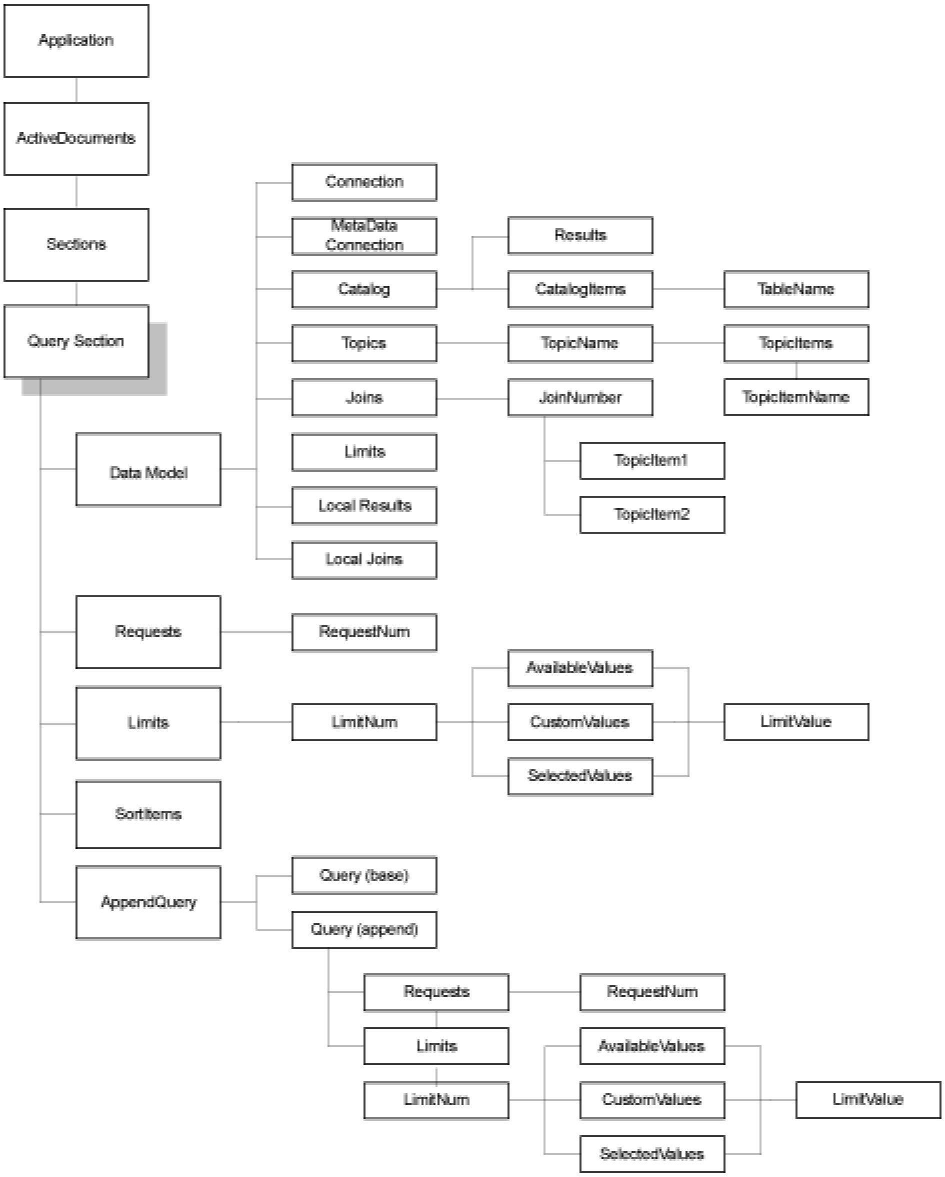
Sections

Objects subordinate to Sections:

- Query Section
- Dashboard Section
- Chart Section
- Results, Report, and Pivot Sections
- Table and OLAPQuery Sections

Query Section

Objects subordinate to Query Section:



Dashboard Section

Objects subordinate to Dashboard Section:

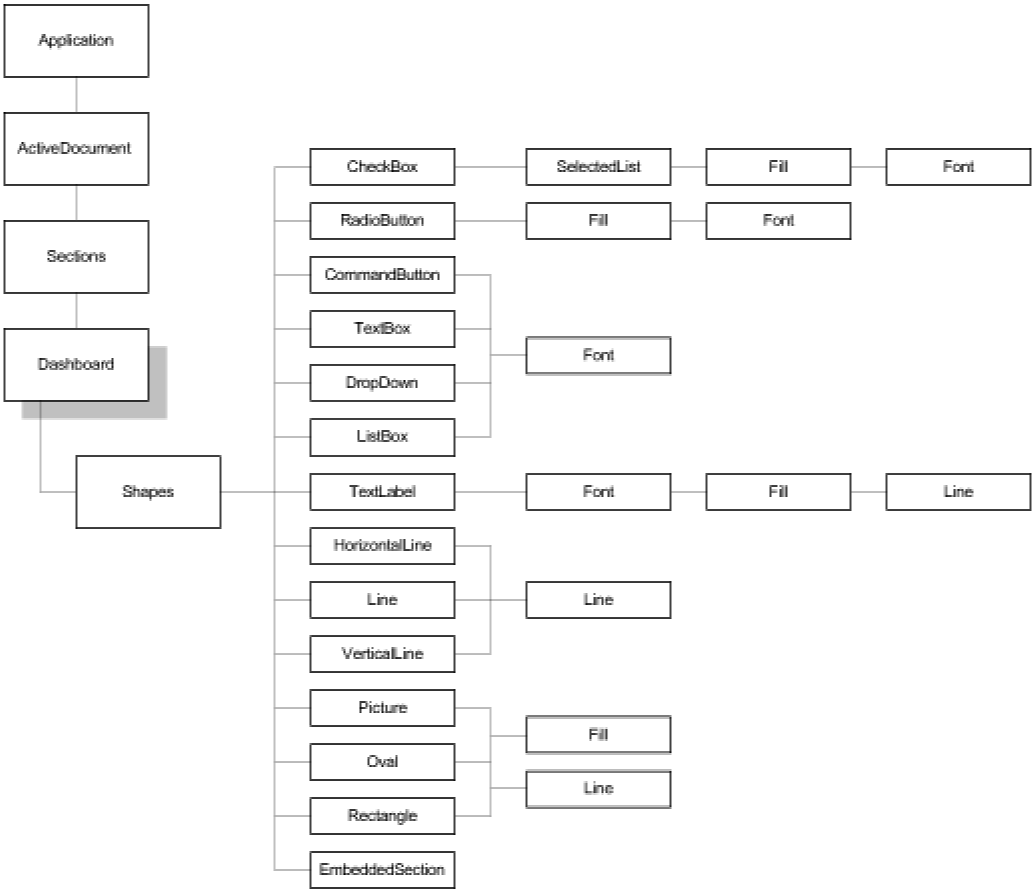
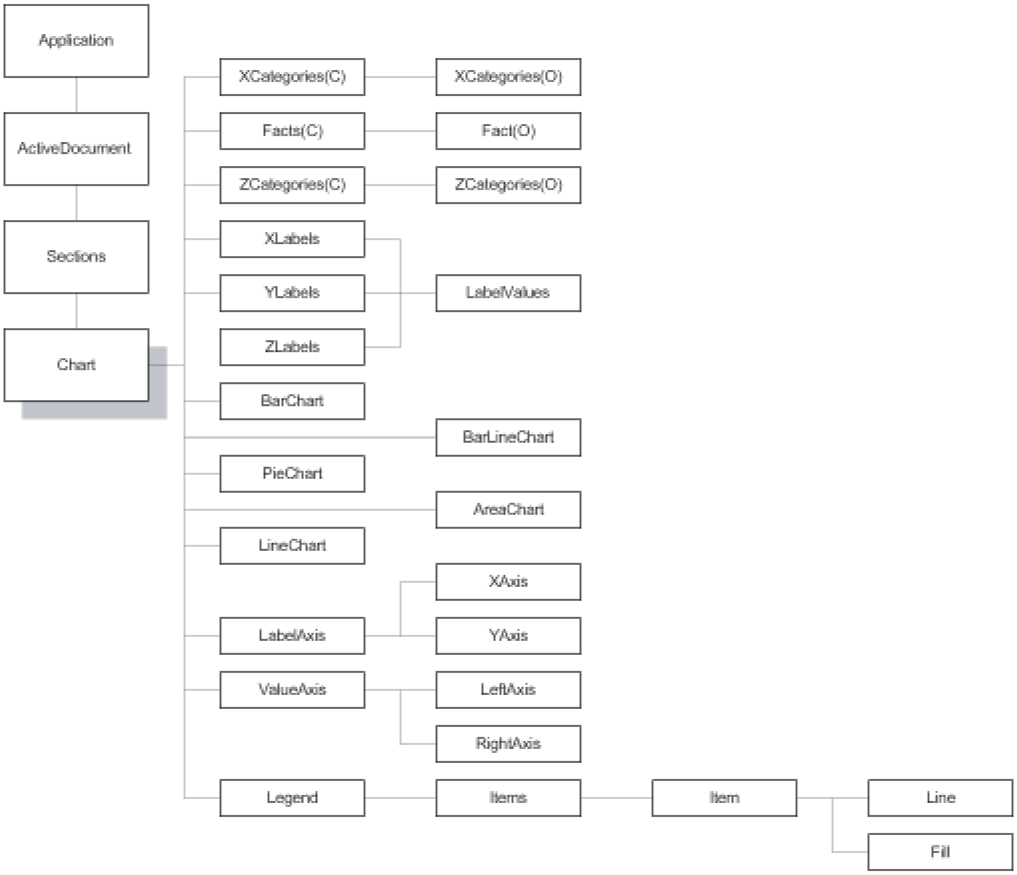


Chart Section

Objects subordinate to Chart Section:



Results, Report, and Pivot Sections

Objects subordinate to Results, Reports, and Pivot sections:

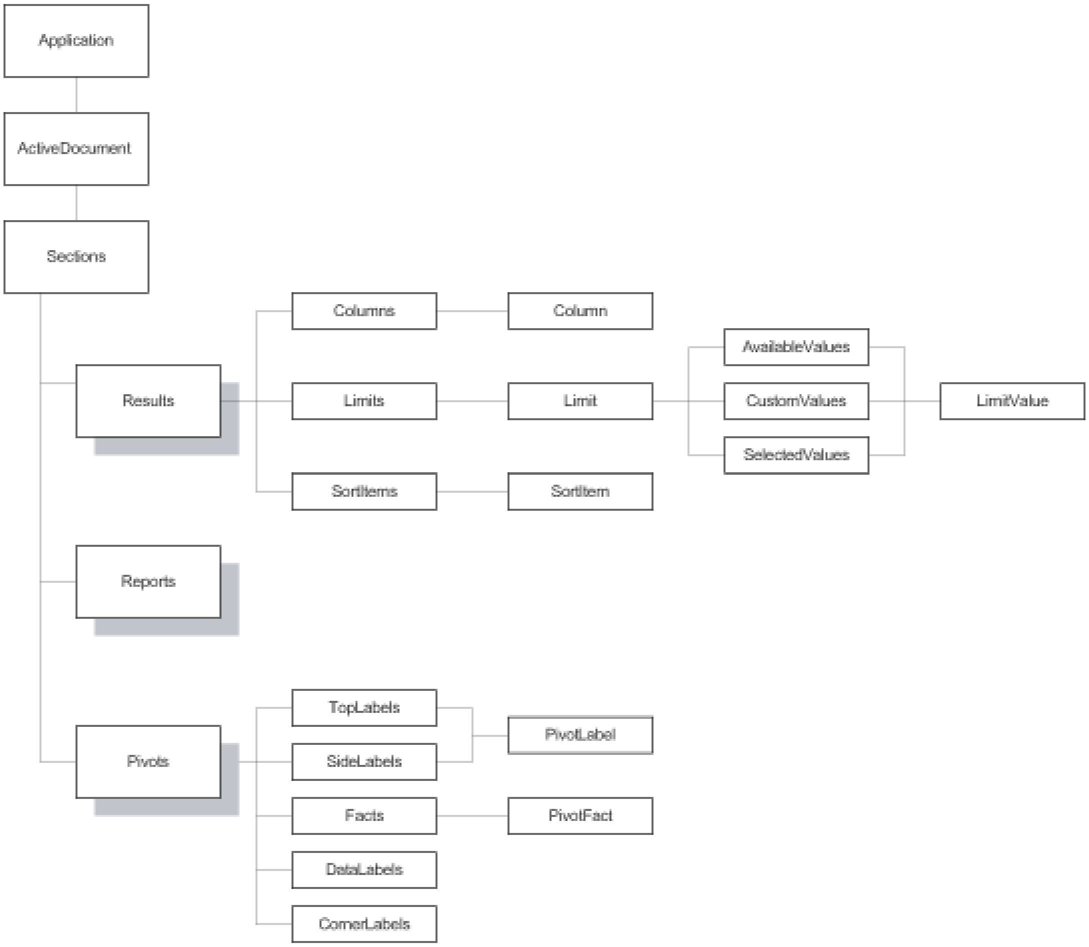
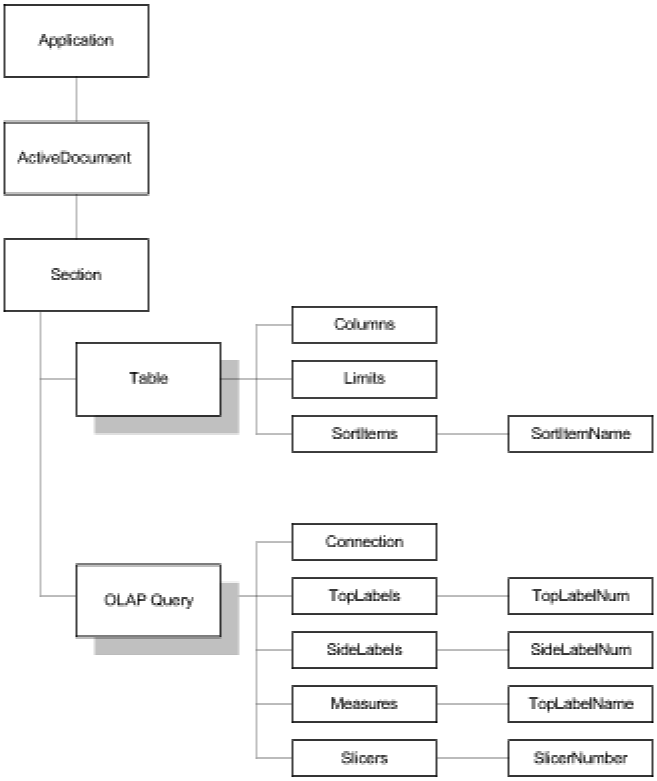


Table and OLAPQuery Sections

Objects subordinate to Table and OLAPQuery Sections:


Application
ActiveDocument
Section
Table
Columns
Limits
SortItems
SortItemName
OLAP Query
Connection
TopLabels
TopLabelNum
SideLabels
SideLabelNum
Measures
TopLabelName
Slicers
SlicerNumber

11

JavaScript Examples

In This Chapter

Displaying and Entering Values in Text Boxes

You use Interactive Reporting Studio text boxes to display output to and gather input from the application.

Ways in which, in Run mode, you can use text boxes:

- Entering values
- Displaying values

- Displaying read-only information
- Validating data
- Calculating data

Text boxes are associated with three events: OnEnter, OnChange, and OnExit.

Example—For OnEnter, attach a JavaScript script

```
/* OnEnter Event—enables CommandButton */
var sect_name='Dashboard';
var ctrl_name='CommandButton1';
ActiveDocument.Sections[sect_name].Shapes[ctrl_name].Enabled = true;
```

Example—For OnChange, attach a JavaScript script

```
/* OnChange Event- validates changes*/
var sect_name='Dashboard';
var ctrl_name='TextBox1';
if (ActiveDocument.Sections[sect_name].Shapes[ctrl_name].Text=='Hello')
{
Alert('Hello is an Invalid Entry');
}
```

Example—For OnExit, attach a Javascript script

```
/* OnExit Event- increments variable counter */
var sect_name='Dashboard';
var ctrl_name='TextBox1';
if (ActiveDocument.Sections[sect_name].Shapes[ctrl_name].Text=='2')
{
x=x+1;
}
```

Retrieving and Setting Object Properties

Interactive Reporting objects have associated properties, which represent attributes such as name, visible, enabled, and text. Most properties can be set using the Properties dialog box in the dashboard section.

Example—Get a ListBox property

```
/* Get the value of the ListBox MultiSelect property*/
var sect_name='Dashboard';
var ctrl_name='ListBox1';
TextBox1.Text =
ActiveDocument.Sections[sect_name].Shapes[ctrl_name].MultiSelect;
```

Example—Set a CheckBox property

```
/* Set the value of the CheckBox Checked property */
var sect_name='Dashboard';
var ctrl_name='CheckBox1';
ActiveDocument.Sections[sect_name].Shapes[ctrl_name].Checked = true;
```

Example—Get a RadioButton group property

```
/* Get the value of the RadioButton Group property */
var sect_name='Dashboard';
var ctrl_name='CheckBox1';
TextBox1.Text =
ActiveDocument.Sections[sect_name].Shapes[ctrl_name].Group;
```

Object Model Placement and Sizing

All shapes and controls on a dashboard have a node called Placement, which contains properties and a method to enable the shape or control to be moved or resized.

The Shapes collection contains two methods, CreateShape and RemoveShape, to enable shapes to be created and deleted using COM or JavaScript.

Placement Node

The Placement node has four properties and one method that enable the parent shape or control to be moved or resized.

All values given to and supplied from the Placement properties and methods are pixel values.

Placement Properties

These are the Placement properties.

XOffset—Sets or returns the position of the left-hand edge of a shape

YOffset—Sets or returns the position of the top of a shape

Width—Sets or returns the width of a shape

Height—Sets or returns the height of a shape

Placement Method

The Placement method, Modify(XOffset, YOffset, Width, Height), changes the origin and size of the shape.

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

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

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

Creating and Deleting Shapes

The Shapes collection includes two methods (CreateShape and RemoveShape) that enable shapes and controls to be created and deleted using COM or JavaScript.

CreateShape Function

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

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

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

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

A reference to the newly created shape is returned from this function. You can use the reference to set the Placement of the shape using the Placement properties or functions, see "Placement Node" on page 269, and you can set the name of the shape using the Name property.

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

RemoveShape Function

RemoveShape(shapeName) can be used to delete a shape from a dashboard using COM or JavaScript. Pass the name of the shape to be deleted as the shapeName parameter.

Caution! The RemoveShape operation cannot be undone.

Using the Placement Properties and Method

In the procedural example, an object; for example, a graphic of a ball, is moved across a screen by using the Placement properties and method.

Note: Modify the example to suit your requirements.

Placing Objects

Example—Using the properties and method of the Placement node.

➤ To move an object (graphic) across a dashboard frame:

1. In Interactive Reporting Studio, select **File**, then **New**

   New File is displayed.

2. Select **Other**, then **A Blank Document**, and then click **OK.**

   The document opens in Design mode.

3. From **Elements**, expand **Graphics**, drag the rectangle onto the dashboard and resize it to be bigger.

4. Right-click the rectangle, and select **Properties.**

   Properties is displayed.

5. In **Properties**, enter a **Name** for the rectangle, and click **OK.**

   For example, type Field. The rectangular shape is the area that the ball graphic moves across.

6. **Optional:** Select **Border and Background**, and change the background color.

7. From **Graphics**, drag a picture onto the dashboard, and then navigate to and select an image of a ball.

8. Right-click the ball graphic, and select **Properties.**

9. In **Properties**, enter a **Name** for the graphic, and click **OK.**

   For example, type Ball.

10. From **Elements**, expand **Controls**, and drag an option button onto the dashboard.

    With the option button selected, drag a copy to the right to create a duplicate.

11. Right-click the first option button, and select **Properties.**

12. In **Properties,** enter a **Name**, **Title**, and **Group Name**, and then click **OK.**

    For example, type Left as the Name, Move Left 100 as the Title, and LeftRight as the Group Name.

13. Repeat step 12 for the second option button with alternate data.

    For example, type Right as the Name, Move Right 100 as the Title, and LeftRight as the Group Name.

14. From **Controls**, drag a command button onto the dashboard, right-click, and select **Properties.**

15. In **Properties**, enter a **Title**, and click **OK.**

    For example, type Move Ball.

16. Right-click Move Ball again, and select **Scripts.**

    Script Editor is displayed.

17. Copy and paste this script into the editor. You can modify the script to suit your requirements.

```
var moveHoriz
if (Right.Checked){
   moveHoriz = 100
}else{
  moveHoriz = (-100)
}
var left = Ball.Placement.XOffset
var top = Ball.Placement.YOffset
var width = Ball.Placement.Width
var height = Ball.Placement.Height

if (moveHoriz > 0){
   if ( left+width+moveHoriz > Field.Placement.XOffset + Field.Placement.Width){
      Alert ("Reached the right end of the field","Goal !!! Yay!!!")
      Ball.Placement.XOffset = Field.Placement.XOffset + Field.Placement.Width - 
Ball.Placement.Width
      Left.Checked = true
      return
   }
}else{
   if ( left+moveHoriz < Field.Placement.XOffset){
      Alert ("Reached the left end of the field","Goal !!! Yay!!!")
      Ball.Placement.XOffset = Field.Placement.XOffset
      Right.Checked = true
      return
   }
}
Ball.Placement.Modify(left + moveHoriz, top, width, height)
```

18 Click **OK** to close Script Editor.

19 Press **Ctrl+D** to exit Design mode, and save the dashboard.

Verifying Functionality

When the objects and the script are added to the dashboard, verify that the code is functioning correctly.

Test the functionality by clicking Move Ball. The ball moves 100 pixels at a time. When it reaches the edge of the rectangle, an alert is displayed.

Using CreateShape and RemoveShape

You use the CreateShape method to add a shape to a dashboard; for example, a goal for football, and you use the RemoveShape method to remove the shape of the goal. To perform the physical adding and removing of the shape, configure two command buttons.

Note: Modify the example to suit your requirements.

➤ To configure two command buttons:

1 In Interactive Reporting Studio, use the dashboard from the Placing Objects procedure.

2 Press **Ctrl+D** to enter Design mode.

3 From **Elements**, expand **Controls**, and drag two command buttons onto the dashboard.

4 Right-click CommandButton2, and select **Properties.**

5 In **Properties**, enter a **Name** and **Title.**

For example, type AddGoal as the Name, and Add a Goal as the Title.

6 **Repeat steps 4-5 for CommandButton3 with alternate data.**

For example, type DelGoal as the Name, and Delete the Goal as the Title.

7 **From Graphics, drag a rectangle onto the dashboard, right-click, and select Properties.**

The rectangle will be added or removed by the command buttons.

8 **In Properties, enter a Name.**

For example, type LeftGoal.

9 **Select Add a Goal, right-click, and select Scripts.**

10 **In Script Editor, copy and paste this code, or modify to suit your requirements.**

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

11 **Click OK to close Script Editor.**

12 **Right-click Delete the Goal, and select Scripts.**

13 **In Script Editor, copy and paste this code, or modify to suit your requirements.**

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

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

14 **Click OK to close Script Editor.**

15 **Press Ctrl+D to exit Design mode, and save the dashboard.**

Verify CreateShape and RemoveShape Functionality

Test the functionality by clicking Add a Goal. The goal shape is displayed. Click Delete the Goal, and the goal shape is removed.

Enabling and Disabling Controls

Dashboard graphics and controls have an Enable property that determines whether, in Run mode, they are enabled or disabled. Users can use enabled objects to trigger events. Users cannot use disabled objects (which are displayed as dimmed) to trigger events. The Enable property is available, for graphics and controls, from the Object tab in the Properties dialog box.

Example—Enable a control

```
/* Enables controls */
var sect_name='Dashboard';
var ctrl_name='TextBox1';
```

```
ActiveDocument.Sections[sect_name].Shapes[ctrl_name].Enabled = true;
```

Example—Disable a control

```
/* Disables controls */
var sect_name='DashboardDashboard';
var ctrl_name='TextBox1';
ActiveDocument.Sections[sect_name].Shapes[ctrl_name].Enabled = false;
```

Controlling the Visibility of Graphics and Controls

Dashboard graphics and controls have a Visible property that determines whether, in Run mode, they are displayed. Users can use only visible objects to trigger events. The Visible property is available, for graphics and controls, from the Object tab in the Properties dialog box.

Example—Make a control visible

```
/* Makes control Visible */
var sect_name='Dashboard';
var ctrl_name='TextBox1';
ActiveDocument.Sections[sect_name].Shapes[ctrl_name].Visible = true;
```

Example—Make a control invisible

```
/* Makes control Invisible */
var sect_name='Dashboard';
var ctrl_name='TextBox1';
ActiveDocument.Sections[sect_name].Shapes[ctrl_name].Visible = false;
```

Creating Interactive Reporting Database Connection Files (OCEs)

Example—Create an Interactive Reporting database connection file (OCE)

```
// try to create sample.oce from scratch.
// create SQLNet-Oracle8 oce - save as sample.oce
MyConnection = ActiveDocument.Sections["Query"].DataModel.Connection
MyConnection.Open("c:\\OCEs\\Sample.oce")
MyConnection.Username = "hyperion"
MyConnection.SetPassword("hyperion")
MyConnection.Connect()
MyConnection.SaveAs("c:\\temp\\sample.oce")

ActiveDocument.Sections["DataModel"].DataModel.Connection.Open("c:\\temp\\astro8.oce")
// need to connect ?
ActiveDocument.Sections["DataModel"].DataModel.Connection.UserName = "hyperion"
ActiveDocument.Sections["DataModel"].DataModel.Connection.SetPassword("hyperion")
ActiveDocument.Sections["DataModel"].DataModel.Connection.Connect()
```

Displaying a Connection Login Box

Example—Display a connection login box

```
ExecuteBScript(“set logon root, 'OCENAME', 'd:\\programfiles\\hyperion\\oces\\Astro
SQLNet Oracle8.oce': connect logon root”)
```

Downloading Data Models

Example—Download from the repository a data model, standard query, or standard query with report

```
//download a data model, standard query or standard query with reports //from a local
repository
//(document name to gain the download), (type of document), (repository //owner) (group
with access), (name of document)
ExecuteBScript("download doc root, 'SQR', 'ts', 'PUBLIC', 'Sales")
```

Displaying Table Catalogs

Example—Display a list of tables available on the database

```
// display table catalog
ActiveDocument.Sections["DataModel"].DataModel.Catalog.Refresh()
```

Adding Topics To Data Model Sections

Example—Add topics to a data model section

```
// add topics to DataModel section
CatItem =
ActiveDocument.Sections["DataModel"].DataModel.Catalog.CatalogItems["PCW_ITEMS"]
ActiveDocument.Sections["DataModel"].DataModel.Topics.Add(CatItem)
```

Setting Up Topic Object Variables

Example—Set up topic object variables

```
// setting up topic objects variables...
PCWItems = ActiveDocument.Sections["DataModel"].DataModel.Topics["PCW_ITEMS"]
PCWSales = ActiveDocument.Sections["DataModel"].DataModel.Topics["PCW_SALES"]
PCWCustomers = ActiveDocument.Sections["DataModel"].DataModel.Topics["PCW_CUSTOMERS"]
PCWPeriods = ActiveDocument.Sections["DataModel"].DataModel.Topics["PCW_PERIODS"]
```

Adding Joins

Example—Add a join

```
// add join between PCW_PERIODS (Day) and PCW_SALES (Order_Date)
PCWPeriods_Day = PCWPeriods.TopicItems["Day"]
PCWSales_OrderDate = PCWSales.TopicItems["Order_Date"]
Day_OrderDate_Join =
ActiveDocument.Sections["DataModel"].DataModel.Joins.Add(PCWPeriods_Day,PCWSales_OrderDa
te,
bqJoinSimpleEqual)
```

Adding Items to the Request Line

Example—Add items to the request line

```
// add items to the request line
ActiveDocument.Sections["Query"].Requests.Add("PCW_CUSTOMERS", "Store")
ActiveDocument.Sections["Query"].Requests.Add("PCW_SALES", "Store_Id")
ActiveDocument.Sections["Query"].Requests.Add("PCW_SALES", "Order_Date")
ActiveDocument.Sections["Query"].Requests.Add("PCW_SALES", "Delivery_Date")
ActiveDocument.Sections["Query"].Requests.Add("PCW_SALES", "Units")
ActiveDocument.Sections["Query"].Requests.Add("PCW_SALES", "Amount")
ActiveDocument.Sections["Query"].Requests.Add("PCW_CUSTOMERS", "City")
ActiveDocument.Sections["Query"].Requests.Add("PCW_CUSTOMERS", "State")
ActiveDocument.Sections["Query"].Requests.Add("PCW_PERIODS", "Year")
```

Adding Computed Columns to Query Request Lines

Example—Add a computed column to a query request line

```
// add computed column to Query request line - Amount/Units
ActiveDocument.Sections["Query"].Requests.AddComputedItem
("CompItem","Amount/Units",3)
```

Creating and Setting Variable Filters (Limits)

Example—Create and set variable filters

```
// create and set variable limit - Store_Id
mylimit = ActiveDocument.Sections["Query"].Limits.CreateLimit("PCW_SALES.Store_Id")
mylimit.Operator = bqLimitOperatorLessThanOrEqual
mylimit.CustomValues.Add(10)
mylimit.SelectedValues.Add(10)
ActiveDocument.Sections["Query"].Limits.Add(mylimit)
mylimit.VariableLimit = true
```

Using a BrioQuery 5.5 Limit Dialog Box to Store Values

Example—Use a BrioQuery 5.5 Limit dialog box to store values in a text box

```
ExecuteBScript("modify limit root.'Pcw Customers'.'Store Type'.'Store Type'")
var limit = ActiveDocument.Sections["Query"].Limits["Store Type"]
var TextBox = ActiveSection.Shapes["TextBox1"]
```

```
if (!limit.Ignore)
{
TextBox.Text = limit.SelectedValues[1]
}
else
{
TextBox.Text =""
}
```

Turning Off Page Headers on Report First Page

Example—For the first page of a report, turn off page headers

```
if ( PageNm==1)
{' '}
else
{"Query Processed:  "+ Format(new Date(), "d-mmm-yyyy")}
```

Turning Off the Prompt to Save Dialog Box

Example—On an OnShutdown event, shut down the Interactive Reporting application

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

12 Troubleshooting Scripts

In This Chapter

Identifying Errors

When syntax or runtime errors occur and scripts fail to execute, you must debug the code. Preventing errors is preferable to finding and fixing errors. Therefore, observe carefully the protocols required by JavaScript.

Space-Saving Variables

Exception to the code-entry rule: If you plan to use an object model path repeatedly, define it as a variable, to save space and create a compact script.

Example—What not to type:

```
ActiveDocument.Sections["Query"].DataModel.Connection.Username = "hyperion"
ActiveDocument.Sections["Query"].DataModel.Connection.SetPassword("hyperion")ActiveDocum
ent.Sections["Query"].DataModel.Connection.Connect
```

Example—What to type:

```
DMPath = ActiveDocument.Sections["Query"].DataModel.Connection
DMPath.Username = "hyperion"
DMPath.SetPassword("hyperion")
DMPath.Connect
```

Treat space-saving variables as object model paths. That is, insert periods between object model segments, and do not add unnecessary spaces.

Also, include only objects in the path; that is, do not include methods or property segments for the objects:

Example—Incorrect script, because `ActiveDocument.Sections["Query"].Limits` does not have an `Activate()` method:

```
LPath = ActiveDocument.Sections["Query"].Limits
LPath.Activate()
```

Example—Correct script:

```
LPath = ActiveDocument.Sections["Query"]
LPath.Activate()
```

Case-Sensitive Code

JavaScript is case-sensitive.

Case-related rules:

- JavaScript statements (for example, `var`, `if…else`, `while`, `switch`, and so on) begin with lowercase letters. The script `Var StringName = "John Smith"` fails because `Var` is capitalized.
- JavaScript core operators start with uppercase letters, for example, `new Date()`. The script `new date()` fails because `date` is not capitalized.
- Object model path segments begin with uppercase letters; for example, `ActiveDocument.Sections["Dashboard"].Activate()`. Both `activeDocument.Sections["Dashboard"].Activate()` and `Activedocument.Sections["Dashboard"].Activate()` fail because the `ActiveDocument` is not properly capitalized.
- You must refer to variables as you define them. For example, if you define a variable as `var StringName`, you must refer to it as `StringName`, not `Stringname` or `stringName` or `stringname`.

Assignment Operators Versus Comparison Operators

JavaScript distinguishes between assignment and comparison operators.

Example—Assignment operator:

```
myvar = 5
```

Example—Comparison operator:

```
if (myvar == 5)
```

Be careful with operators, particularly when assigning argument values to methods.

Example—Syntax that assigns hyperion to DMPath.SetPassword.

```
DMPath = ActiveDocument.Sections[”Query”].DataModel.Connection
//This works…
DMPath.SetPassword(“hyperion”)

//This does not!!!!
DMPath.SetPassword = "hyperion"
```

Conditional Tests

When using if statements, avoid impossible conditional tests.

Example— A incorrect script, which returns myvar is not 5!, even though myvar is 5, because 5 is not the same as five:

```
var myvar = 5
if ( myvar == "five")
  {
  Alert("myvar = 5!")
  }
else
  {
  Alert("myvar is not 5!")
  }
```

You need to know how variables report in various situations. The Console.Writeln() and Alert() methods are especially useful in diagnosing problems related to inaccurate reporting of variables.

Example—A correct script, in which the JavaScript core operator String is used to format myvar for the Console window:

```
var myvar = 5
Console.Writeln(String(myvar))
if ( myvar == "five")
  {
  Alert("myvar = 5!")
  }
else
  {
  Alert("myvar is not 5!")
  }
```

If you are comparing a value that you selected in a list or drop-down list to another value, verify the retrieved value before you compare it. You want to avoid confusing the placement of the selected item with the value of the selected item.

For example, if a drop–down list contains 4, 9, 15, 25, and 36 and you select 36, the following script returns myvar is 5!:

```
var myvar = DropDown1.SelectedIndex
Console.Writeln(String(myvar))
if ( myvar == 5)
  {
```

```
  Alert("myvar = 5!")
  }
else
  {
  Alert("myvar is not 5!")
  }
```

The result occurs because `DropDown1.SelectedIndex` returns the placement, not the value, of 36.

Consider another example: A drop-down list contains one, two, three, four and five. When you select five, the following script returns `myvar = five!`

```
DropDown1 = ActiveDocument.Sections["Dashboard"].Shapes.DropDown1
var myvar = DropDown1.SelectedIndex
Console.Writeln(String(myvar))
if ( myvar == 5)
  {
  Alert("myvar = five!")
  }
else
  {
  Alert("myvar is not five!")
  }
```

Five is returned because five is the fifth item. You receive the correct result, but for the wrong reason.

Consider another example: The drop-down list contains the values one, two, three, four, and five. When you select five, the following script returns the value that is displayed in the drop-down list (the correct value, five):

```
DropDown1 = ActiveDocument.Sections["Dashboard"].Shapes.DropDown1
var myvar = DropDown1[DropDown1.SelectedIndex]
Console.Writeln(String(myvar))
if ( myvar == "five")
  {
  Alert("myvar = 5!")
  }
else
  {
  Alert("myvar is not 5!")
  }
```

Syntax Displayed in the Description Pane

On the bottom left of the Script Editor, directly above the Help button, is the Description pane. The Description pane shows you the necessary syntax for any item you select in the Object browser.

For example, if, in the Object browser, you navigate to Application, then ActiveDocument, then Sections, then Query, and then Methods and select `Activate(),` the description pane reads `void Activate().`

This display indicates that the `Activate()` method does not accept arguments.

If you select `Export()`, the description pane reads

```
void Export(String Filename, BqExportFileFormat FileFormat, [optional] Boolean
IncludeHeaders)
```

This display indicates that `Export()` accepts three arguments, two required and one optional.

Recalculating Results

Scripts that include limits (filters) may execute slowly because the scripts must recalculate the data set each time a limit is modified. You can use the `SuspendRecalculate` property to prevent a results limit from forcing recalculations.

Example—Limit values are dynamically selected from a list, but recalculation occurs only after the last value is selected:

```
Sections[sect_name].Limits[limit_col].SuspendRecalculation = true;
Sections[sect_name].Limits[limit_col].SelectedValues.RemoveAll();
for(I = 1; I <= ListBox2.SelectedList.Count;I++)
{
  NewLimitValue = ListBox2.SelectedList[I];
  newname  +=  ListBox2.SelectedList[I]
  Sections[sect_name].Limits[limit_col].SelectedValues.Add(NewLimitValue);
}
Sections["Results"].Limits["1"].SuspendRecalculation = false;
Sections[sect_name].Limits[limit_col].Ignore=false; // Trigger recalculation now
```

Designing Scripts

JavaScript is an interpreted, not a compiled, language that evaluates and runs code line in sequence. If JavaScript finds a problem with a code line, it stops. Although the syntax checker of Interactive Reporting Studio Script Editor identifies some obvious syntax errors, many errors are not noticed until runtime.

Identifying whether each line of code executes or fails, as you develop scripts, is a time-effective way to identify and avoid problems.

The Console window displays error messages and alert values that are generated by the JavaScript interpreter. During script debugging cycles, you can write messages to the Console window to track the state of variables and the progress of the script. If a syntax error is detected, the error and the line number in which it occurs are displayed. You can use the line number to move directly to the error in the Script Editor.

You use one of three methods to write to the Console window: `Console.Write()`, `Console.Writeln()`, or `Alert()`.`Console.Write()` and `Console.Writeln()` are essentially identical. `Console.Write()` does not add carriage returns at the ends of lines, and `Console.Writeln()` does add carriage returns.

`Console.Writeln()` is usually, but not always, the preferred technique. It enables the script to run without user interaction, and the Console window records each line as it is written to the Console.

Note: `Console.Writeln()` is spelled with a lowercase *l* and *n*.

The Console window also displays the buffer that contains messages about the errors that occur from when Interactive Reporting Studio starts. Thus, the Console window may display information that is no longer of value.

➤ To clear the error buffer, select **Edit**, then **Clear**.

You can access the Console window from any section within a document; it remains open until you close it. When the Console window is closed, the buffer size is 1,000 bytes. When the Console window is open, the buffer size is 641 bytes.

To work through a challenging section of code, you should use `Alert()`.

Whatever method you use, you must identify the beginning and end of each script and each code line before the script or each code line executes.

Example—A script that moves to the query section and removes limits:

```
Console.Writeln("Start Query Script")
Console.Writeln("Step1")
ActiveDocument.Sections["Query"].Activate()
Console.Writeln("Step2")
ActiveDocument.Sections["Query"].Limits.RemoveAll()
Console.Writeln("Step3")
Console.Writeln("End Query Script")
```

For the example script, the Console window displays the following message:

```
Start Query Script
Step1
Step2
Step3
End Query Script
```

Code Entry

To avoid errors, add code to the Script Editor scripting pane by double-clicking an object in the Object browser or by cutting and pasting rather than by typing.

Bypass Errors

The `try-catch` block, which is borrowed from Java, is used to bypass errors.

General syntax for a `try-catch` block:

```
try
{do something}
catch(errorname)
{do something with the error}
finally
```

```
{do something else}
```

Example of a `try-catch` block:

```
QPath = ActiveDocument.Sections["Query"].Limits
try
{QPath.Activate()}
catch(e)
{Alert(e.toString())}
finally
{Alert("We're Done!")}
```

The `try-catch` block does not usually identify definition errors but does identify errors such as use of a lowercase *d* in `date()`:

```
try
{Alert(new date())}
catch(e)
{Alert(e.toString())}
finally
{Alert("We're Done!")}
```

Getting Assistance with Problem Scripts

If you follow all recommended practices and your script does not function properly, consider opening a call with Hyperion Solutions Customer Support at 1-877-901-4975 or visit the Hyperion Web site (http://www.hyperion.com).

When you contact Hyperion Solutions Customer Support, be prepared to show the Interactive Reporting document that contains the problem script and to specify the section and control within which the script resides.

If your data is confidential, consider using the sample script that ships with Interactive Reporting Studio to duplicate your Interactive Reporting document, saving your document file without results, or limiting the results sets.

➤ To set results options, select **Query**, then **Query Option**.

Problems in one script may result from problems in another script. Therefore, Hyperion Solutions Customer Support may also need to evaluate your startup scripts and your dashboard section scripts. For this reason, it is recommended that you use the `Console.Writeln()` method to identify each code line to the Console window.

13

Designing for EPM Workspace

In This Chapter

Architecture of EPM Workspace

EPM Workspace generates dynamic HTML, enabling users to interact with Interactive Reporting documents from browser interfaces. Users perform actions such as drilling into and processing data, changing chart types, and swinging pivots, and EPM Workspace generates HTML pages on demand.

EPM Workspace Components

- **Interactive Reporting HTML Servlet**—An information broker between the browser and Interactive Reporting
- **Interactive Reporting Service**—The back-end component that opens, manages, and renders HTML versions of Interactive Reporting documents (The HTML documents are returned to Interactive Reporting HTML Servlet.)
- **Data Access Service** (DAS)—The component that is responsible for and manages all database requests

EPM Workspace Performance-Enhancing Features

EPM Workspace offers several performance-enhancing features:

- Partial Document Loading
- Multithreading
- Distributed Components
- Disk Caching Of Interactive Reporting Documents

Partial Document Loading

Only required sections of Interactive Reporting documents, rather than whole documents, are loaded.

Multithreading

Rather than one process executing serially on one thread or several processes executing on multiple individual threads (requiring greater memory overhead), one process executes concurrently on multiple threads (in a multitasking or multiprocessing environment).

Distributed Components

Rather than software components being centralized within a system, they are modularized and deployed anywhere within a network, with communication coordinated by messages passed among components.

Disk Caching Of Interactive Reporting Documents

Rather than disk access being required for everyInteractive Reporting document request, data read from disk is stored in memory and thus is readily available to the next request.

Interactive Reporting Features Supported in EPM Workspace

Interactive Reporting sections:

- Drill down and drill into (chart and pivot)
- Drill anywhere (chart and pivot)
- Swing pivots
- Add and remove totals (pivot)
- Add and remove items (results, chart, and pivot)
- Data functions (chart, pivot)
- Add, remove, and modify cume (cumulative) (chart and pivot)
- Surface values (pivot)
- Show and hide items
- Sort
- Grouping labels (chart and pivot)
- Add and remove grand and break totals (results)
- Specify chart type (chart)
- Set legend on XYZ Axis (chart)
- Show values (bar, pie, and line charts)
- Process relational and OLAP queries
- Recognition of document and dashboard section events
- Blank content areas for empty sections

Interactive Reporting graphics and control objects:

- Recognition of most dashboard object events
- Interaction with dashboard controls
- Most embedded dashboard section objects
- Most dashboard and report graphics objects

Interactive Reporting object model:

- Most functions of the object model
- Definition and manipulation of filters (limits) through the object model

Interactive Reporting Web Client features:

Four of the six Interactive Reporting Web Client adaptive states. See Behaviors specific to Interactive Reporting features supported in EPM Workspace.

Behaviors specific to Interactive Reporting features supported in EPM Workspace:

- For Interactive Reporting Web Client—Adaptive States Query and Analyze and Data Model and Analyze are not completely available, because only their view, process, and analyze portions are honored. Adaptive States override defined roles.
- When Process All occurs (dashboard and report) or when the ProcessAll() method is called, users are prompted for connection information prior to processing, not following each query process.
- All errors are handled by log files. There is no feature equivalent to the Console window, but most errors are displayed.
- If no data items are supplied for a section, the browser pop-up menu is overwritten with one "add item" pop-up menu.

EPM Workspace Limitations—Designing and Using Interactive Reporting Document Sections

Because of limitations of the HTML standard or a user set specification, some Interactive Reporting functions are not supported in EPM Workspace, and some functions behave differently in EPM Workspace.

- General Functions—EPM Workspace Limitations
- Query and Data Model Sections—EPM Workspace Limitations
- Results and Table Sections—EPM Workspace Limitations
- Pivot Sections—EPM Workspace Limitations
- Chart Sections—EPM Workspace Limitations
- Dashboard Sections—EPM Workspace Limitations
- Report Sections—EPM Workspace Limitations

General Functions—EPM Workspace Limitations

General Interactive Reporting functions that cannot be performed in EPM Workspace:

- Define layout
- Format (Format defined in the published document is displayed, format is applied only through the object model.)
- Hide and show, insert and delete, and duplicate and rename sections (The functions can be performed through the object model.)
- Set or read tools options (for example, default formats)
- Export to text, Excel, Lotus, or JPEG formats
- Export scripts to text
- Use native print (can print using the browser, Acrobat (PDF format), or through Scheduler)
- Password-protect documents

- Create or use custom menus
- Insert, delete, or show page headers and footers

From the user interface, for OLAP sections, EPM Workspace supports only sort, drill-up, drill-down, and auto-size width. However, additional OLAP functionality is available through the object model.

Query and Data Model Sections—EPM Workspace Limitations

Interactive Reporting functions related to query and data model sections that EPM Workspace does not support:

- Access to data model sections from the user interface (Sections can be accessed through the object model.)
- Programmatic application of variable filters (Filters can be applied non-programmatically.)
- Query canceling
- Query log and custom SQL options
- Addition of subqueries
- Creation of local result tables or derived queries (Local results and derived queries can be displayed.)
- Creation of union queries

Ways in which Interactive Reporting functions related to query and data model sections behave differently in EPM Workspace.

- For Interactive Reporting documents that contain union queries, the first query is displayed. For union queries, the request and filter panes in data layout are read-only, and there is no union controller line.
- Subqueries are indented in the Sections pane but displayed as regular queries in the content area.
- For queries that contain multiple join paths, EPM Workspace defaults to the first join path.
- Users cannot open Interactive Reporting document (BQY) in EPM Workspace bthat contains only a data model section.

OLAPQuery Sections—EPM Workspace Limitations

Interactive Reporting functions related to OLAPQuery sections that EPM Workspace does not support:

- Drill through
- Ad hoc OLAPQuery capabilities from the user interface (OLAPQuery building is available through the object model.)

- Access to OLAPQuery sections from the user interface (Sections can be accessed through the object model.)
- Filters from the user interface (Filters are supported through the object model.)
- Programmatic application of variable filters (Filters can be applied non-programmatically.)
- Query canceling

Results and Table Sections—EPM Workspace Limitations

Interactive Reporting functions related to results and table sections that cannot be performed in EPM Workspace:

- Import results sets
- Sort computed columns based on order functions
- Show or hide row numbers
- Insert, delete, or modify computed columns (Columns can be inserted, deleted, and modified through the object model.)
- Group columns
- Suppress duplicates
- Set conditional formatting conditions
- Enable grid lines, borders, or background

For format-related items (conditional formatting, grid lines, border, and background), format can be displayed in EPM Workspace and set through the object model.

Pivot Sections—EPM Workspace Limitations

Interactive Reporting functions related to pivot sections that cannot be performed in EPM Workspace:

- Drill to detail
- Insert, delete, or modify computed columns (Sections can be accessed through the object model.)
- Change or restore pivot-label names
- Manually refresh data (Sections can be accessed through the object model.)
- Pivot data labels or corner labels
- Use the Chart-This-Pivot function
- Set or remove pivot page breaks

Chart Sections—EPM Workspace Limitations

Interactive Reporting functions related to chart sections that cannot be performed in EPM Workspace:

- Drill to detail
- Resize individual chart components
- Insert, delete, or modify computed columns (Sections can be accessed through the object model.)
- Manually refresh data (Sections can be accessed through the object model.)
- Use the Pivot-This-Chart function
- Select or specify the location of chart legends
- Reorder chart items
- Change or restore chart-label names
- Rotate pie charts

Dashboard Sections—EPM Workspace Limitations

Interactive Reporting functions related to dashboard sections that cannot be performed in EPM Workspace:

- Access dashboard design mode
- Define dashboard tab order
- Perform OnRowDoubleClick (Active ESOs), DoubleClick (ListBox), OnChange (TextBox), or OnEnter (TextBox)
- Use diagonal lines, round rectangles, or ovals
- Align or rotate text label objects

Interactive Reporting functions related to dashboard sections that behave differently in EPM Workspace:

- Active objects embedded in dashboard sections behave like hyperlinks
- View-only objects with AutoSize=Off that are embedded in dashboard sections behave like hyperlinks
- For unsupported graphics objects (diagonal lines, round rectangles, and ovals), other graphics objects are substituted (lines, rectangles, and rectangles, respectively).
- For the DropDown control, the OnSelection event is not executed if the first selection is the first list item—an HTML limitation. To resolve the problem, users must select another item prior to selecting the first item or developers must make the first item a blank entry.
- Command-button text wrap is available only through Internet Explorer, and not when the 508 accessibility feature is enabled.

Report Sections—EPM Workspace Limitations

Interactive Reporting functions related to report sections that cannot be performed in EPM Workspace:

- Enable users to build or lay out reports
- Use diagonal lines, round rectangles, or ovals
- Align or rotate text label objects

Interactive Reporting functions related to report sections that behave differently in EPM Workspace:

- The Data Path field is set to local path.
- For unsupported graphics objects (diagonal lines, round rectangles, and ovals), other graphics objects are substituted (lines, rectangles, and rectangles, respectively).

Computed Items EPM Workspace Limitations

Special characters must not be used in column or control names in Interactive Reporting documents files to be deployed in the EPM Workspace. These characters include:

- |
- $
-
- \
- half width Yen
- half width Won

Using these characters in controls is not supported, and can contribute to rendering issues.

Creating Predefined Drill-Down Paths

For EPM Workspace, you can create predefined drill-down paths that move through the levels of detail that are defined in the data model. Drill-down paths are associated with dimensional tables, which consist of numerous attributes about business processes, such as about product lines or geographical locations. As the designer, you specify the items and the order of items through which users drill down when they perform chart or pivot analysis.

Within dimensional tables, topic items are included in drill-down paths, and fact values are excluded from drill-down paths. Thus, if you want to exclude items from drill-down paths, you tag them as fact values. In EPM Workspace, unlike in Interactive Reporting Studio, you cannot use missing items that are not defined in the drill-path as pivot and chart items.

In EPM Workspace, drill-down paths are accessed from a shortcut-menu option: “Drilldown into (ITEM_NAME).” In EPM Workspace, unlike in Interactive Reporting Studio, drill-down paths are not context-sensitive. When users select from the shortcut menu, all available drill-down paths are displayed. Each path shows the topic item that is being drilled and the label item from which it is drilled. Drilled items add their return values as pivot-label or chart-label items.

Drill-down-path definitions originate in the data model or in query tables. All pivots and charts sections derived from data models that include drill-down paths inherit the drill-down-path definitions. This capability can be used to augment the Drill Anywhere option in Interactive Reporting Studio and Interactive Reporting Web Client, or the administrator can disable Drill Anywhere and permit users to drill on only one predefined path.

➤ To define drill-down paths in query or data model sections:

1 Click the topic window header.

2 Select **View**, then **Properties.**

  Topic Properties is displayed. By default, topic items are displayed in the order in which they are defined in the underlying table. You can hide or show selected items (click Hide All or Show All), or you can alphabetize items (click Sort).

3 Define the drill path:

  - Click Sort to arrange items alphabetically
  - Click Hide All and Show All to toggle the display of items in the topic

4 Click **Set as Dimension** to establish the item order in which a user can drill-down when charts or pivots are analyzed.

  The drill path is defined under Items to Display.

5 **Optional:** Move selected items up or down in the topic list by clicking **Up** or **Down.**

6 Click **OK.**

If you intend to use a topic item on the request line but eliminate it from the drill-down path, add the item to the request line before completing the following procedure.

➤ To remove items from the drill-down path:

1 Select the table to which the item belongs, right-click, and select**Properties.**

  Topic Item Properties is displayed.

2 From **Items to Display**, double-click an item

  The asterisk (*) displayed to the left of the topic item is removed.

3 Click **OK.**

Chart Sizing

When you work with Interactive Reporting charts, you see multiple rectangular regions. By default, there is one global rectangular region, called *size object*. Typically, the graphic object (which contains the chart) and other objects (such as the objects holding the chart legend and the chart labels) fit inside the size object.

When you manipulate the sizes of objects, you may affect the rendering of the chart in HTML format and cause elements to be clipped. To avoid clipping elements rendered in HTML-rendered charts, ensure that all chart elements fit inside the global object. (You can see the rectangular outlines of the objects by clicking various parts of the chart.)

Locating Errors

When syntax or runtime errors occur and scripts fail to execute, you must debug the code. Preventing errors is preferable to finding and fixing errors. Therefore, observe carefully the protocols required by JavaScript. See Chapter 12, "Troubleshooting Scripts".

Console Window

The Console window, which displays the error buffer, is not available from EPM Workspace.

Error Logs

The EPM Workspace displays errors to an HTML dialog box or generates entries to error logs:

- Interactive Reporting BI1 [server] log
- Interactive Reporting Data Access Service [server] log
- Interactive Reporting DAServlet log
- Interactive Reporting HTMLServlet message log

For information about interpreting the logs, consult Hyperion Solutions Customer Service.

try-catch Block

You can use try_catch blocks to test the usefulness of syntax or, within scripts, to isolate a sequence of steps. See "Bypass Errors" on page 284.

Controls

Controls, inserted into dashboard sections, enable users to interact with the application dynamically. Display-related controls supported for Interactive Reporting document files:

- Command button
- Radio button
- Check box
- List box
- Drop down

- Text box
- Embedded browser
- Hyperlink

Note: Using Interactive Reporting reserved keywords (such as ActiveSection) can result in unpredictable behavior.

Control Object Properties

Basic control properties such as name, visible, auto-size, and so on are accessible through the object model, but not through the EPM Workspace user interface. Some limitations apply to control properties.

Table 34 Control Limitations in EPM Workspace

Control	Limitation in EPM Workspace
Command button	Text wrap is available only in Internet Explorer and not available when the 508 accessibility feature is enabled.
Drop down	The OnSelection event does not fire when the first selection from the down-down list is the first item on the list. You must enter a blank for the first item or instruct users to select some other list item before selecting the first list item.
Text box	The OnEnter and OnChange events do not fire. **Note:** In Netscape, dashboard text boxes with the password property set to TRUE display only the bottom half of the cursor at the very top of the text box. The issue is visual only; functionality is intact.

Graphics

From EPM Workspace, users can display graphics in documents but cannot insert graphics into dashboard sections.

The graphics available for you, as a designer, to use in EPM Workspace documents:

- Text label—Overline effect; double-overline effect; vertical and horizontal rotation; vertical and horizontal, up and down rotation
- Line
- Horizontal line
- Rectangle
- Round rectangle
- Oval
- Picture—Picture clip effect (upper left corner clip of the image), picture tile effect

If you are working with graphics that EPM Workspace does not support, substitute supported graphics or omit the graphics.

Note: All graphics support the OnClick event.

Borders, Background, and Fonts

EPM Workspace enables you to apply border, background, and font properties for graphic objects in the object model but does not enable users viewing the document in EPM Workspace to modify the properties.

Events

Interactive Reporting Studio and Interactive Reporting Web Client have events (document events, dashboard-section events, and dashboard-object events) that can fire in Interactive Reporting documents, generally in response to user actions.

Within EPM Workspace, all document events (including OnStartUp, OnShutDown, OnPreProcess, and OnPostProcess) and all dashboard section events (including OnActivate and OnDeActivate) are supported. Two dashboard object events (OnClick and OnExit—for text boxes only) are supported, and four dashboard object events (OnRowDoubleClick; OnDoubleClick; OnChange; and OnEnter) are not supported.

Note: Document events can be turned on or off.

When deploying documents for use on the Web, you must evaluate how events behave within Interactive Reporting documents.

Client-Side JavaScript

You can use client-side JavaScript to designate scripts to run in client-browser sessions. Because JavaScript enables fast responses to mouse clicks, form inputs, and page-navigation actions, it is useful for validating form information.

For example, you can script a JavaScript function on the HTML page to confirm that users entered required information, such as address and telephone number. If required information was not supplied, the embedded script displays a dialog box. Thus, server response is required only for non-scriptable browser functions, and, by using JavaScript, you avoid form redrawing, server processing and download of invalid data.

Note: Object model methods that are not supported and properties that are associated with OnClientXXX event scripts are ignored by Internet Explorer and Safari. When, in a script, Netscape 7 and Mozilla encounter non-supported object model methods and properties, execution stops. Only some object model methods and properties can be associated with OnclientXXX scripts: TextBox.Text, TextBox.Enable, TextBox.Visible, TextBox.Font, TextBox.Scrollable, TextBox.Name, TextBox.Password, and TextBox.Type.

Client Status

Status indicators that client-side JavaScript uses to instruct server code to run or not run:

- ActiveSection.ClientScriptStatus
- *Object*.ClientScriptStatus

For example, if a client-side JavaScript script (which might be activated by a command button) determines that a user entered alphabetic rather than numeric data, the server-side JavaScript script does not run until the user enters the data correctly.

You use ActiveSection.ClientScriptStatus indicators, which take Boolean values, to coordinate actions between objects.

You use *Object*.ClientScriptStatus indicators, which take Boolean values, to initiate actions for objects. *Object* is a placeholder for the object name; for example, CommandButton1.ClientScriptStatus is a valid status indicator. Independent objects are subject to the script status settings of ActiveSection.

Table 35 Behavior of Status-Indicator Settings in EPM Workspace

Indicator Value	ActiveSection.ClientScriptStatus	Object ClientScriptStatus
True	Server-side scripts are initialized when client-side script execution concludes. Client-side scripts are initialized when the browser page is refreshed.	Client-side scripts are initialized when users perform a required action, such as clicking a button.
False	No server-side script is executed.	No server-side script is executed.

For example, consider a dashboard that contains a password field and Submit and Cancel buttons. The password field contains a client-side script that requires an alphanumeric password of at least 6 characters. If ActiveSection.ClientScriptStatus is false (the user entered the password incorrectly) and the user clicks Submit, the password is not sent to the server. However, the Cancel button can reset ActiveSection.ClientScriptStatus to true and enable cancel logic, that is implemented at the server level, to run.

Client-Side Events

Client-side events are displayed in the Event Trigger drop-down list of Script Editor. All events are executed in the EPM Workspace browser, not on the server.

Table 36 Client-Side Events Associated with Controls

Event	Controls That Support the Event	Action That Invokes Event
`OnClientClick`	Command button, radio button, check box, list box, drop-down list, text box	Clicking a control **Note:** Using the Euro symbol or other special characters in control names for the OnClient script causes a JavaScript error in the EPM Workspace. Dashboard control names must not contain these characters: %, #, &, *, !, , \, /
`OnClientDoubleClick`	List box	Clicking a control
`OnClientEnter`	Text box	Entering a text box
`OnClientExit`	Text box	Leaving a text box
`OnCliehttp://wealthtrack.com/ntSelection`	Drop down	Selecting an item from a drop-down list

Text Box Events and Properties

When creating client-side JavaScript, designers can use text box events and properties. In EPM Workspace, for text box events, both client-side and server-side components run, but server-side components do not execute.

Text box properties that can be used in client-side JavaScript:

- TextBox.Text
- TextBox.Enable
- TextBox.Visible
- TextBox.Font
- TextBox.Scrollable
- TextBox.Name
- TextBox.Password
- TextBox.Type

Alert Dialog Box

EPM Workspace supports use of alert dialog boxes, modal windows that display messages and are displayed as full Web pages. When a box is displayed, to continue working on the browser, users must dismiss the box (by clicking OK).

The Alert() method can be called from any supported event, except OnStartUp and OnShutDown.

Up to three custom-named buttons can be displayed on alert dialog boxes. When users select a button, the integer associated with the button is returned. For example, if the user selects button #1, the number 1 is returned.

Syntax used, in the object model, to create the alert dialog box:

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

Toolbars

Control of which toolbars are displayed for EPM Workspace versions of Interactive Reporting documents is determined in the Interactive Reporting object model. Toolbars can be hidden, to limit user control of Interactive Reporting documents. For example, the standard and formatting toolbars can be hidden when documents are opened. Toolbars can be displayed as needed. For example, the navigation toolbar can be displayed, to extend the users' ability to navigate.

All methods and properties, including all individual toolbar properties, of the toolbar collection in the Interactive Reporting object model are available to you.

Properties of the standard, paging, and navigation toolbars:

- Name
- Type
- Visible

Note: Paging toolbar properties are available as constant values in the BqToolbars group. If you try to access the paging toolbar in Interactive Reporting Studio or Interactive Reporting Web Client, the script command is ignored, no exception is recorded, and the script continues.

Toolbars Not Required in EPM Workspace

Toolbars that are not required in EPM Workspace:

- Formatting—If, from EPM Workspace, a script attempts to access associated properties for a toolbar, the script command is ignored, no exceptions are recorded, and the script continues.
- Paging and Navigation—If the standard Interactive Reporting toolbar is enabled, these toolbars are not enabled because they are subsets of the standard toolbar.

Standard Interactive Reporting Toolbar

The icons of the Interactive Reporting toolbar are specific to features used exclusively for Interactive Reporting document files:

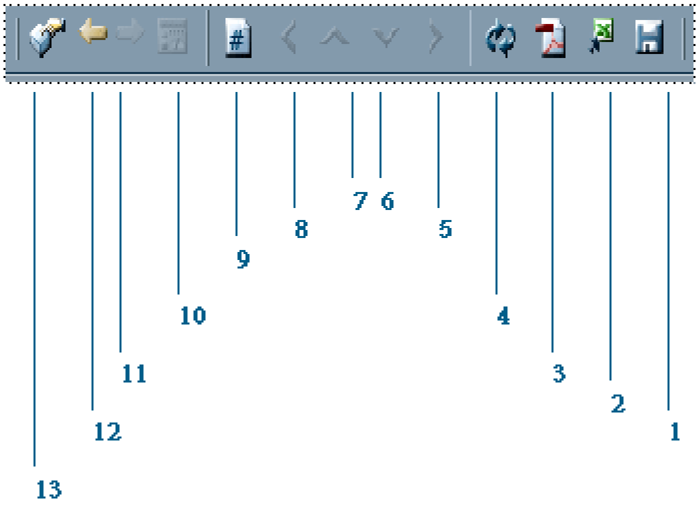


Table 37 Standard Interactive Reporting Toolbar options EPM Workspace

Number	Name	Description
1	Save	Saves the file locally and launches the Interactive Reporting document file in the Interactive Reporting Web Client so you can view the document and save it to your desktop for offline viewing (Interactive Reporting document files can be viewed only by the full desktop or Interactive Reporting Web Client. If the Interactive Reporting Web Client is not installed, the browser is launched automatically.) **Tip:** If you want to save the Interactive Reporting document file to the repository, select File, then Save or File, and then Save As.
2	Export to XLS	Exports a section to Excel and launches it inside your browser, if the MIME type is set to recognize the XLS file extension (Thereafter, the file is saved locally and manipulated through Excel. If the MIME type is not set to recognize the XLS file extension, a Save As dialog box is displayed, to enable you to save to a local location.)
3	Export to PDF	Exports a section to Portable Document Format (PDF) and launches it inside your browser, if the PDF MIME type is set in your browser (If the PDF MIME type is not set in your browser, the browser Save As dialog box is invoked.)
4	Refresh	In all but dashboard and report sections, refreshes the current section against the database server, to retrieve the most current data set In dashboard and report sections, refreshes all queries, in the order in which they are displayed in the Sections catalog of the full client (For example, in an Interactive Reporting document file with Query1, Query2, and Query3, Query1 is executed first, Query2 is executed second, and so on.)
5	Page Right	In report and chart sections, moves one page or one view to the right, respectively (To move to the first page or view to the right, select Shift+Click+right arrow.)
6	Page Down	In report and chart sections, moves one page or one view down, respectively (To move to the bottom page or view, select Shift+Click+down arrow.)
7	Page Up	In report and chart sections, moves one page or one view up, respectively (To move to the top page or view, select Shift+Click+up arrow.)
8	Page Left	In report and chart sections, moves one page or one view, respectively, to the left (To move to the first page or view to the left, press Shift+Click+left arrow.)

Number	Name	Description
9	Current Page	On the tooltip, displays the number of the current page For chart types, except pie, scatter, and bubble, tooltips display data points on the x and y axes (Based on the Maximum Bars Displayed property in the Label Axis dialog box.)
10	Dashboard Home	Displays the dashboard Home section
11	Navigate Forward	Moves to the next section
12	Navigate Back	Returns to the last viewed section
13	Data Layout	Enables the Data Layout feature

Event Controls for Toolbar Display

You use object model commands to determine which toolbars are displayed in EPM Workspace and which toolbars are defined for embedded sections in personal pages.

Within EPM Workspace, script commands that display toolbars and that are executed through trigger events supported by the object model:

- Document Scripts Trigger Events (used with OnStartup, On Shutdown, OnPreProcess, and OnPostProcess)
- Dashboard Section Display Trigger Events (used with OnActivate and OnDeactivate)
- Dashboard Object Trigger Events (used with OnClick, OnSelection, and OnExit)

Rules for Toolbars in EPM Workspace

Rules for the display, in EPM Workspace, of the standard and paging toolbars:

- Only one toolbar can be displayed at one time.
- If a script sets the Visible property of both toolbars to true, the toolbar latest in the script execution is displayed, and the Visible property of the toolbar earlier in the execution is set to false.
- EPM Workspace may display no toolbars.
- The Page *x* of *y* field displays the current page number and an unknown *y* value until you navigate to the last page. From the time that you first arrive at the last page and for the remainder of the session, the Page *x* of *y* field displays the current page number and the total-number-of-pages number. Section fields that display Page *x* of *y* values match the Page *x* of *y* tooltip. If an Interactive Reporting document file contains a section with page information and the file is saved without the results from the section, when the Interactive Reporting document file is processed, only the first page of the section is generated

These EPM Workspace toolbar rules are consistent with the rules enforced by the Personal Page Display Properties option buttons.

Personal Pages

For personal pages, toolbar display settings that are specified through object model script commands supersede toolbar display settings that are specified by owners. Therefore, the designer of the Interactive Reporting document file, not the personal pages owner, determines which functions and options are available to users, and script commands to show or hide toolbars for embedded sections are executed even if they alter the owner's toolbar selections.

For personal pages, if an Interactive Reporting document file contains no object model script commands that govern toolbar display, the owner's toolbar-display selections are enforced.

Section 508 Compliance

Section 508 compliance behavior for toolbars in Interactive Reporting document files:

- The first line of the page (invisible link) is reached by pressing Alt+H, regardless of which toolbar is displayed on the page
- An invisible link enables a user to skip the paging toolbar
- ALT text for all paging toolbar buttons is provided
- When in 508 compliant mode, the toolbar property Visible must show or hide the correct version of the standard or paging toolbar

Accessibility

The Accessibility property enables designers of dashboard sections to display dashboard sections and objects that are more accessible to disabled users.

The Accessibility property is a read-only Boolean (true or false) value that exists as a document-collection-object property:

```
Document[<collection index value or document file name>].Accessibility
ActiveDocument.Accessibility
```

To set the Accessibility property, users use the 508 Compliance Preferences setting, which is within the Browse Publish application. The setting persists only for the duration of an Interactive Reporting document session, not for the duration of an Interactive Reporting document file. Users with the Accessibility property enabled set the property by selecting from the document list, an Interactive Reporting document file to display in EPM Workspace. Rules governing which Boolean value applies to the property:

- If the Accessibility property is enabled in the Browse Publish application, the property is set to true
- If the Accessibility property is disabled, the default, which is false, applies

The property is not set for document-link selections on Interactive Reporting Web Client, regardless of the user's Accessibility status, as Interactive Reporting Web Client is not Section 508 compliant.

Guided Analysis and Reporting

Guided analysis and reporting enables you to select a point of view (POV) from one Hyperion product and seamlessly pass it to another Hyperion product. Guided analysis and reporting focuses the second product on the POV of the first product and eliminates the need for you to launch the second product and drill down to the preferred POV.

For example, if you are navigating through a bar chart in Web Analysis and require information from a relational data source, you select a bar on the chart (as the POV) and select the Related Content option. EPM Workspace launches the relevant Interactive Reporting document file and displays a relational chart that is based on the POV sent from Web Analysis.

Note: Depending upon the type of link established at design time, you may be prompted to navigate to the preferred Interactive Reporting document file.

In the background, in Oracle's Hyperion® Web Analysis, a direct or embedded link is defined, and the Interactive Reporting document file to which to send the POV is selected. When an Interactive Reporting document file is selected, a SmartCut (a link—URL—to an item in the Hyperion Foundation repository) is created, and the EPM Workspace client is invoked. The Smartcut includes the parameters that define the POV.

```
 http://aserver.hyperion.com/workspace/browse/get/AFolder/Test.bqy?
store=21&product=A&year=2003
```

Processing the POV

As a designer of POV-linked Interactive Reporting document files, you provide a script to parse the SmartCut, identify the relevant POV information, navigate the Interactive Reporting object model, and apply the POV values where and if appropriate.

➤ To process POVs:

1 **Retrieve the SmartCut for the document.**

2 **Create an OnStartUp script that performs these actions:**

- Parses the SmartCut, using JavaScript and the object-model-sessions object
- Identifies the POV parameters and assigns to them to global variables
- Traverses the object model tree and applies applicable values where and if appropriate

Parsing SmartCuts and Storing Data in Global Variables

Use the sessions object of the object model to parse the SmartCut.

Extract all relevant data and store it in global variables.

Example of using the sessions object to extract values from a SmartCut:

```
var store_id = Session.URL.Item(‘store’);
var product_id= Session.URL.Item('product');
var year_id= Session.URL.Item('year');
```

Traversing the Object Model Tree

Starting from the root of the object model tree, traverse relevant branches and nodes.

Apply values to relevant properties.

The best place to apply the property-value code is the OnStartUp script.

Example of traversing a query section branch of the object model tree and applying Store as the query limit:

```
//gets the store id value from the SmartCut
var A_Limit=Session.URL.Item('store_id');

//identifies query section to work with
var QuerySect=ActiveDocument.Sections["Query"];

//checks all limits till the appropriate one is found and applies limit
for (i=1; i<=QuerySect.Limits.Count; i++)
{
    if  (QuerySect.Limits[i].Name=="Store")
     QuerySect.Limits[i].SelectedValues.Add(A_Limit)
}
ActiveDocument.Sections["Query"].Process()
```

Object Model Items Excluded from EPM Workspace

Some objects, methods, and properties of the object model do not affect the EPM Workspace environment. Therefore, when designing an Interactive Reporting document file for use with EPM Workspace, you must exclude (not reference) some objects, methods, and properties.

If a script encounters an excluded method or property, a warning is entered in an error log, and script execution continues, if possible. If you must include excluded methods or properties, include them in Interactive Reporting document files that are designed for desktop viewing.

Object model items that are excluded from EPM Workspace (applicable to all operations that depend on Interactive Reporting Service, including Interactive Reporting document jobs):

- ActiveDocument.Close()
- ActiveDocuments.Modified()
- ActiveDocument.PromptToSave()
- ActiveDocument.SetODSPassword()
- ActiveDocument.ODSUsername (Set only)
- ActiveDocument.Save()
- ActveDocument.SaveAs()
- ActiveDocument.Sections["SectionName"].Copy()

- ActiveDocument.Sections[“Chart”].XLabels.DrillInto()
- ActiveDocument.Sections[“Dashboard”].Shapes[“TextBox1”].OnChange()
- ActiveDocument.Sections[“Dashboard”].Shapes[“TextBox1”].OnEnter()
- ActiveDocument.Sections[“Dashboard”].Shapes[“ListBox1”].OnDoubleClick()
- ActiveDocument.Sections[“Dashboard”].Shapes[“Results1”].OnRowDoubleClick()
- ActiveDocument.Sections[“OLAPQuery”].Slicers.Add()—Only the case where the last argument VariableSlicer=TRUE. Ignores the last argument, always defaulting to VariableSlicer=FALSE
- ActiveDocument.Sections["Query"].DataModel.Connection.DBLibAllowChangeDatabase
- ActiveDocument.Sections["Query"].DataModel.Connection.DBLibApiSeverity
- ActiveDocument.Sections["Query"].DataModel.Connection.DBLibDatabaseCancel
- ActiveDocument.Sections["Query"].DataModel.Connection.DBLibPacketSize
- ActiveDocument.Sections["Query"].DataModel.Connection.DBLibServerSeverity
- ActiveDocument.Sections["Query"].DataModel.Connection.DBLibUseQuotedIdentifiers
- ActiveDocument.Sections["Query"].DataModel.Connection.DBLibUseSQLTable
- ActiveDocument.Sections["Query"].DataModel.Connection.SaveWithoutUsername
- ActiveDocument.Sections["Query"].SaveResults
- ActiveSection.Shapes["Pivot1"].CellValue
- Application.CreateConnection()
- Application.DoEvents()
- Application.LoadSharedLibrary()
- Application.Shell()
- Application.Quit()
- Console.Write() (writes content to log)
- Console.WriteLn() (writes content to log)
- Documents.Add()
- Documents.New()
- Documents.Open()
- JOOLE Objects
- Dashboard Export to HTML (Dashboards only exports JPG in other client applications)

Object Model Properties Irrelevant to EPM Workspace That Must Be Retained

Some object model methods and properties do not affect the EPM Workspace user interface, but the values that they set must be retained within Interactive Reporting document files. When scripts encounter the irrelevant methods and properties, no error log entry is recorded, and script execution continues.

Note: In the EPM Workspace, an Interactive Reporting document file is saved locally for use with other Interactive Reporting applications, EPM Workspace changes are saved.

Object model items that are irrelevant to EPM Workspace but must be retained:

- ActiveDocument.ShowSectionTitleBar
- ActiveDocument.Sections["Query"].DataModel.Connection.SaveWithoutUsername
- ActiveDocument.Sections["Query"].DataModel.Connection.ShowMetadata
- ActiveDocument.Sections["Query"].DataModel.MetaDataConnection.SaveWithoutUsername
- ActiveDocument.Sections["Query"].DataModel.MetaDataConnection.ShowMetadata
- ActiveDocument.Sections["Query"].DataModel.ShowIconJoins
- ActiveDocument.Sections["Query"].Limits["Quarter"].VariableLimit
- ActiveDocument.Sections["Query"].SaveResults
- ActiveDocument.Sections["DataModel"].DataModel.AutoJoin
- ActiveDocument.Sections["DataModel"].DataModel.Connection.SaveWithoutUsername
- ActiveDocument.Sections["DataModel"].DataModel.Connection.ShowMetadata
- ActiveDocument.Sections["DataModel"].DataModel.Limits["Year"].VariableLimit
- ActiveDocument.Sections["DataModel"].DataModel.MetaDataConnection.SaveWithoutUsername
- ActiveDocument.Sections["DataModel"].DataModel.MetaDataConnection.ShowMetadata
- ActiveDocument.Sections["DataModel"].DataModel.ShowIconJoins
- ActiveDocument.Sections["Dashboard"].Shapes["EmbeddedSection"].ScrollbarsAlwaysShown
- ActiveDocument.Sections["Dashboard2"].Shapes["EmbeddedSectionResults"].ShowOutliner
- ActiveDocument.Sections["Dashboard2"].Shapes["Embedded"].ShowRowNumbers
- Application.ShowMenuBar
- Application.ShowStatusBar
- Application.StatusText
- Application.Visible
- Application.WindowState
- BqFontEffectOverDouble
- BqFontEffectOverLine

- BqFontEffectStrikeThru
- BqFontEffectSuperScript
- BqFontEffectSubScript
- Toolbars["Formatting"].Name
- Toolbars["Formatting"].Type
- Toolbars["Formatting"].Visible

Note: Some object model methods invoke (or can be set to invoke) a dialog box. EPM Workspace suppresses most dialog prompts and assumes the default selection. Three dialog boxes are supported Logon, Variable Filter, and Alert.

Note: Users must provide absolute network paths for object-model methods that require local-path information.

Note: If an explicit prompt for information from the user (that is, Connect() is not called, user names and passwords associated with OCEs are used regardless of scripted values. If an explicit prompt is called, user-supplied, scripted values are used. If scripted values are not available when an explicit prompt is called, the process behaves as if no user name or password is provided.

User Embedded HTML

You can embed images, such GIF or JPEG, and hypertext links in the cells of tables and pivots. Embedded data can be viewed in EPM Workspace and in Interactive Reporting document files to be exported to HTML pages.

User embedded HTML is written “as is” in the EPM Workspace source code wherever the text would normally use the function wrapper @HTML(<html image/link>).

How function wrappers are added to text is a customer-implementation choice. For example, if the HTML that you want to use is stored in an Oracle database column titled HTMLDATA, you might add ‘@HTML(’ || HTMLDATA || ‘)’ to the request line, where || is the Oracle string-concatenation operator. This method can also be used to generate a computed item in results.

If you want report headers or dashboard pages to contain hard coded links, such as to a corporate home page, you can enter, into a text label field, a literal string.

Example: @HTML(<a href=’http://www.hyperion.com’>Company Web site</a>)

Interactive Reporting Studio and Interactive Reporting Web Client cannot guarantee the format or appearance of cell data that is adjacent to cells that contain user-defined HTML. When viewed in a non-HTML context (such as Interactive Reporting Web Client or Interactive Reporting Studio), embedded, user-defined HTML information is displayed as text.

➤ To embed images or hyperlinks into table or pivot cells:

1 **Select the cell in which to embed the image or hyperlink.**

2 **n the cell, type @HTML(<HTML image/link>).**

When Interactive Reporting Studio and Interactive Reporting Web Client encounter the @HTML wrapper, it is deleted from the cell and the content up to but not including the wrapper is exported “as is,” without character substitutions. The parentheses () is also deleted.

BQY-XML Formatting

EPM Workspace can read and view BQY-XML documents. The BQY-XML format is defined by the XML Schema, that is provided by Hyperion in the workspace\xml\IHTMLServlet installation folder. The BQY-XML format supports dashboard sections only, with a limited set of controls:

- Text label
- Hyperlink
- Embedded browser
- Picture

The BQY-XML format is not fully functional with equivalent objects and controls in Interactive Reporting document files. To create a BQY-XML document, use an external editor (Oracle Hyperion Enterprise Performance Management System does not provide an editing tool). Design your Interactive Reporting document file after the following XML Schemas provided in the workspace\xml\IHTMLServlet installation folder:

- common.xsd
- brioquery.xsd
- eis.xsd

Verify the format of BQY-XML documents before importing them into the repository. See the bq.bqxml bq.template and bqxmlample files in the workspace\xml\IHTMLServlet folder, for information about working with or modeling an XML Schema.

User Credentials in Scripting

When scripting includes supplied credentials (through the SetPassword (Method) and Username (Property), the credentials are used in establishing an Interactive Reportingdatabase connection file. If no credentials are supplied with the script, the behavior follows the settings used by the Interactive Reporting database connection file and section mapping, together with the options of the Connect method.

Fixing Scripted Credentials Errors

Script-provided credentials that are not used as expected, cause Interactive Reporting documents files to be unusable without modification to the scripts.

When using scripted credentials, if database login failures occur, administrators can configure the system to take remedial action. Errors are reported to the user by default, and no further action is taken.

When the remedial action option is enabled, if the scripted credentials fail again to log into the database, retry the login using settings from the Interactive Reporting database connection file and section mapping, together with the options of the Connect method.

A login failure due to incorrectly scripted credentials will always be logged. The remedial action only affects users of Interactive Reporting Web Client, but does not affect EPM Workspace or Interactive Reporting Studio usage.

When the failure is logged, stop the Web server and enter `WebClient.Applications.DAServlet.RetryUsingDefaultDBCredential = true` in the ws.conf file. Save ws.conf and restart the Web server.

PrintOut() Method Support in EPM Workspace

The PrintOut() method is an Interactive Reporting document and section level function, that is available regardless of the state of the application. As long as the application is running, PrintOut() method is available through scripting, and in Oracle's Hyperion® Interactive Reporting Studio or Oracle's Hyperion® Interactive Reporting Web Client, prints the Interactive Reporting document file according to the arguments provided, or opens a standard Print dialog box.

If EPM Workspace users encounter PrintOut() method in scripts, the data received in the browser includes a PDF file.

Note: Script run in full before the PDF data is returned to the browser. This may cause some unexpected behavior with Interactive Reporting document files, such as those that display an alert that printing is complete, when in fact it has not yet started. Script authors should be aware of these factors when designing applications.

Caution! In Adobe Acrobat Reader, the default print settings are set to first page only. Reset the print range options to print the full document.

PrintOut() method is prevented from executing in Interactive Reporting documents displayed in EPM Workspace if an Alert method is used after PrintOut() method. In EPM Workspace optional parameters of the method; for example, pages to print and number of copies are ignored, use the Adobe Acrobat Reader Print dialog box to specify those options.

PrintOut() method is not supported:

- In scripts that run as the result of Document events (such as OnStartup)
- For section-level activation and deactivation events
- When used as a method of query or data model type sections

PrintOut() method has no effect on jobs; therefore, no printout occurs.

PrintOut() method takes five optional arguments:

- Start page—A number, equivalent to setting the first number to print in a print dialog box, and must represent the first page to print in the document. This value is ignored in EPM Workspace.
- End page—A number, equivalent to setting the last number to print in a print dialog box, and must represent the last page to print in the document. This value is ignored in EPM Workspace.
- Number of copies—A number, equivalent to setting the print dialog box option for number of copies, and must represent 1 to n copies. This value is ignored in EPM Workspace.
- PrintOut filename—A string, equivalent to setting the filename for printing to file, and should have a printer extension; for example, PRN or PS. The file is generated by the default printer driver, and requires that a default printer be available. The value must represent a properly formatted local or UNC path. URL syntax is not supported. This value is ignored in the Oracle Enterprise Performance Management Workspace, Fusion Edition.
- Prompt for dialog box—A Boolean, equivalent to displaying or hiding the print dialog box. The value must be false (0) or true (a number other than 0). The method does not persist with the Oracle's Hyperion® Interactive Reporting document file or application.

Note: The placeholder empty string; for example, (“”,””,””,””, “”) must be used to pass no value. If insufficient arguments are passed to the method, the PrintOut dialog box opens in a set location that cannot be programmatically changed.

Anti-Aliasing and Charts

Anti-aliasing improves chart images by displaying jagged lines as smooth, and affects performance. In applications where performance is critical, use the server.xml setting to prevent charts from being anti-aliased.

➤ To disable anti-aliasing:

1 **In a text editor, open server.xml.**

2 **Search and locate the section** `<SERVICES app="bq">`**.**

3 **Search and locate the section** `<service type="BrioQuery">`**.**

4 **In the section <properties>, copy and paste the string:**

```
<property defid="0ad70321-0001-08aa-000000e738090110"
name="DISABLE_ANTIALIASING">true</property>

 <SERVICES app="bq">
```

```
...
    <service type="BrioQuery">
...
      <properties>
...
        <property defid="0ad70321-0001-08aa-000000e738090110"
name="DISABLE_ANTIALIASING">true</property>
...
      </properties>
    </service>
  </SERVICES>
```

A

Abbreviations and Acronyms

Abbreviation	Meaning
ABC	activity-based costing
ABM	Activity-Based Management
ADO	ActiveX Data Object
AE	accountability element
AJP	Apache JServ Protocol
AJAX	Asynchronous JavaScript and XML
API	application programming interface
ASMTP	Authenticated SMTP
ASP	Active Server Pages
BAT	batch file extension
BI	Business Intelligence
BPM	Business Performance Management
CA	certificate authority
CMD	command file extension
CN	common name
COGS	cost of goods sold
CORBA	Common Object Request Broker Architecture
CPM	corporate performance management
CSC	custom calculation scripts file extension
DBCS	double-byte character set
DBMS	database management system
DC	domain component
DCOM	Distributed Component Object Model

Abbreviation	Meaning
DHTML	Dynamic Hypertext Markup Language
DIT	directory information tree
DLL	dynamic link library
DN	distinguished name
DNS	Domain Name System
DOM	Document Object Model
DSN	data source name
DTD	Document Type Definition
EAR	enterprise application archive file
EIS	executive information system
EJB	Enterprise JavaBeans
EPB	Enterprise Planning and Budgeting
EPM	Enterprise Performance Management
ERP	enterprise resource planning
ESM	editable source master
ESMTP	Extended SMTP
FP	fix pack
FTP	File Transfer Protocol
GAAP	generally accepted accounting principles
GIF	Graphics Interchange Format
GSKit7	IBM Global Security Kit 7
GUI	graphical user interface
GSM	Global Service Manager
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol layered over the SSL protocol; secure HTTP
ID	identification
I/O	input/output

Abbreviation	Meaning
IP	Internet Protocol
JAIN	Java APIs for Integrated Networks
JDBC	Java Database Connectivity
JDK	Java Development Kit
J2EE	Java 2 Platform, Enterprise Edition
JFC	Java Foundation Classes
JRE	Java Runtime Environment
JSP	JavaServer Pages
JSSE	Java Secure Socket Extension
JVM	Java Virtual Machine
KPI	key performance indicator
LAN	local area network
LCM	Life Cycle Management
LDAP	Lightweight Directory Access Protocol
LRO	linked reporting object
LSC	Local Service Configurator
LSM	Local Service Manager
MDDB	multidimensional database
MDX	Multidimensional Expression Language
MIME	Multipurpose Internet Mail Extensions
MSAD	Microsoft Active Directory
ND	Network Deployment
NFS	network file system
NTFS	New Technology file system
NTLM	Windows NT LAN Manager
OCI	Oracle Call Interface
ODBC	open database connectivity
OLAP	online analytical processing

Abbreviation	Meaning
OLE	Object Linking and Embedding
ORA	Oracle file name extension
ORB	Object Request Broker
OTL	outline file extension (Oracle Essbase)
PDF	Portable Document Format
P&L	profit and loss
POV	point of view
PRX	Adapter icon file name extension
PV	present value
RAM	random access memory
RDBMS	relational database management system
REP	report scripts file extension
RMI	Remote Method Invocation
ROM	read-only memory
RPC	Remote Procedure Call
RSC	Remote Service Configurator
RTP	runtime prompt
RUL	Business Rules file extension
SAP JCo, JCo	SAP Java Connector
SDK	Software Development Kit
SE	strategy element
SEM	Strategic Enterprise Management
SID	(Oracle) System Identification value (database instance)
SMTP	Simple Mail Transfer Protocol
SOAP	Simple Object Access Protocol
SP	service pack
SPM	Strategic Performance Management
SQL	structured query language

Abbreviation	Meaning
SSAS	SQL Server Analysis Services
SSL	Secure Sockets Layer
SSO token	single sign-on token
STP	Summary Time Period
TAR	tape archive (UNIX archive file)
TBH	To be hired
TCP/IP	Transmission Control Protocol based on Internet Protocol
UDA	user-defined attribute; Universal Data Access
UDL	Universal Data Link
UI	user interface
UID	user identification
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
URN	Uniform Resource Name
UTF-8	8-bit Unicode Transformation Format
UUID	universally unique identifier
VBIS	Vignette Business Integration Studio
VNC	Virtual Network Computing
WAN	wide area network
WAR	WebARchive file
W3C	World Wide Web Consortium
WWW	World Wide Web
XML	Extensible Markup Language
Xvfb	X virtual frame buffer
XREF	cross reference; Data reference source to a remote cube
ZIP	data compression and archival file format

Glossary

! *See bang character (!).*

#MISSING *See missing data (#MISSING).*

access permissions A set of operations that a user can perform on a resource.

accessor Input and output data specifications for data mining algorithms.

account A dimension that represents an accounting container that identifies the location and primary nature of the data.

account blocking The process by which accounts accept input data in the consolidated file. Blocked accounts do not receive their value through the additive consolidation process.

account eliminations Accounts which have their values set to zero in the consolidated file during consolidation.

account type How an account's value flows over time, and its sign behavior. Account type options can include expense, income, asset, liability, and equity.

accountability map A visual, hierarchical representation of the responsibility, reporting, and dependency structure of the accountability teams (also known as critical business areas) in an organization.

accounts dimension A dimension type that makes accounting intelligence available. Only one dimension can be defined as Accounts.

active service A service whose Run Type is set to Start rather than Hold.

active user A user who is entitled to access the system.

active user/user group The user or user group identified as the current user by user preferences. Determines default user preferences, dynamic options, access, and file permissions. You can set the active user to your user name or any user group to which you belong.

activity-level authorization Defines user access to applicationsand the types of activities they can perform on applications, independent of the data that will be operated on.

ad hoc report An online analytical query created on-the-fly by an end user.

adaptive states Interactive Reporting Web Client level of permission.

adjustment *See journal entry (JE).*

Advanced Relational Access The integration of a relational database with an Essbase multidimensional database so that all data remains in the relational database and is mapped to summary-level data residing in the Essbase database.

agent An Essbase server process that starts and stops applications and databases, manages connections from users, and handles user-access security. The agent is referred to as ESSBASE.EXE.

aggregate cell A cell comprising several cells. For example, a data cell that uses Children(Year) expands to four cells containing Quarter 1, Quarter 2, Quarter 3, and Quarter 4 data.

aggregate function A type of function, such as sum or calculation of an average, that summarizes or performs analysis on data.

aggregate limit A limit placed on an aggregated request line item or aggregated metatopic item.

aggregate storage database The database storage model designed to support large-scale, sparsely distributed data which is categorized into many, potentially large dimensions. Upper level members and formulas are dynamically calculated, and selected data values are aggregated and stored, typically with improvements in overall aggregation time.

aggregate view A collection of aggregate cells based on the levels of the members within each dimension. To reduce calculation time, values are pre-aggregated and stored as aggregate views. Retrievals then start from aggregate view totals and add up from there.

aggregation The process of rolling up and storing values in an aggregate storage database; the stored result of the aggregation process.

aggregation script In aggregate storage databases only, a file that defines a selection of aggregate views to be built into an aggregation.

alias An alternative name. For example, for a more easily identifiable column descriptor you can display the alias instead of the member name.

alias table A table that contains alternate names for members.

alternate hierarchy A hierarchy of shared members. An alternate hierarchy is based upon an existing hierarchy in a database outline, but has alternate levels in the dimension. An alternate hierarchy allows the same data to be seen from different points of view.

ancestor A branch member that has members below it. For example, the members Qtr2 and 2006 are ancestors of the member April.

appender A Log4j term for destination.

application (1) A software program designed to run a specific task or group of tasks such as a spreadsheet program or database management system. (2) A related set of dimensions and dimension members that are used to meet a specific set of analytical and/or reporting requirements.

application currency The default reporting currency for the application.

Application Migration Utility A command-line utility for migrating applications and artifacts.

area A predefined set of members and values that makes up a partition.

arithmetic data load A data load that performs operations on values in the database, such as adding 10 to each value.

artifact An individual application or repository item; for example, scripts, forms, rules files, Interactive Reporting documents, and financial reports. Also known as an object.

asset account An account type that stores values that represent a company's assets.

attribute Characteristics of a dimension member. For example, Employee dimension members may have attributes of Name, Age, or Address. Product dimension members can have several attributes, such as a size and flavor.

attribute association A relationship in a database outline whereby a member in an attribute dimension describes a characteristic of a member of its base dimension. For example, if product 100-10 has a grape flavor, the product 100-10 has the Flavor attribute association of grape. Thus, the 100-10 member of the Product dimension is associated with the Grape member of the Flavor attribute dimension.

Attribute Calculations dimension A system-defined dimension that performs these calculation operations on groups of members: Sum, Count, Avg, Min, and Max. This dimension is calculated dynamically and is not visible in the database outline. For example, using the Avg member, you can calculate the average sales value for Red products in New York in January.

attribute dimension A type of dimension that enables analysis based on the attributes or qualities of dimension members.

attribute reporting A reporting process based on the attributes of the base dimension members. *See also base dimension*.

attribute type A text, numeric, Boolean, date, or linked-attribute type that enables different functions for grouping, selecting, or calculating data. For example, because the Ounces attribute dimension has the type numeric, the number of ounces specified as the attribute of each product can be used to calculate the profit per ounce for that product.

authentication Verification of identity as a security measure. Authentication is typically based on a user name and password. Passwords and digital signatures are forms of authentication.

authentication service A core service that manages one authentication system.

auto-reversing journal A journal for entering adjustments that you want to reverse in the next period.

automated stage A stage that does not require human intervention, for example, a data load.

axis (1) A straight line that passes through a graphic used for measurement and categorization. (2) A report aspect used to arrange and relate multidimensional data, such as filters, pages, rows, and columns. For example, for a data query in Simple Basic, an axis can define columns for values for Qtr1, Qtr2, Qtr3, and Qtr4. Row data would be retrieved with totals in the following hierarchy: Market, Product.

backup A duplicate copy of an application instance.

balance account An account type that stores unsigned values that relate to a particular point in time.

balanced journal A journal in which the total debits equal the total credits.

bang character (!) A character that terminates a series of report commands and requests information from the database. A report script must be terminated with a bang character; several bang characters can be used within a report script.

bar chart A chart that can consist of one to 50 data sets, with any number of values assigned to each data set. Data sets are displayed as groups of corresponding bars, stacked bars, or individual bars in separate rows.

base currency The currency in which daily business transactions are performed.

base dimension A standard dimension that is associated with one or more attribute dimensions. For example, assuming products have flavors, the Product dimension is the base dimension for the Flavors attribute dimension.

base entity An entity at the bottom of the organization structure that does not own other entities.

batch calculation Any calculation on a database that is done in batch; for example, a calculation script or a full database calculation. Dynamic calculations are not considered to be batch calculations.

batch file An operating system file that can call multiple ESSCMD scripts and run multiple sessions of ESSCMD. On Windows-based systems, batch files have BAT file extensions. On UNIX, batch files are written as a shell script.

batch POV A collection of all dimensions on the user POV of every report and book in the batch. While scheduling the batch, you can set the members selected on the batch POV.

batch processing mode A method of using ESSCMD to write a batch or script file that can be used to automate routine server maintenance and diagnostic tasks. ESSCMD script files can execute multiple commands and can be run from the operating system command line or from within operating system batch files. Batch files can be used to call multiple ESSCMD scripts or run multiple instances of ESSCMD.

block The primary storage unit which is a multidimensional array representing the cells of all dense dimensions.

block storage database The Essbase database storage model categorizing and storing data based on the sparsity of data values defined in sparse dimensions. Data values are stored in blocks, which exist only for sparse dimension members for which there are values.

Blocked Account An account that you do not want calculated in the consolidated file because you want to enter it manually.

book A container that holds a group of similar Financial Reporting documents. Books may specify dimension sections or dimension changes.

book POV The dimension members for which a book is run.

bookmark A link to a reporting document or a Web site, displayed on a personal page of a user. The two types of bookmarks are My Bookmarks and image bookmarks.

bounding rectangle The required perimeter that encapsulates the Interactive Reporting document content when embedding Interactive Reporting document sections in a personal page, specified in pixels for height and width or row per page.

broadcast message A simple text message sent by an administrator to a user who is logged on to a Planning application. The message displays information to the user such as system availability, notification of application refresh, or application backups.

budget administrator A person responsible for setting up, configuring, maintaining, and controlling an application. Has all application privileges and data access permissions.

build method A method used to modify database outlines. Choice of a build method is based on the format of data in data source files.

business process A set of activities that collectively accomplish a business objective.

business rules Logical expressions or formulas that are created within an application to produce a desired set of resulting values.

cache A buffer in memory that holds data temporarily.

calc script A set of commands that define how a database is consolidated or aggregated. A calculation script may also contain commands that specify allocation and other calculation rules separate from the consolidation process.

Calculated Accounts You cannot alter the formulas in Calculated Accounts. These formulas are fixed in order to maintain the accounting integrity of the model you are building. For example, the formula for Net Income, a Calculated Account, is modeled into Strategic Finance and can not be changed in either historical or forecast periods.

calculated member in MaxL DML A member designed for analytical purposes and defined in the optional WITH section of a MaxL DML query.

calculation The process of aggregating data, or of running a calculation script on a database.

calculation status A consolidation status that indicates that some values or formula calculations have changed. You must reconsolidate to get the correct values for the affected entity.

calendar User-defined time periods and their relationship to each other. Q1, Q2, Q3, and Q4 comprise a calendar or fiscal year.

cascade The process of creating multiple reports for a subset of member values.

categories Groupings by which data is organized. For example, Month

cause and effect map Depicts how the elements that form your corporate strategy relate and how they work together to meet your organization's strategic goals. A Cause and Effect map tab is automatically created for each Strategy map.

CDF *See custom-defined function (CDF).*

CDM *See custom-defined macro (CDM).*

cell (1) The data value at the intersection of dimensions in a multidimensional database; the intersection of a row and a column in a worksheet. (2) A logical group of nodes belonging to one administrative domain.

cell note A text annotation for a cell in an Essbase database. Cell notes are a type of LRO.

CHANGED status Consolidation status that indicates data for an entity has changed.

chart A graphical representation of spreadsheet data. The visual nature expedites analysis, color-coding, and visual cues that aid comparisons.

chart template A template that defines the metrics to display in Workspace charts.

child A member with a parent above it in the database outline.

choice list A list of members that a report designer can specify for each dimension when defining the report's point of view. A user who wants to change the point of view for a dimension that uses a choice list can select only the members specified in that defined member list or those members that meet the criteria defined in the function for the dynamic list.

clean block A data block that where the database is fully calculated, if a calculation script calculates all dimensions at once, or if the SET CLEARUPDATESTATUS command is used in a calculation script.

cluster An array of servers or databases that behave as a single resource which share task loads and provide failover support; eliminates one server or database as a single point of failure in a system.

clustered bar charts Charts in which categories are viewed side-by-side; useful for side-by-side category analysis; used only with vertical bar charts.

code page A mapping of bit combinations to a set of text characters. Different code pages support different sets of characters. Each computer contains a code page setting for the character set requirements of the language of the computer user. In the context of this document, code pages map characters to bit combinations for non-Unicode encodings. *See also encoding*.

column A vertical display of information in a grid or table. A column can contain data from one field, derived data from a calculation, or textual information.

committed access An Essbase Kernel Isolation Level setting that affects how Essbase handles transactions. Under committed access, concurrent transactions hold long-term write locks and yield predictable results.

computed item A virtual column (as opposed to a column that is physically stored in the database or cube) that can be calculated by the database during a query, or by Interactive Reporting Studio in the Results section. Computed items are calculations of data based on functions, data items, and operators provided in the dialog box and can be included in reports or reused to calculate other data.

configuration file The security platform relies on XML documents to be configured by the product administrator or software installer. The XML document must be modified to indicate meaningful values for properties, specifying locations and attributes pertaining to the corporate authentication scenario.

connection file *See Interactive Reporting connection file (.oce)*.

consolidated file (Parent) A file into which all of the business unit files are consolidated; contains the definition of the consolidation.

consolidation The process of aggregating data from dependent entities to parent entities. For example, if the dimension Year consists of the members Qtr1, Qtr2, Qtr3, and Qtr4, its consolidation is Year.

consolidation file (*.cns) The consolidation file is a graphical interface that enables you to add, delete or move Strategic Finance files in the consolidation process using either a Chart or Tree view. It also enables you to define and modify the consolidation.

consolidation rule Identifies the rule that is executed during the consolidation of the node of the hierarchy. This rule can contain customer specific formulas appropriate for the correct consolidation of parent balances. Elimination processing can be controlled within these rules.

content Information stored in the repository for any type of file.

context variable A variable that is defined for a particular task flow to identify the context of the taskflow instance.

contribution The value added to a parent from a child entity. Each child has a contribution to its parent.

conversion rate *See exchange rate*.

cookie A segment of data placed on your computer by a Web site.

correlated subqueries Subqueries that are evaluated once for every row in the parent query; created by joining a topic item in the subquery with a topic in the parent query.

Cost of Debt Value determined by using a weighted average Yield to Maturity (YTM) of a company's entire debt portfolio. Use is the current YTM rate rather than the nominal cost of debt. The coupon rate determines the interest payment, but it does not always reflect the actual cost of the company's debt today. As required returns change, the price of a debt issue also changes so that the actual interest payments and anticipated proceeds, at maturity, yield the investors their revised required return. Therefore, the YTM fully reflects the current return demanded by debt holders and the rate at which existing debt would have to be replaced.

Cost of Equity The return an investor expects to earn on an individual stock. Using the CAPM method, the Cost of Equity is equal to:

Cost of Preferred Represents the expected return to preferred stockholders. Like debt, you need to enter the yield to maturity on preferred stock, but without the tax shielding.

critical business area (CBA) An individual or a group organized into a division, region, plant, cost center, profit center, project team, or process; also called accountability team or business area.

critical success factor (CSF) A capability that must be established and sustained to achieve a strategic objective; owned by a strategic objective or a critical process and is a parent to one or more actions.

crosstab reporting Categorizes and summarizes data in table format. The table cells contain summaries of the data that fit within the intersecting categories. For example, a crosstab report of product sales information could show size attributes, such as Small and Large, as column headings and color attributes, such as Blue and Yellow, as row headings. The cell in the table where Large and Blue intersect could contain the total sales of all Blue products that are sized Large.

cube A block of data that contains three or more dimensions. An Essbase database is a cube.

currency conversion A process that converts currency values in a database from one currency into another. For example, to convert one U. S. dollar into the European euro, the exchange rate (for example, 0.923702) is multiplied with the dollar (1* 0.923702). After conversion, the European euro amount is .92.

Currency Overrides In any input period, the selected input method can be overridden to enable input of that period's value as Default Currency/Items. To override the input method, enter a pound sign (#) either before or after the number.

currency partition A dimension type that separates local currency members from a base currency, as defined in an application. Identifies currency types, such as Actual, Budget, and Forecast.

custom calendar Any calendar created by an administrator.

custom dimension A dimension created and defined by users. Channel, product, department, project, or region could be custom dimensions.

custom property A property of a dimension or dimension member that is created by a user.

custom report A complex report from the Design Report module, composed of any combination of components.

custom-defined function (CDF) Essbase calculation functions developed in Java and added to the standard Essbase calculation scripting language using MaxL. *See also custom-defined macro (CDM)*.

custom-defined macro (CDM) Essbase macros written with Essbase calculator functions and special macro functions. Custom-defined macros use an internal Essbase macro language that enables the combination of calculation functions and they operate on multiple input parameters. *See also custom-defined function (CDF)*.

cycle through To perform multiple passes through a database while calculating it.

dashboard A collection of metrics and indicators that provide an interactive summary of your business. Dashboards enable you to build and deploy analytic applications.

data cache A buffer in memory that holds uncompressed data blocks.

data cell *See cell*.

data file cache A buffer in memory that holds compressed data (PAG) files.

data form A grid display that enables users to enter data into the database from an interface such as a Web browser, and to view and analyze data or related text. Certain dimension member values are fixed, giving users a specific view into the data.

data function That computes aggregate values, including averages, maximums, counts, and other statistics, that summarize groupings of data.

data load rules A set of criteria that determines how to load data from a text-based file, a spreadsheet, or a relational data set into a database.

data lock Prevents changes to data according to specified criteria, such as period or scenario.

data mining The process of searching through an Essbase database for hidden relationships and patterns in a large amount of data.

data model A representation of a subset of database tables.

data value *See cell*.

database connection File that stores definitions and properties used to connect to data sources and enables database references to be portable and widely used.

Default Currency Units Define the unit scale of data. For example, If you select to define your analysis in Thousands, and enter "10", this is interpreted as "10,000".

dense dimension In block storage databases, a dimension likely to contain data for every combination of dimension members. For example, time dimensions are often dense because they can contain all combinations of all members. Contrast with sparse dimension.

dependent entity An entity that is owned by another entity in the organization.

descendant Any member below a parent in the database outline. In a dimension that includes years, quarters, and months, the members Qtr2 and April are descendants of the member Year.

Design Report An interface in Web Analysis Studio for designing custom reports, from a library of components.

destination currency The currency to which balances are converted. You enter exchange rates and convert from the source currency to the destination currency. For example, when you convert from EUR to USD, the destination currency is USD.

detail chart A chart that provides the detailed information that you see in a Summary chart. Detail charts appear in the Investigate Section in columns below the Summary charts. If the Summary chart shows a Pie chart, then the Detail charts below represent each piece of the pie.

dimension A data category used to organize business data for retrieval and preservation of values. Dimensions usually contain hierarchies of related members grouped within them. For example, a Year dimension often includes members for each time period, such as quarters and months.

dimension build The process of adding dimensions and members to an Essbase outline.

dimension build rules Specifications, similar to data load rules, that Essbase uses to modify an outline. The modification is based on data in an external data source file.

dimension tab In the Pivot section, the tab that enables you to pivot data between rows and columns.

dimension table (1) A table that includes numerous attributes about a specific business process. (2) In Essbase Integration Services, a container in the OLAP model for one or more relational tables that define a potential dimension in Essbase.

dimension type A dimension property that enables the use of predefined functionality. Dimensions tagged as time have a predefined calendar functionality.

dimensionality In MaxL DML, the represented dimensions (and the order in which they are represented) in a set. For example, the following set consists of two tuples of the same dimensionality because they both reflect the dimensions (Region, Year): { (West, Feb), (East, Mar) }

direct rate A currency rate that you enter in the exchange rate table. The direct rate is used for currency conversion. For example, to convert balances from JPY to USD, In the exchange rate table, enter a rate for the period/scenario where the source currency is JPY and the destination currency is USD.

dirty block A data block containing cells that have been changed since the last calculation. Upper level blocks are marked as dirty if their child blocks are dirty (that is, they have been updated).

display type One of three Web Analysis formats saved to the repository: spreadsheet, chart, and pinboard.

dog-ear The flipped page corner in the upper right corner of the chart header area.

domain In data mining, a variable representing a range of navigation within data.

drill-down Navigation through the query result set using the dimensional hierarchy. Drilling down moves the user perspective from aggregated data to detail. For example, drilling down can reveal hierarchical relationships between years and quarters or quarters and months.

drill-through The navigation from a value in one data source to corresponding data in another source.

duplicate alias name A name that occurs more than once in an alias table and that can be associated with more than one member in a database outline. Duplicate alias names can be used with duplicate member outlines only.

duplicate member name The multiple occurrence of a member name in a database, with each occurrence representing a different member. For example, a database has two members named "New York." One member represents New York state and the other member represents New York city.

duplicate member outline A database outline containing duplicate member names.

Dynamic Calc and Store members A member in a block storage outline that Essbase calculates only upon the first retrieval of the value. Essbase then stores the calculated value in the database. Subsequent retrievals do not require calculating.

Dynamic Calc members A member in a block storage outline that Essbase calculates only at retrieval time. Essbase discards calculated values after completing the retrieval request.

dynamic calculation In Essbase, a calculation that occurs only when you retrieve data on a member that is tagged as Dynamic Calc or Dynamic Calc and Store. The member's values are calculated at retrieval time instead of being precalculated during batch calculation.

dynamic hierarchy In aggregate storage database outlines only, a hierarchy in which members are calculated at retrieval time.

dynamic member list A system-created named member set that is based on user-defined criteria. The list is refreshed automatically whenever it is referenced in the application. As dimension members are added and deleted, the list automatically reapplies the criteria to reflect the changes.

dynamic reference A pointer in the rules file to header records in a data source.

dynamic report A report containing data that is updated when you run the report.

Dynamic Time Series A process that performs period-to-date reporting in block storage databases.

dynamic view account An account type indicating that account values are calculated dynamically from the data that is displayed.

Elements Displays a list of elements available to the active section. If Query is the active section, a list of database tables is displayed. If Pivot is the active section, a list of results columns is displayed. If Dashboard is the active section, a list of embeddable sections, graphic tools, and control tools are displayed.

Eliminated Account An account that does not appear in the consolidated file.

elimination The process of zeroing out (eliminating) transactions between entities within an organization.

employee A user responsible for, or associated with, specific business objects. Employees need not work for an organization; for example, they can be consultants. Employees must be associated with user accounts for authorization purposes.

encoding A method for mapping bit combinations to characters for creating, storing, and displaying text. Each encoding has a name; for example, UTF-8. Within an encoding, each character maps to a specific bit combination; for example, in UTF-8, uppercase A maps to HEX41. *See also code page* and *locale*.

ending period A period enabling you to adjust the date range in a chart. For example, an ending period of "month", produces a chart showing information through the end of the current month.

Enterprise View An Administration Services feature that enables management of the Essbase environment from a graphical tree view. From Enterprise View, you can operate directly on Essbase artifacts.

entity A dimension representing organizational units. Examples: divisions, subsidiaries, plants, regions, products, or other financial reporting units.

Equity Beta The riskiness of a stock, measured by the variance between its return and the market return, indicated by an index called "beta". For example, if a stock's return normally moves up or down 1.2% when the market moves up or down 1%, the stock has a beta of 1.2.

essbase.cfg An optional configuration file for Essbase. Administrators may edit this file to customize Essbase Server functionality. Some configuration settings may also be used with Essbase clients to override Essbase Server settings.

EssCell A function entered into an Essbase Spreadsheet Add-in to retrieve a value representing an intersection of specific Essbase database members.

ESSCMD A command-line interface for performing Essbase operations interactively or through batch script files.

ESSLANG The Essbase environment variable that defines the encoding used to interpret text characters. *See also encoding*.

ESSMSH *See MaxL Shell*.

exceptions Values that satisfy predefined conditions. You can define formatting indicators or notify subscribing users when exceptions are generated.

exchange rate A numeric value for converting one currency to another. For example, to convert 1 USD into EUR, the exchange rate of 0.8936 is multiplied with the U.S. dollar. The European euro equivalent of $1 is 0.8936.

exchange rate type An identifier for an exchange rate. Different rate types are used because there may be multiple rates for a period and year. Users traditionally define rates at period end for the average rate of the period and for the end of the period. Additional rate types are historical rates, budget rates, forecast rates, and so on. A rate type applies to one point in time.

expense account An account that stores periodic and year-to-date values that decrease net worth if they are positive.

Extensible Markup Language (XML) A language comprising a set of tags used to assign attributes to data that can be interpreted between applications according to a schema.

external authentication Logging on to Oracle's Hyperion applications with user information stored outside the applications, typically in a corporate directory such as MSAD or NTLM.

externally triggered events Non-time-based events for scheduling job runs.

Extract, Transform, and Load (ETL) Data source-specific programs for extracting data and migrating it to applications.

extraction command An Essbase reporting command that handles the selection, orientation, grouping, and ordering of raw data extracted from a database; begins with the less than (<) character.

fact table The central table in a star join schema, characterized by a foreign key and elements drawn from a dimension table. This table typically contains numeric data that can be related to all other tables in the schema.

field An item in a data source file to be loaded into an Essbase database.

file delimiter Characters, such as commas or tabs, that separate fields in a data source.

filter A constraint on data sets that restricts values to specific criteria; for example, to exclude certain tables, metadata, or values, or to control access.

flow account An unsigned account that stores periodic and year-to-date values.

folder A file containing other files for the purpose of structuring a hierarchy.

footer Text or images at the bottom of report pages, containing dynamic functions or static text such as page numbers, dates, logos, titles or file names, and author names.

format Visual characteristics of documents or report objects.

formula A combination of operators, functions, dimension and member names, and numeric constants calculating database members.

frame An area on the desktop. There are two main areas: the navigation and workspace frames.

free-form grid An object for presenting, entering, and integrating data from different sources for dynamic calculations.

free-form reporting Creating reports by entering dimension members or report script commands in worksheets.

function A routine that returns values or database members.

generation A layer in a hierarchical tree structure that defines member relationships in a database. Generations are ordered incrementally from the top member of the dimension (generation 1) down to the child members.

generation name A unique name that describes a generation.

generic jobs Non-SQR Production Reporting or non-Interactive Reporting jobs.

global report command A command in a running report script that is effective until replaced by another global command or the file ends.

grid POV A means for specifying dimension members on a grid without placing dimensions in rows, columns, or page intersections. A report designer can set POV values at the grid level, preventing user POVs from affecting the grid. If a dimension has one grid value, you put the dimension into the grid POV instead of the row, column, or page.

group A container for assigning similar access permissions to multiple users.

GUI Graphical user interface

highlighting Depending on your configuration, chart cells or ZoomChart details may be highlighted, indicating value status: red (bad), yellow (warning), or green (good).

Historical Average An average for an account over a number of historical periods.

holding company An entity that is part of a legal entity group, with direct or indirect investments in all entities in the group.

host A server on which applications and services are installed.

host properties Properties pertaining to a host, or if the host has multiple Install_Homes, to an Install_Home. The host properties are configured from the LSC.

Hybrid Analysis An analysis mapping low-level data stored in a relational database to summary-level data stored in Essbase, combining the mass scalability of relational systems with multidimensional data.

hyperlink A link to a file, Web page, or an intranet HTML page.

Hypertext Markup Language (HTML) A programming language specifying how Web browsers display data.

identity A unique identification for a user or group in external authentication.

image bookmarks Graphic links to Web pages or repository items.

IMPACTED status Indicates changes in child entities consolidating into parent entities.

implied share A member with one or more children, but only one is consolidated, so the parent and child share a value.

inactive group A group for which an administrator has deactivated system access.

inactive service A service suspended from operating.

INACTIVE status Indicates entities deactivated from consolidation for the current period.

inactive user A user whose account has been deactivated by an administrator.

income account An account storing periodic and year-to-date values that, if positive, increase net worth.

index (1) A method where Essbase uses sparse-data combinations to retrieve data in block storage databases. (2) The index file.

index cache A buffer containing index pages.

index entry A pointer to an intersection of sparse dimensions. Index entries point to data blocks on disk and use offsets to locate cells.

index file An Essbase file storing block storage data retrieval information, residing on disk, and containing index pages.

index page A subdivision in an index file. Contains pointers to data blocks.

input data Data loaded from a source rather than calculated.

Install_Home A variable for the directory where Oracle's Hyperion applications are installed. Refers to one instance of Oracle's Hyperion application when multiple applications are installed on the same computer.

integration Process that is run to move data between Oracle's Hyperion applications using Shared Services. Data integration definitions specify the data moving between a source application and a destination application, and enable the data movements to be grouped, ordered, and scheduled.

intelligent calculation A calculation method tracking updated data blocks since the last calculation.

Interactive Reporting connection file (.oce) Files encapsulating database connection information, including: the database API (ODBC, SQL*Net, etc.), database software, the database server network address, and database user name. Administrators create and publish Interactive Reporting connection files (.oce).

intercompany elimination *See elimination.*

intercompany matching The process of comparing balances for pairs of intercompany accounts within an application. Intercompany receivables are compared to intercompany payables for matches. Matching accounts are used to eliminate intercompany transactions from an organization's consolidated totals.

intercompany matching report A report that compares intercompany account balances and indicates if the accounts are in, or out, of balance.

interdimensional irrelevance A situation in which a dimension does not intersect with other dimensions. Because the data in the dimension cannot be accessed from the non-intersecting dimensions, the non-intersecting dimensions are not relevant to that dimension.

intersection A unit of data representing the intersection of dimensions in a multidimensional database; also, a worksheet cell.

Investigation *See drill-through.*

isolation level An Essbase Kernel setting that determines the lock and commit behavior of database operations. Choices are: committed access and uncommitted access.

iteration A "pass" of the budget or planning cycle in which the same version of data is revised and promoted.

Java Database Connectivity (JDBC) A client-server communication protocol used by Java based clients and relational databases. The JDBC interface provides a call-level API for SQL-based database access.

job output Files or reports produced from running a job.

job parameters Reusable, named job parameters that are accessible only to the user who created them.

jobs Documents with special properties that can be launched to generate output. A job can contain Interactive Reporting, SQR Production Reporting, or generic documents.

join A link between two relational database tables or topics based on common content in a column or row. A join typically occurs between identical or similar items within different tables or topics. For example, a record in the Customer table is joined to a record in the Orders table because the Customer ID value is the same in each table.

journal entry (JE) A set of debit/credit adjustments to account balances for a scenario and period.

JSP Java Server Pages.

latest A Spreadsheet key word used to extract data values from the member defined as the latest time period.

layer (1) The horizontal location of members in a hierarchical structure, specified by generation (top down) or level (bottom up). (2) Position of objects relative to other objects. For example, in the Sample Basic database, Qtr1 and Qtr4 are in the same layer, so they are also in the same generation, but in a database with a ragged hierarchy, Qtr1 and Qtr4 might not be in same layer, though they are in the same generation.

legend box A box containing labels that identify the data categories of a dimension.

level A layer in a hierarchical tree structure that defines database member relationships. Levels are ordered from the bottom dimension member (level 0) up to the parent members.

level 0 block A data block for combinations of sparse, level 0 members.

level 0 member A member that has no children.

liability account An account type that stores "point in time" balances of a company's liabilities. Examples of liability accounts include accrued expenses, accounts payable, and long term debt.

life cycle management The process of managing application information from inception to retirement.

line chart A chart that displays one to 50 data sets, each represented by a line. A line chart can display each line stacked on the preceding ones, as represented by an absolute value or a percent.

line item detail The lowest level of detail in an account.

link (1) A reference to a repository object. Links can reference folders, files, shortcuts, and other links. (2) In a task flow, the point where the activity in one stage ends and another begins.

link condition A logical expression evaluated by the taskflow engine to determine the sequence of launching taskflow stages.

linked data model Documents that are linked to a master copy in a repository

linked partition A shared partition that enables you to use a data cell to link two databases. When a user clicks a linked cell in a worksheet, Essbase opens a new sheet displaying the dimensions in the linked database. The user can then drill down those dimensions.

linked reporting object (LRO) A cell-based link to an external file such as cell notes, URLs, or files with text, audio, video, or pictures. (Only cell notes are supported for Essbase LROs in Financial Reporting.)

local currency An input currency type. When an input currency type is not specified, the local currency matches the entity's base currency.

local report object A report object that is not linked to a Financial Reporting report object in Explorer. *Contrast with linked reporting object (LRO).*

local results A data model's query results. Results can be used in local joins by dragging them into the data model. Local results are displayed in the catalog when requested.

locale A computer setting that specifies a location's language, currency and date formatting, data sort order, and the character set encoding used on the computer. Essbase uses only the encoding portion. *See also encoding* and *ESSLANG.*

locale header record A text record at the beginning of some non-Unicode-encoded text files, such as scripts, that identifies the encoding locale.

location alias A descriptor that identifies a data source. The location alias specifies a server, application, database, user name, and password. Location aliases are set by DBAs at the database level using Administration Services Console, ESSCMD, or the API.

locked A user-invoked process that prevents users and processes from modifying data

locked data model Data models that cannot be modified by a user.

LOCKED status A consolidation status indicating that an entity contains data that cannot be modified.

Log Analyzer An Administration Services feature that enables filtering, searching, and analysis of Essbase logs.

LRO *See linked reporting object (LRO).*

LSC services Services configured with the Local Service Configurator. They include Global Services Manager (GSM), Local Services Manager (LSM), Session Manager, Authentication Service, Authorization Service, Publisher Service, and sometimes, Data Access Service (DAS) and Interactive Reporting Service.

managed server An application server process running in its own Java Virtual Machine (JVM).

manual stage A stage that requires human intervention to complete.

Map File Used to store the definition for sending data to or retrieving data from an external database. Map files have different extensions (.mps to send data; .mpr to retrieve data).

Map Navigator A feature that displays your current position on a Strategy, Accountability, or Cause and Effect map, indicated by a red outline.

Marginal Tax Rate Used to calculate the after-tax cost of debt. Represents the tax rate applied to the last earned income dollar (the rate from the highest tax bracket into which income falls) and includes federal, state and local taxes. Based on current level of taxable income and tax bracket, you can predict marginal tax rate.

Market Risk Premium The additional rate of return paid over the risk-free rate to persuade investors to hold "riskier" investments than government securities. Calculated by subtracting the risk-free rate from the expected market return. These figures should closely model future market conditions.

master data model An independent data model that is referenced as a source by multiple queries. When used, "Locked Data Model" is displayed in the Query section's Content pane; the data model is linked to the master data model displayed in the Data Model section, which an administrator may hide.

mathematical operator A symbol that defines how data is calculated in formulas and outlines. Can be any of the standard mathematical or Boolean operators; for example, +, -, *, /, and %.

MaxL The multidimensional database access language for Essbase, consisting of a data definition language (MaxL DDL) and a data manipulation language (MaxL DML). *See also* *MaxL DDL*, *MaxL DML*, and *MaxL Shell*.

MaxL DDL Data definition language used by Essbase for batch or interactive system-administration tasks.

MaxL DML Data manipulation language used in Essbase for data query and extraction.

MaxL Perl Module A Perl module (essbase.pm) that is part of Essbase MaxL DDL. This module can be added to the Perl package to provide access to Essbase databases from Perl programs.

MaxL Script Editor A script-development environment in Administration Services Console. MaxL Script Editor is an alternative to using a text editor and the MaxL Shell for administering Essbase with MaxL scripts.

MaxL Shell An interface for passing MaxL statements to Essbase Server. The MaxL Shell executable file is located in the Essbase bin directory (UNIX: essmsh, Windows: essmsh.exe).

MDX (multidimensional expression) The language that give instructions to OLE DB for OLAP- compliant databases, as SQL is used for relational databases. When you build the OLAPQuery section's Outliner, Interactive Reporting Clients translate requests into MDX instructions. When you process the query, MDX is sent to the database server, which returns records that answer your query. *See also* *SQL spreadsheet*.

measures Numeric values in an OLAP database cube that are available for analysis. Measures are margin, cost of goods sold, unit sales, budget amount, and so on. *See also* *fact table*.

member A discrete component within a dimension. A member identifies and differentiates the organization of similar units. For example, a time dimension might include such members as Jan, Feb, and Qtr1.

member list A named group, system- or user-defined, that references members, functions, or member lists within a dimension.

member load In Essbase Integration Services, the process of adding dimensions and members (without data) to Essbase outlines.

member selection report command A type of Report Writer command that selects member ranges based on outline relationships, such as sibling, generation, and level.

member-specific report command A type of Report Writer formatting command that is executed as it is encountered in a report script. The command affects only its associated member and executes the format command before processing the member.

merge A data load option that clears values only from the accounts specified in the data load file and replaces them with values in the data load file.

metadata A set of data that defines and describes the properties and attributes of the data stored in a database or used by an application. Examples of metadata are dimension names, member names, properties, time periods, and security.

metadata sampling The process of retrieving a sample of members in a dimension in a drill-down operation.

metadata security Security set at the member level to restrict users from accessing certain outline members.

metaoutline In Essbase Integration Services, a template containing the structure and rules for creating an Essbase outline from an OLAP model.

metric A numeric measurement computed from business data to help assess business performance and analyze company trends.

migration audit report A report generated from the migration log that provides tracking information for an application migration.

migration definition file (.mdf) A file that contains migration parameters for an application migration, enabling batch script processing.

migration log A log file that captures all application migration actions and messages.

migration snapshot A snapshot of an application migration that is captured in the migration log.

MIME Type (Multipurpose Internet Mail Extension) An attribute that describes the data format of an item, so that the system knows which application should open the object. A file's mime type is determined by the file extension or HTTP header. Plug-ins tell browsers what mime types they support and what file extensions correspond to each mime type.

mining attribute In data mining, a class of values used as a factor in analysis of a set of data.

minireport A report component that includes layout, content, hyperlinks, and the query or queries to load the report. Each report can include one or more minireports.

missing data (#MISSING) A marker indicating that data in the labeled location does not exist, contains no value, or was never entered or loaded. For example, missing data exists when an account contains data for a previous or future period but not for the current period.

model (1) In data mining, a collection of an algorithm's findings about examined data. A model can be applied against a wider data set to generate useful information about that data. (2) A file or content string containing an application-specific representation of data. Models are the basic data managed by Shared Services, of two major types: dimensional and non-dimensional application objects. (3) In Business Modeling, a network of boxes connected to represent and calculate the operational and financial flow through the area being examined.

monetary A money-related value.

multidimensional database A method of organizing, storing, and referencing data through three or more dimensions. An individual value is the intersection point for a set of dimensions.

named set In MaxL DML, a set with its logic defined in the optional WITH section of a MaxL DML query. The named set can be referenced multiple times in the query.

native authentication The process of authenticating a user name and password from within the server or application.

nested column headings A report column heading format that displays data from multiple dimensions. For example, a column heading that contains Year and Scenario members is a nested column. The nested column heading shows Q1 (from the Year dimension) in the top line of the heading, qualified by Actual and Budget (from the Scenario dimension) in the bottom line of the heading.

NO DATA status A consolidation status indicating that this entity contains no data for the specified period and account.

non-dimensional model A Shared Services model type that includes application objects such as security files, member lists, calculation scripts, and Web forms.

non-unique member name *See duplicate member name.*

note Additional information associated with a box, measure, scorecard or map element.

null value A value that is absent of data. Null values are not equal to zero.

numeric attribute range A feature used to associate a base dimension member that has a discrete numeric value with an attribute that represents a value range. For example, to classify customers by age, an Age Group attribute dimension can contain members for the following age ranges: 0-20, 21-40, 41-60, and 61-80. Each Customer dimension member can be associated with an Age Group range. Data can be retrieved based on the age ranges rather than on individual age values.

ODBC Open Database Connectivity. A database access method used from any application regardless of how the database management system (DBMS) processes the information.

OK status A consolidation status indicating that an entity has already been consolidated, and that data has not changed below it in the organization structure.

OLAP Metadata Catalog In Essbase Integration Services, a relational database containing metadata describing the nature, source, location, and type of data that is pulled from the relational data source.

OLAP model In Essbase Integration Services, a logical model (star schema) that is created from tables and columns in a relational database. The OLAP model is then used to generate the structure of a multidimensional database.

online analytical processing (OLAP) A multidimensional, multiuser, client-server computing environment for users who analyze consolidated enterprise data in real time. OLAP systems feature drill-down, data pivoting, complex calculations, trend analysis, and modeling.

Open Database Connectivity (ODBC) Standardized application programming interface (API) technology that allows applications to access multiple third-party databases.

organization An entity hierarchy that defines each entity and their relationship to others in the hierarchy.

origin The intersection of two axes.

outline The database structure of a multidimensional database, including all dimensions, members, tags, types, consolidations, and mathematical relationships. Data is stored in the database according to the structure defined in the outline.

outline synchronization For partitioned databases, the process of propagating outline changes from one database to another database.

P&L accounts (P&L) Profit and loss accounts. Refers to a typical grouping of expense and income accounts that comprise a company's income statement.

page A display of information in a grid or table often represented by the Z-axis. A page can contain data from one field, derived data from a calculation, or text.

page file Essbase data file.

page heading A report heading type that lists members represented on the current page of the report. All data values on the page have the members in the page heading as a common attribute.

page member A member that determines the page axis.

palette A JASC compliant file with a .PAL extension. Each palette contains 16 colors that complement each other and can be used to set the dashboard color elements.

parallel calculation A calculation option. Essbase divides a calculation into tasks and calculates some tasks simultaneously.

parallel data load In Essbase, the concurrent execution of data load stages by multiple process threads.

parallel export The ability to export Essbase data to multiple files. This may be faster than exporting to a single file, and it may resolve problems caused by a single data file becoming too large for the operating system to handle.

parent adjustments The journal entries that are posted to a child in relation to its parent.

parents The entities that contain one or more dependent entities that report directly to them. Because parents are both entities and associated with at least one node, they have entity, node, and parent information associated with them.

partition area A subcube within a database. A partition is composed of one or more areas of cells from a portion of the database. For replicated and transparent partitions, the number of cells within an area must be the same for the data source and target to ensure that the two partitions have the same shape. If the data source area contains 18 cells, the data target area must also contain 18 cells to accommodate the number of values.

partitioning The process of defining areas of data that are shared or linked between data models. Partitioning can affect the performance and scalability of Essbase applications.

pattern matching The ability to match a value with any or all characters of an item entered as a criterion. Missing characters may be represented by wild card values such as a question mark (?) or an asterisk (*). For example, “Find all instances of apple” returns apple, but “Find all instances of apple*” returns apple, applesauce, applecranberry, and so on.

percent consolidation The portion of a child's values that is consolidated to its parent.

percent control Identifies the extent to which an entity is controlled within the context of its group.

percent ownership Identifies the extent to which an entity is owned by its parent.

performance indicator An image file used to represent measure and scorecard performance based on a range you specify; also called a status symbol. You can use the default performance indicators or create an unlimited number of your own.

periodic value method (PVA) A process of currency conversion that applies the periodic exchange rate values over time to derive converted results.

permission A level of access granted to users and groups for managing data or other users and groups.

persistence The continuance or longevity of effect for any Essbase operation or setting. For example, an Essbase administrator may limit the persistence of user name and password validity.

personal pages A personal window to repository information. You select what information to display and its layout and colors.

personal recurring time events Reusable time events that are accessible only to the user who created them.

personal variable A named selection statement of complex member selections.

perspective A category used to group measures on a scorecard or strategic objectives within an application. A perspective can represent a key stakeholder (such as a customer, employee, or shareholder/financial) or a key competency area (such as time, cost, or quality).

pie chart A chart that shows one data set segmented in a pie formation.

pinboard One of the three data object display types. Pinboards are graphics, composed of backgrounds and interactive icons called pins. Pinboards require traffic lighting definitions.

pins Interactive icons placed on graphic reports called pinboards. Pins are dynamic. They can change images and traffic lighting color based on the underlying data values and analysis tools criteria.

pivot The ability to alter the perspective of retrieved data. When Essbase first retrieves a dimension, it expands data into rows. You can then pivot or rearrange the data to obtain a different viewpoint.

planner Planners, who comprise the majority of users, can input and submit data, use reports that others create, execute business rules, use task lists, enable e-mail notification for themselves, and use Smart View.

planning unit A data slice at the intersection of a scenario, version, and entity; the basic unit for preparing, reviewing, annotating, and approving plan data.

plot area The area bounded by X, Y, and Z axes; for pie charts, the rectangular area surrounding the pie.

plug account An account in which the system stores any out of balance differences between intercompany account pairs during the elimination process.

POV (point of view) A feature for working with dimension members not assigned to row, column, or page axes. For example, you could assign the Currency dimension to the POV and select the Euro member. Selecting this POV in data forms displays data in Euro values.

precalculation Calculating the database prior to user retrieval.

precision Number of decimal places displayed in numbers.

predefined drill paths Paths used to drill to the next level of detail, as defined in the data model.

presentation A playlist of Web Analysis documents, enabling reports to be grouped, organized, ordered, distributed, and reviewed. Includes pointers referencing reports in the repository.

preserve formulas User-created formulas kept within a worksheet while retrieving data.

primary measure A high-priority measure important to your company and business needs. Displayed in the Contents frame.

product In Shared Services, an application type, such as Planning or Performance Scorecard.

Production Reporting *See SQR Production Reporting.*

project An instance of Oracle's Hyperion products grouped together in an implementation. For example, a Planning project may consist of a Planning application, an Essbase cube, and a Financial Reporting Server instance.

promote The action to move a data unit to the next review level, allowing a user having the appropriate access to review the data. For example, an analyst may promote the data unit to the next level for his supervisor's review.

promotion The process of transferring artifacts from one environment or machine to another; for example, from a testing environment to a production environment.

property A characteristic of an artifact, such as size, type, or processing instructions.

provisioning The process of granting users and groups specific access permissions to resources.

proxy server A server acting as an intermediary between workstation users and the Internet to ensure security.

public job parameters Reusable, named job parameters created by administrators and accessible to users with requisite access privileges.

public recurring time events Reusable time events created by administrators and accessible through the access control system.

PVA *See periodic value method (PVA).*

qualified name A member name in a qualified format that differentiates duplicate member names in a duplicate member outline. For example, [Market].[East].[State].[New York] or [Market].[East].[City].[New York]

query Information requests from data providers. For example, used to access relational data sources.

query governor An Essbase Integration Server parameter or Essbase Server configuration setting that controls the duration and size of queries made to data sources.

range A set of values including upper and lower limits, and values falling between limits. Can contain numbers, amounts, or dates.

reconfigure URL URL used to reload servlet configuration settings dynamically when users are already logged on to the Workspace.

record In a database, a group of fields making up one complete entry. For example, a customer record may contain fields for name, address, telephone number, and sales data.

recurring template A journal template for making identical adjustments in every period.

recurring time event An event specifying a starting point and the frequency for running a job.

redundant data Duplicate data blocks that Essbase retains during transactions until Essbase commits updated blocks.

regular journal A feature for entering one-time adjustments for a period. Can be balanced, balanced by entity, or unbalanced.

Related Accounts The account structure groups all main and related accounts under the same main account number. The main account is distinguished from related accounts by the first suffix of the account number.

relational database A type of database that stores data in related two-dimensional tables. *Contrast with multidimensional database.*

replace A data load option that clears existing values from all accounts for periods specified in the data load file, and loads values from the data load file. If an account is not specified in the load file, its values for the specified periods are cleared.

replicated partition A portion of a database, defined through Partition Manager, used to propagate an update to data mastered at one site to a copy of data stored at another site. Users can access the data as though it were part of their local database.

Report Extractor An Essbase component that retrieves report data from the Essbase database when report scripts are run.

report object In report designs, a basic element with properties defining behavior or appearance, such as text boxes, grids, images, and charts.

report script A text file containing Essbase Report Writer commands that generate one or more production reports.

Report Viewer An Essbase component that displays complete reports after report scripts are run.

reporting currency The currency used to prepare financial statements, and converted from local currencies to reporting currencies.

repository Stores metadata, formatting, and annotation information for views and queries.

resources Objects or services managed by the system, such as roles, users, groups, files, and jobs.

restore An operation to reload data and structural information after a database has been damaged or destroyed, typically performed after shutting down and restarting the database.

restructure An operation to regenerate or rebuild the database index and, in some cases, data files.

result frequency The algorithm used to create a set of dates to collect and display results.

review level A Process Management review status indicator representing the process unit level, such as Not Started, First Pass, Submitted, Approved, and Published.

Risk Free Rate The rate of return expected from "safer" investments such as long-term U.S. government securities.

role The means by which access permissions are granted to users and groups for resources.

roll-up *See consolidation*.

root member The highest member in a dimension branch.

row heading A report heading that lists members down a report page. The members are listed under their respective row names.

RSC services Services that are configured with Remote Service Configurator, including Repository Service, Service Broker, Name Service, Event Service, and Job Service.

rules User-defined formulas.

runtime prompt A variable that users enter or select before a business rule is run.

sampling The process of selecting a representative portion of an entity to determine the entity's characteristics. *See also metadata sampling*.

saved assumptions User-defined Planning assumptions that drive key business calculations (for example, the cost per square foot of office floor space).

scale The range of values on the Y axis of a chart.

scaling Scaling determines the display of values in whole numbers, tens, hundreds, thousands, millions, and so on.

scenario A dimension for classifying data (for example, Actuals, Budget, Forecast1, and Forecast2).

schedule Specify the job that you want to run and the time and job parameter list for running the job.

scope The area of data encompassed by any Essbase operation or setting; for example, the area of data affected by a security setting. Most commonly, scope refers to three levels of granularity, where higher levels encompass lower levels. From highest to lowest, these levels are as follows: the entire system (Essbase Server), applications on Essbase Server, or databases within Essbase Server applications. *See also persistence*.

score The level at which targets are achieved, usually expressed as a percentage of the target.

scorecard Business Object that represents the progress of an employee, strategy element, or accountability element toward goals. Scorecards ascertain this progress based on data collected for each measure and child scorecard added to the scorecard.

scorecard report A report that presents the results and detailed information about scorecards attached to employees, strategy elements, and accountability elements.

secondary measure A low-priority measure, less important than primary measures. Secondary measures do not have Performance reports but can be used on scorecards and to create dimension measure templates.

Section pane Lists all sections that are available in the current Interactive Reporting Client document.

security agent A Web access management provider (for example, Netegrity SiteMinder) that protects corporate Web resources.

security platform A framework enabling Oracle's Hyperion applications to use external authentication and single sign-on.

serial calculation The default calculation setting Essbase divides a calculation pass into tasks and calculates one task at a time.

services Resources that enable business items to be retrieved, changed, added, or deleted. Examples: Authorization and Authentication.

servlet A piece of compiled code executable by a Web server.

Servlet Configurator A utility for configuring all locally installed servlets.

session The time between login and logout for a user connected to Essbase Server.

set In MaxL DML, a required syntax convention for referring to a collection of one or more tuples. For example, in the following MaxL DML query, SELECT { [100-10] } ON COLUMNS FROM Sample.Basic { [100-10] } is a set.

shared member A member that shares storage space with another member of the same name, preventing duplicate calculation of members that occur multiple times in an Essbase outline.

Shared Services Application enabling users to share data between supported Oracle's Hyperion products by publishing data to Shared Services and running data integrations.

sibling A child member at the same generation as another child member and having the same immediate parent. For example, the members Florida and New York are children of East and each other's siblings.

single sign-on Ability to access multiple Oracle's Hyperion products after a single login using external credentials.

slicer In MaxL DML, the section at the end of a query that begins with and includes the keyword WHERE.

smart tags Keywords in Microsoft Office applications that are associated with predefined actions available from the Smart Tag menu. In Oracle's Hyperion applications, smart tags can also be used to import Reporting and Analysis content, and access Financial Management and Essbase functions.

SmartCut A link to a repository item, in URL form.

snapshot Read-only data from a specific time.

source currency The currency from which values originate and are converted through exchange rates to the destination currency.

sparse dimension In block storage databases, a dimension unlikely to contain data for all member combinations when compared to other dimensions. For example, not all customers have data for all products.

SPF files Printer-independent files created by a SQR Production Reporting server, containing a representation of the actual formatted report output, including fonts, spacing, headers, footers, and so on.

SQL spreadsheet A data object that displays the result set of a SQL query.

SQR Production Reporting A specialized programming language for data access, data manipulation, and creating SQR Production Reporting documents.

stacked charts A chart where the categories are viewed on top of one another for visual comparison. This type of chart is useful for subcategorizing within the current category. Stacking can be used from the Y and Z axis in all chart types except pie and line. When stacking charts the Z axis is used as the Fact/Values axis.

stage A task description that forms one logical step within a taskflow, usually performed by an individual. A stage can be manual or automated.

stage action For automated stages, the invoked action that executes the stage.

standard dimension A dimension that is not an attribute dimension.

standard journal template A journal function used to post adjustments that have common adjustment information for each period. For example, you can create a standard template that contains the common account IDs, entity IDs, or amounts, then use the template as the basis for many regular journals.

Standard Template The Standard template is the basis for the basic Strategic Finance file. The Standard template contains all default settings. All new files are created from the Standard template unless another template is selected.

Start in Play The quickest method for creating a Web Analysis document. The Start in Play process requires you to specify a database connection, then assumes the use of a spreadsheet data object. Start in Play uses the highest aggregate members of the time and measures dimensions to automatically populate the rows and columns axes of the spreadsheet.

Status bar The status bar at the bottom of the screen displays helpful information about commands, accounts, and the current status of your data file.

stored hierarchy In aggregate storage databases outlines only. A hierarchy in which the members are aggregated according to the outline structure. Stored hierarchy members have certain restrictions, for example, they cannot contain formulas.

strategic objective (SO) A long-term goal defined by measurable results. Each strategic objective is associated with one perspective in the application, has one parent, the entity, and is a parent to critical success factors or other strategic objectives.

Strategy map Represents how the organization implements high-level mission and vision statements into lower-level, constituent strategic goals and objectives.

structure view Displays a topic as a simple list of component data items.

Structured Query Language A language used to process instructions to relational databases.

Subaccount Numbering A system for numbering subaccounts using non-sequential, whole numbers.

subscribe Flags an item or folder to receive automatic notification whenever the item or folder is updated.

Summary chart In the Investigates Section, rolls up detail charts shown below in the same column, plotting metrics at the summary level at the top of each chart column.

super service A special service used by the startCommonServices script to start the RSC services.

supervisor A user with full access to all applications, databases, related files, and security mechanisms for a server.

supporting detail Calculations and assumptions from which the values of cells are derived.

suppress rows Excludes rows containing missing values, and underscores characters from spreadsheet reports.

symmetric multiprocessing (SMP) A server architecture that enables multiprocessing and multithreading. Performance is not significantly degraded when a large number of users connect to an single instance simultaneously.

sync Synchronizes Shared Services and application models.

synchronized The condition that exists when the latest version of a model resides in both the application and in Shared Services. *See also model*.

system extract Transfers data from an application's metadata into an ASCII file.

tabs Navigable views of accounts and reports in Strategic Finance.

target Expected results of a measure for a specified period of time (day, quarter, etc.,)

task list A detailed status list of tasks for a particular user.

taskflow The automation of a business process in which tasks are passed from one taskflow participant to another according to procedural rules.

taskflow definition Represents business processes in the taskflow management system. Consists of a network of stages and their relationships; criteria indicating the start and end of the taskflow; and information about individual stages, such as participants, associated applications, associated activities, and so on.

taskflow instance Represents a single instance of a taskflow including its state and associated data.

taskflow management system Defines, creates, and manages the execution of a taskflow including: definitions, user or application interactions, and application executables.

taskflow participant The resource who performs the task associated with the taskflow stage instance for both manual and automated stages.

Taxes - Initial Balances Strategic Finance assumes that the Initial Loss Balance, Initial Gain Balance and the Initial Balance of Taxes Paid entries have taken place in the period before the first Strategic Finance time period.

TCP/IP *See Transmission Control Protocol/Internet Protocol (TCP/IP)*.

template A predefined format designed to retrieve particular data consistently.

time dimension Defines the time period that the data represents, such as fiscal or calendar periods.

time events Triggers for execution of jobs.

time scale Displays metrics by a specific period in time, such as monthly or quarterly.

time series reporting A process for reporting data based on a calendar date (for example, year, quarter, month, or week).

Title bar Displays the Strategic Finance name, the file name, and the scenario name Version box.

token An encrypted identification of one valid user or group on an external authentication system.

top and side labels Column and row headings on the top and sides of a Pivot report.

top-level member A dimension member at the top of the tree in a dimension outline hierarchy, or the first member of the dimension in sort order if there is no hierarchical relationship among dimension members. The top-level member name is generally the same as the dimension name if a hierarchical relationship exists.

trace level Defines the level of detail captured in the log file.

traffic lighting Color-coding of report cells, or pins based on a comparison of two dimension members, or on fixed limits.

transformation (1) Transforms artifacts so that they function properly in the destination environment after application migration. (2) In data mining, modifies data (bidirectionally) flowing between the cells in the cube and the algorithm.

translation *See currency conversion.*

Transmission Control Protocol/Internet Protocol (TCP/IP) A standard set of communication protocols linking computers with different operating systems and internal architectures. TCP/IP utilities are used to exchange files, send mail, and store data to various computers that are connected to local and wide area networks.

transparent login Logs in authenticated users without launching the login screen.

transparent partition A shared partition that enables users to access and change data in a remote database as though it is part of a local database

triangulation A means of converting balances from one currency to another via a third common currency. In Europe, this is the euro for member countries. For example, to convert from French franc to Italian lira, the common currency is defined as European euro. Therefore, in order to convert balances from French franc to Italian lira, balances are converted from French franc to European euro and from European euro to Italian lira.

triggers An Essbase feature whereby data is monitored according to user-specified criteria which when met cause Essbase to alert the user or system administrator.

trusted password A password that enables users authenticated for one product to access other products without reentering their passwords.

trusted user Authenticated user

tuple MDX syntax element that references a cell as an intersection of a member from each dimension. If a dimension is omitted, its top member is implied. Examples: (Jan); (Jan, Sales); ([Jan], [Sales], [Cola], [Texas], [Actual])

two-pass An Essbase property that is used to recalculate members that are dependent on the calculated values of other members. Two-pass members are calculated during a second pass through the outline.

unary operator A mathematical indicator (+, -, *, /, %) associated with an outline member. The unary operator defines how the member is calculated during a database roll-up.

Unicode-mode application An Essbase application wherein character text is encoded in UTF-8, enabling users with computers set up for different languages to share application data.

unique member name A non-shared member name that exists only once in a database outline.

unique member outline A database outline that is not enabled for duplicate member names.

upper-level block A type of data block wherein at least one of the sparse members is a parent-level member.

user directory A centralized location for user and group information. Also known as a repository or provider.

user variable Dynamically renders data forms based on a user's member selection, displaying only the specified entity. For example, user variable named Department displays specific departments and employees.

user-defined attribute (UDA) User-defined attribute, associated with members of an outline to describe a characteristic of the members. Users can use UDAs to return lists of members that have the specified UDA associated with them.

user-defined member list A named, static set of members within a dimension defined by the user.

validation A process of checking a business rule, report script, or partition definition against the outline to make sure that the object being checked is valid.

value dimension Used to define input value, translated value, and consolidation detail.

variance Difference between two values (for example, planned and actual value).

version Possible outcome used within the context of a scenario of data. For example, Budget - Best Case and Budget - Worst Case where Budget is scenario and Best Case and Worst Case are versions.

view Representation of either a year-to-date or periodic display of data.

visual cue A formatted style, such as a font or a color, that highlights specific types of data values. Data values may be dimension members; parent, child, or shared members; dynamic calculations; members containing a formula; read only data cells; read and write data cells; or linked objects.

Web server Software or hardware hosting intranet or Internet Web pages or Web applications.

weight Value assigned to an item on a scorecard that indicates the relative importance of that item in the calculation of the overall scorecard score. The weighting of all items on a scorecard accumulates to 100%. For example, to recognize the importance of developing new features for a product, the measure for New Features Coded on a developer's scorecard would be assigned a higher weighting than a measure for Number of Minor Defect Fixes.

wild card Character that represents any single character (?) or group of characters (*) in a search string.

WITH section In MaxL DML, an optional section of the query used for creating re-usable logic to define sets or members. Sets or custom members can be defined once in the WITH section, and then referenced multiple times during a query.

workbook An entire spreadsheet file with many worksheets.

write-back The ability for a retrieval client, such as a spreadsheet, to update a database value.

ws.conf A configuration file for Windows platforms.

wsconf_platform A configuration file for UNIX platforms.

XML *See Extensible Markup Language (XML).*

Y axis scale Range of values on Y axis of charts displayed in Investigate Section. For example, use a unique Y axis scale for each chart, the same Y axis scale for all Detail charts, or the same Y axis scale for all charts in the column. Often, using a common Y axis improves your ability to compare charts at a glance.

Zero Administration Software tool that identifies version number of the most up-to-date plug-in on the server.

zoom Sets the magnification of a report. For example, magnify a report to fit whole page, page width, or percentage of magnification based on 100%.

ZoomChart Used to view detailed information by enlarging a chart. Enables you to see detailed numeric information on the metric that is displayed in the chart.

Index