COGNOS®

Cognos Impromptu(R)

PowerPrompts Reference

Product Information

This document applies to Impromptu (R) Version 7.1 and may also apply to subsequent releases. To check for newer versions of this document, visit the Cognos support Web site (http://support.cognos.com).

Copyright

Table of Contents

Chapter 1: Create a PowerPrompts Application ***7***
- Starting Information and Requirements *8*
- Differences Between PowerPrompts 6.0 and Series 7 *8*
- Upgrade PowerPrompts 6.0 Applications to 7.1 *8*
- PowerPrompts Security and Access Manager Integration *9*
- Initialization File Settings *10*
- Add a Virtual Directory for Images *10*
- Start PowerPrompts *10*
- Set the HTML Editor and HTML Template *11*
- Use a JavaScript File to Add Code to Your Pages *11*

Create a New Application *12*
- Format the Entry, Exit, and Error Pages *13*
- Add a Page *13*
- Import a Page *13*
- Edit a Page *14*
- Delete a Page *14*

Link Pages: Overview *15*
- Add a Link by Dragging *16*
- Add a Link by Using the Dialog Box *16*
- Add a Link Condition *17*
- Delete a Link *17*

Dynamos: Overview *18*
- Create a Dynamo Using the Wizard *19*
- Manually Create a Dynamo *20*

Script: Overview *21*
- Add a Script *21*

Test and Deploy Your Application: Overview *22*
- Select a Report to Test Your Application *22*
- Verify the Application *23*
- Test the Application *23*
- Fix Script Errors *24*
- View the Generated Script *24*
- Notify Your Impromptu Web Reports Administrator *25*

Chapter 2: How To Topics ***27***
- Filter a Report for Multiple Values *27*
- Save User Choices Between Sessions *28*
- Create a Summary of User Choices *28*
- Add Impromptu User Class Filter to Picklists *29*
- Create a Type-in Picklist *29*
- Create a Cascading Picklist on a Single Page *30*
- Using ADO from PowerPrompts *31*
- Retrieve Database User ID and Password from Access Manager *32*
- Create a Different Look for Different Upfront Themes *32*
- Create a Single PowerPrompts Application for Multiple Languages *33*

Chapter 3: JavaScript in PowerPrompts ***35***
- Master Table of Server-Side JavaScript Objects *35*
- #include Statement and Other Preprocessor Directives *38*

App Object *39*
- BackURL Property *39*
- CurrentPage Property *40*
- Errors Property *40*
- FinalURL Property *41*
- IsTestMode Property *41*
- Path Property *42*
- ReportScript Property *42*
- Variables Property *42*
- RunDynamo Method *43*

Connection Object *43*
- Execute Method *44*

Field Object *44*
- HTMLEncodedValue Property *45*
- Index Property *45*
- Name Property *46*
- Type Property *47*
- Value Property *48*
- Operator() Method (Field) *49*

Query Object *49*
- SQL Property *50*
- AddColumn Method *51*
- AndFilterBy Method *51*
- AndSummaryFilterBy Method *52*
- AssociateColumn Method *52*
- GroupBy Method *53*
- OrFilterBy Method *53*
- OrSummaryFilterBy Method *54*
- RemoveColumn Method *54*
- SetDistinct Method *54*
- SetPromptValue Method *55*
- SortBy Method *55*

Recordset Object *56*
- CurrentRecordIndex Property *56*
- EOF Property *57*
- Fields Property *58*
- MaxRecords Property *58*
- Close Method *59*
- MoveNext Method *60*
- Open Method *60*

Request Object *62*
- Cookies Property *62*
- ServerVariables Property *62*
- Variables Property *63*

Response Object *64*
- ContentType Property *64*
- AppendCookie Method *65*
- AppendHeader Method *65*
- Clear Method *65*
- ClearContent Method *66*
- ClearHeaders Method *66*
- Redirect Method *66*
- Write Method *66*
- WriteFile Method *67*
- Writeln Method *67*

Server Object *69*
- ScriptTimeout Property *69*
- CreateObject Method *70*
- FormatNumber Method *70*
- HTMLEncode Method *71*
- URLDecode Method *71*
- URLEncode Method *72*

StringList Object *72*
- Count Property *73*
- Contains Method *73*
- Item Method *74*
- Join Method *75*
- Operator() Method (StringList) *75*
- toString Method *76*

Upfront Object *76*
- Language Property *76*
- Locale Property *77*
- Theme Property *77*
- ExecuteCommand Method *78*
- GetPageFragment Method *78*

User Object *78*
- Description Property *79*
- Email Property *79*
- Telephone Property *80*
- UserClass Property *80*
- UserClasses Property *80*
- UserName Property *81*

JavaScript Report Alteration Methods *81*
GetColumnTitle Method *82*
GetQuery Method *82*
GetReport Method *82*
GetReportObject Method *83*
AddDataItem Report Method *83*
ApplyTemplate Report Method *84*
RemoveReportObject Report Method *84*
SetListInsertCursor Report Method *85*
SetPrimaryFrame Report Method *85*
AddConditionalFormat Report Object Method *86*
ApplyStyle Report Object Method *86*
HorizontalAlign Report Object Method *86*
HorizontalAlignToColumn Report Object Method *87*
SetText Report Object Method *87*
SetTextJustification Report Object Method *88*
VerticalAlign Report Object Method *88*
ApplyStyle Column Title Method *89*
SetText Column Title Method *89*
SetTextJustification Column Title Method *89*
AddColumn Query Method *90*
AddNamedCondition Query Method *90*
AndFilterBy Query Method *90*
AndSummaryFilterBy Query Method *91*
AssociateColumn Query Method *91*
GroupBy Query Method *91*
OrFilterBy Query Method *92*
OrSummaryFilterBy Query Method *92*
RemoveColumn Query Method *92*
SetMaxRows Query Method *93*
SetPromptValue Query Method *93*
SortBy Query Method *94*
Impromptu Expressions in PowerPrompts *94*
JavaScript Client Methods *95*
FormatUserVar Method *95*
GetUserVar Method *96*
GetUserVarValues Method *96*

Index ***97***

Chapter 1: Create a PowerPrompts Application

PowerPrompts is an application that guides report consumers through a series of HTML pages where they select the information that they want in a report. The report consumer then sees the report that is generated based on the selections. For example, using a PowerPrompts application report consumers can add

- columns to a report
- column formatting
- a report template
- conditional formatting
- filters

Use PowerPrompts Developer Studio to create these Web-based applications for Impromptu Web Reports.

PowerPrompts is available with the Administrator version of Impromptu.

Use this checklist when you create a PowerPrompts application:

- Create a series of HTML pages that will prompt your report consumers to define the contents of a report.
- Create the instructions to navigate through the HTML pages.
- Pages can include conditional navigational links so that report consumers can follow branches of the PowerPrompts application rather than each page of the application.
- Create any dynamos that return data from a logical database and then add those dynamos to the appropriate HTML pages.
- Create the script that changes the report that is associated with the PowerPrompts application.
- Save the instructions as a PowerPrompts application file (.xxm).
- Select an Impromptu report with which to test the PowerPrompts application.
- Test the PowerPrompts application and correct any errors.
- Notify the Impromptu Web Reports Administrator that the PowerPrompts application is ready to be deployed.

Use PowerPrompts to reduce the number of reports you maintain. You can create a single report that users can filter to suit their needs instead of creating separate reports for each user. Therefore, one report associated with a PowerPrompts application can show the data that would require a whole series of reports without PowerPrompts.

Your report consumers can now use the same report for many queries and they need not understand the metadata behind the report when they select what they want to see.

Related Topics

- "Set the HTML Editor and HTML Template" (p. 11)
- "Start PowerPrompts" (p. 10)
- "Create a New Application" (p. 12)
- "Format the Entry, Exit, and Error Pages" (p. 13)
- "Add a Page" (p. 13)
- "Link Pages: Overview" (p. 15)
- "Dynamos: Overview" (p. 18)

Starting Information and Requirements

PowerPrompts Developer Studio is an application development tool. As a PowerPrompts application designer, you should know

- HTML
- JavaScript
- SQL (if you want to use dynamos)
- Logical databases (database definitions from Impromptu)
- Web servers

Differences Between PowerPrompts 6.0 and Series 7

There are some major differences between PowerPrompts 6.0 and Series 7:

- Scripting language is now ECMAScript (or JavaScript as most people call it)
- If you used FormatUserVar, use the following code instead:
  ```
  <%"'"+ App.Variables("StateList").Join("','") + "'"%>
  ```
- All indices are now zero based. They were one based in 6.0. For example, the following index is from PowerPrompts 6.0:
  ```
  <%GetDataCol(1)%>
  ```
  It is replaced in Series 7 with
  ```
  <%=rs.Fields(0)%>
  ```
- Function calls can now be nested.
- The Script Manager dialog box doesn't exist in Series 7. Script conditions are replaced with IF statements in the Script Editor dialog box, such as
  ```
  if ( GetUserVarAsString("SalesTarget") == "Y" )
  ```
- Report methods require a GetReport() method prefixed to them, such as, GetReport().AddDataItem.
- JavaScript is case-sensitive. For example, App.RunDynamo is correct but aPP.rUNdYNAMO is not.

Upgrade PowerPrompts 6.0 Applications to 7.1

PowerPrompts changes your PowerPrompts 6.0 applications so that they run in PowerPrompts 7.1. PowerPrompts does not move client-side JavaScript functionality to server-side or the reverse.

While PowerPrompts 7.1 can create 7.1 applications, it can equally read and execute 6.0 and 7.1 applications.

Steps to Upgrade 6.0 Application

1. From the Start menu, start PowerPrompts.
2. From the File menu, click Open.
3. Select the application (.xxm) file from the 6.0 application folder and click Open.
   You application opens in PowerPrompts Developer Studio.
4. From the File menu, click Save.
   A backup file is created when a Powerprompts 6.0 application is opened in PowerPrompts 7.1. The backup file has the numbers 60 appended to its filename. If you open a PowerPrompts 6.0 application file called Sample.xxm and save it, the backup file is called Sample60.xxm.

The application is saved as a PowerPrompts 7.1 application.

PowerPrompts 6.0 Applications Deployed with Impromptu Web Reports

PowerPrompts applications developed and published to Impromptu Web Reports 6.0 continue to operate under Impromptu Web Reports 7.1 without change or republishing. Impromptu Web Reports 7.1 opens the PowerPrompts 6.0 application and converts it to a PowerPrompts 7.1 application each time the application is executed. To save the conversion time, open the applications in PowerPrompts Developer Studio 7.1, save them, and republish to Impromptu Web Reports 7.1.

PowerPrompts Security and Access Manager Integration

Introduction

PowerPrompts may be required to communicate with databases when you run an application. PowerPrompts must provide the proper connection string and security credentials to do this. PowerPrompts provides you with two options to provide this information, embedded security information and Access Manager integration.

Connection String

The connection string is information used by UDA to open a database connection. The string contains information such as the database ID, the type of database (for example, Oracle, ODBC, MS SQL), and whether user ID and password are required. UDA can then determine what database its supposed to open, and what information UDA needs to pass to the database.

The connection string of a database is not embedded directly in the PowerPrompts application or in any of the application's HTML files. Instead, the database is identified by a logical name and the connection string can be stored in either the Cognos.ini file or in Access Manager.

The Cognos.ini file is a file used by Cognos products that predate Series 7. The file stores database connection strings. A database is listed by logical name and a connection string separated by a '='. User IDs and passwords are not stored in the Cognos.ini file. They must be provided by the application through alternate means (in a script or by prompting).

Connection strings are stored in Access Manager as a property of a data source. Administrators can create user-independent data source objects, identified by a logical name (identical to Cognos.ini data sources). The administrator must specify a connection string for each individual data source. For each data source, the administrator can create one or more data source signon. These signons are then issued to one or more Access Manager user.

When PowerPrompts connects to a database, it tries to fetch the connection string from both locations. It first queries Access Manager for the data source. The user must be able to authenticate with Access Manager, and he must have the data source listed in his reference list. If these conditions are not met, PowerPrompts will then look in the Cognos.ini for the logical database. If the database is listed, PowerPrompts will use that connection string. If the datasource cannot be found in either location, an error is displayed in the browser.

User ID and Password

User ID and passwords are treated differently than connection strings because they control access to the database. They cannot be stored unencrypted in a public location, such as, the Cognos.ini file.

The user ID and password for a database can be embedded directly in the application, either in the application file (.xxm) if the data source is accessed by a dynamo, or in an HTML page if the data source is accessed by a recordset object.

For dynamos, user IDs and passwords are specified in the Dynamo Data Source page of the Dynamo wizard. You can also set or change user ID and password for an existing dynamo, by using the the Dynamo Data Source dialog box. You select the Use the Following Authentication Information check box and type the user ID and password.

To specify user ID and password for a recordset object, authors must provide a 3rd and 4th parameter to the Recordset.Open method. These parameters represent the user ID and password, respectively.

If a user ID and password are specified for a dynamo or recordset object, PowerPrompts uses those values, and those values only. PowerPrompts will not try to authenticate with Access Manager. If the user ID and password are not embedded with the dynamo or recordset object, then PowerPrompts tries to get that information from Access Manager (the user must be able to authenticate, and must have rights to the signon information for the data source).

User ID and password are not always mandatory (in the case of the Great Outdoors sample database, for example). PowerPrompts will try to connect to the database even if there are no credentials available.

If a user ID and password are required, but they're either incorrect or unavailable, an error is displayed in the browser.

Metadata

Credentials and a connection string are needed to fetch metadata from a protected database. PowerPrompts behavior is identical to fetching regular data from database.

Initialization File Settings

PowerPrompts has an ini file (xxxpowerprompts.ini) that contains these entries.

ServerIdleLifeTime

Sets the maximum number of seconds that a new DataAccess process can be idle. Any idle DataAccess processes that exceed this limit are shut down to conserve system resources. The default value is 900 seconds (15 minutes).

Port

Sets the port number that the PowerPrompts DataAccess server listens on. The default value is 2425.

DataServersLimit

Sets the maximum number of new DataAccess processes that can be running simultaneously to service data requests. If this value is 0, there is no limit and as many processes as required are created. If this is a positive integer, the number of DataAccess processes never exceeds this value plus 1. For example, if you set the value to 5, you never have more than 6 processes running simultaneously. The default value is 0.

Add a Virtual Directory for Images

If you want to show graphics on the HTML pages of your PowerPrompts application, you can specify a virtual directory for images.

1. Create the following virtual directory on your Web server.

Alias	Directory	Access
/images	Cognos\cer*n*\webcontent\images	Read

2. Move your images to the ..\webcontent\images folder.
3. Add images to your HTML pages by using the source attribute of the image tag, for example:

```
<IMG SRC="/images/file1.gif">
```

4. Test that your PowerPrompts application shows the images.

Start PowerPrompts

Description

PowerPrompts is available with the Administrator version of Impromptu.

For more information about installing Impromptu, see the installation guide.

Steps

- From the Start menu, start PowerPrompts.

The PowerPrompts Developer Studio opens with a blank workspace. Add pages to and define links between the pages in the PowerPrompts workspace to create a PowerPrompts application.

Related Topics

- "Create a PowerPrompts Application" (p. 7)
- "Set the HTML Editor and HTML Template" (p. 11)
- "Create a New Application" (p. 12)

Set the HTML Editor and HTML Template

Description

Set the default HTML editor you want to use, for example, if you know HTML well, set Notepad as the default HTML editor.

Use regular HTML tags to format PowerPrompts pages. A page can contain any formatting that you can create by using HTML. You can format the pages after you have added them or set a template to apply to each page you add. If you set a template, all pages that you add to the application (except the Error and Exit page) are copies of the template file. You can then add more controls or remove any items you do not want. This helps you maintain a consistent look to your pages and lessens the amount of formatting you have to do.

Notes

- When you import pages into an application the HTML template file does not effect them.
- Alternatively, you can add the contents of a JavaScript file (.js) to each page using the #include statement. For an example, see *Discovering PowerPrompts*.

Steps

1. From the Tools menu, click Options.
2. Click the File Locations tab.
3. In the HTML Editor box, specify the HTML text editor that you want to use to edit the pages of your application.

   This editor is used each time you edit an HTML page from inside PowerPrompts Developer Studio.
4. In the HTML Template File box, specify the HTML template file for this application, and click OK.

   After setting the HTML template file, any pages that you add to the application (except the Error and Exit page) are copies of the template file.

Related Topics

- "Create a PowerPrompts Application" (p. 7)
- "Create a New Application" (p. 12)
- "Import a Page" (p. 13)
- "Use a JavaScript File to Add Code to Your Pages" (p. 11)

Use a JavaScript File to Add Code to Your Pages

Instead of setting an HTML template, you can use the #include statement to add JavaScript and HTML code from a JavaScript file (.js) to the pages of your application. For example, you can add a common header to all your pages or company logo graphic that is contained in the JavaScript file.

The *Discovering PowerPrompts* tutorial uses the Header.js file to create a common look for all the pages of that application.

Steps

1. From the Tools menu, click HTML Editor.
   Your HTML editor appears.
2. Type the common code you want to add to the pages of your application.
   For example, the following code adds the Header.js:
   ```
   <%#include "Header.js"%>
   ```
3. Save the file. Ensure that the file extension is .js.
4. Close your HTML editor.
5. Move the .js file to your PowerPrompts application folder.

For more information about how to use the #include statement, see "#include Statement and Other Preprocessor Directives" (p. 38).

Related Topics

- "Set the HTML Editor and HTML Template" (p. 11)

Create a New Application

Description

Organize your HTML pages in the PowerPrompts Developer Studio workspace.

Each new PowerPrompts application is created with an:
- Entry page, which is the first page that report consumers see
- Exit page, which initiates the script based on the report consumers' selections. All navigational paths through the PowerPrompts application must end with the Exit page, and no links can leave the Exit page.
- Error page, which is displayed if an error occurs when a PowerPrompts application is running. No links can be made to or from the Error page.

The path between the Entry and Exit pages can contain as many pages as you need. For more information about creating navigational paths, see "Link Pages: Overview" (p. 15).

Steps

1. From the File menu, click New.
   The New Application dialog box appears.
2. In the Application Name box, type the name you want to use.
   The Location box automatically specifies a subfolder in the default application folder. The subfolder is created with the same name as your PowerPrompts application. The subfolder will be created when you save the application.
   The application file (.xxm) and the HTML pages (.htm) used in the application are stored in this folder. The HTML pages must not be removed from the application folder.
3. In the Location box, accept the location or specify a new one, and click OK.
4. To test that PowerPrompts and your Web server are working correctly, click Run.
5. If the Entry page is shown in your Web browser, close your Web browser and return to PowerPrompts Developer Studio.

Notes

- You can insert pages for an application anywhere in the workspace. You can also move them at any time and the links will remain intact. They are also created automatically as HTML files in the application folder.
- The New Application dialog box also contains the Report to Run box. You don't need to select a report to run until you have completed your PowerPrompts application. For more information, see "Select a Report to Test Your Application" (p. 22).

Related Topics

- "Add a Page" (p. 13)
- "Create a PowerPrompts Application" (p. 7)
- "Dynamos: Overview" (p. 18)
- "Link Pages: Overview" (p. 15)
- "Set the HTML Editor and HTML Template" (p. 11)
- "Script: Overview" (p. 21)

Format the Entry, Exit, and Error Pages

If you set an HTML template, the Entry page is created with that template applied. If you didn't set an HTML template, the Entry page contains only a Submit button, which allows the report consumer to move the next page. The Error page and Exit page are not created with an HTML template. The Error page runs a PowerPrompts method that shows an error message to the report consumer. The Exit page is created with an HTML form that runs the script against the report. For an example about how to add additional formatting to these pages, see *Discovering PowerPrompts*.

You can set a new Entry page, but you cannot change the Exit or Error pages (except to rename them).

Alternatively, you can add the contents of a JavaScript file (.js) to each page using the #Include statement. For an example, see *Discovering PowerPrompts*.

Related Topics

- "Create a PowerPrompts Application" (p. 7)
- "Set the HTML Editor and HTML Template" (p. 11)

Add a Page

Description

Each page that you add to the application represents an individual HTML page that will provide a prompt for report consumers. You can include any number of prompts on an HTML page.

You can add as many HTML pages as you want to a PowerPrompts application.

Steps

1. Click Insert Page.
2. Click the workspace where you want to add the new page.
   The page is added with the name selected.
3. Type a unique name for the new page and press Enter.

Related Topics

- "Create a New Application" (p. 12)
- "Delete a Page" (p. 14)
- "Edit a Page" (p. 14)
- "Import a Page" (p. 13)
- "Link Pages: Overview" (p. 15)

Import a Page

Description

Instead of adding a blank page or a copy of the HTML template, you can import an existing HTML page that has controls or the formatting you want.

Tip: To import pages from Windows Explorer, copy and paste or drag and drop them into the workspace.

Steps

1. From the File menu, click Import HTML File.
2. Select the HTML file you want to import and click Open.
   The page appears in the PowerPrompts workspace, and a copy of the HTML file appears in the application folder.

Related Topics

- "Add a Page" (p. 13)
- "Create a New Application" (p. 12)
- "Delete a Page" (p. 14)
- "Edit a Page" (p. 14)
- "Link Pages: Overview" (p. 15)

Edit a Page

Description

Use an HTML editor to add the controls, tables, colors, and backgrounds to HTML pages.

For an example about how to add HTML controls to a page, see *Discovering PowerPrompts*.

Steps

1. In the workspace, click the page to edit.
2. From the Tools menu, click HTML Editor.
   Your page opens in the default HTML editor.
   Note: You can also right-click a page and then click Edit HTML to edit a page.

For more information about HTML, see your HTML editor documentation.

Related Topics

- "Add a Page" (p. 13)
- "Create a New Application" (p. 12)
- "Delete a Page" (p. 14)
- "Import a Page" (p. 13)
- "Link Pages: Overview" (p. 15)
- "Set the HTML Editor and HTML Template" (p. 11)

Delete a Page

Description

Delete pages that you no longer need in your PowerPrompts application. When you delete a page, you also delete any links to and from that page.

Steps

1. In the workspace, click a page.
2. Click Delete.
   PowerPrompts deletes the page and any links to and from it.

Tip

- To delete more than one page at a time, select several pages and then click Delete.

Related Topics

- "Add a Page" (p. 13)
- "Create a New Application" (p. 12)
- "Edit a Page" (p. 14)
- "Import a Page" (p. 13)
- "Link Pages: Overview" (p. 15)

Link Pages: Overview

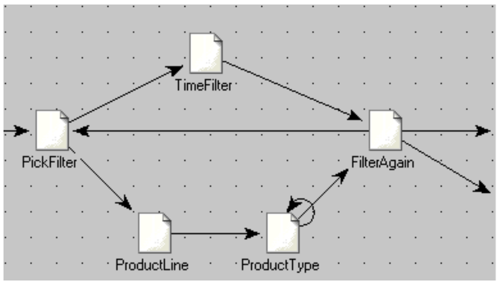
The links you add represent the possible navigational paths that report consumers are guided through when they want to view a report that has this PowerPrompts application associated with it.

You can have as many links as there are other pages in the application as long as every page can reach the Exit page. You can create a navigational path that:

- connects the Entry and Exit pages directly (with no branches or loops)
- shows the same page more than once (has a loop)
- branches from a single page
- has a page that links to itself

An example of a looping navigational path is the path from PickFilter to TimeFilter to FilterAgain and back to PickFilter.

An example of a branching navigational path is PickFilter to TimeFilter or ProductType.



Page Properties Dialog Box

You use the Page Properties dialog box to edit and delete links. When you add a link, it connects two pages in the workspace. If you open the Page Properties dialog box for the page that the link starts from, you can see the properties of that link. Each link from that page is listed in the Links tab. Each link has an entry in the Condition and Page columns.

At runtime, the list of links for each page is evaluated from top to bottom. The first link whose condition is true is the next page to open. Links below that link aren't evaluated. If <DEFAULT> appears in the Condition column, this link is the default link. The condition of the default link is always true. Ensure that the default link is the last link in the Links tab. If you place other links below the default link, the link conditions in those links will never be checked. For more information about link conditions, see "Add a Link Condition" (p. 17).

When you add a link, the Condition column shows <DEFAULT> because you haven't yet added a link condition. Deleting an existing link condition returns the <DEFAULT> to the Condition column for that link.

Tip

- To move the default link (<DEFAULT>), use the Down arrow.

Related Topics

- "Add a Link by Dragging" (p. 16)
- "Add a Link by Using the Dialog Box" (p. 16)
- "Add a Link Condition" (p. 17)
- "Create a New Application" (p. 12)
- "Create a PowerPrompts Application" (p. 7)
- "Delete a Link" (p. 17)
- "Script: Overview" (p. 21)

Add a Link by Dragging

Description

Add a link by dragging between two pages in the workspace. By using this method, you can quickly create the PowerPrompts application.

You can also add a link in the Links tab of the Page Properties dialog box.

Steps

1. Click Create Link.
2. Drag from one page to another.

Related Topics

- "Add a Link by Using the Dialog Box" (p. 16)
- "Add a Link Condition" (p. 17)
- "Delete a Link" (p. 17)
- "Link Pages: Overview" (p. 15)

Add a Link by Using the Dialog Box

Description

If a page has more than one link, use the Page Properties dialog box to create the links because you must add link conditions so that the report consumers can select the page they want. After you have added a link condition, select the page to go to when that condition is met.

You can also add a link by dragging between two pages.

Steps

1. Click Select Page.
2. Right-click the page you want the link to start from, and click Properties.
3. Click the Links tab.
4. Click the New button.
   A new link row appears.
5. If you want a condition for this link, add a link condition.
6. In the Page column, select the page to link to and click OK.

Related Topics

- "Add a Link by Dragging" (p. 16)
- "Add a Link Condition" (p. 17)
- "Delete a Link" (p. 17)
- "Link Pages: Overview" (p. 15)

Add a Link Condition

Description

If a page has more than one link, you must add link conditions so that report consumers can select the page they want. After you have set the link conditions in the Page Properties dialog box, add the HTML code to the page so that report consumers can specify the condition that selects the page they want. At runtime, the list of links for each page is evaluated from top to bottom. The first link whose condition is true is the next page to open. Conditions are based on the form variables set as you complete the application. For example, the condition FilterType = "ByTime" from the *Discovering PowerPrompts* tutorial, is evaluated as true if you click the option button named Filter by Time Periods on the PickFilter page. Links below that link aren't evaluated. Therefore, ensure that your links are in the proper order.

Steps

1. Click Select Page.
2. Right-click the page you want to add a link condition for, and click Properties.
3. Click the Links tab.
4. Click the Condition column and then click the Edit Condition button.
   The Link Condition dialog box appears.
5. Click the New button.
   PowerPrompts finds any name values (for example, NAME="FilterType") from the selected page and shows them in the Name box.
6. In the Name box, type the item to evaluate in your expression. For example, in the expression Price < 10, the name is Price.
7. In the Operator box, click the relational operator to use in your expression. For example, in the expression Price < 10, the operator is less than (<).
8. In the Value box, type the value to use in your expression. For example, in the expression Price < 10, the value is 10.
9. Click OK twice.

Tip

- If you want to use several expressions in your link condition, add more expressions and then join them together by using the AND or OR logical operators.

Related Topics

- "Add a Link by Dragging" (p. 16)
- "Add a Link by Using the Dialog Box" (p. 16)
- "Delete a Link" (p. 17)
- "Link Pages: Overview" (p. 15)

Delete a Link

Description

As you create your PowerPrompts application, you may see links that you no longer need. Delete these links by using the Links tab of the Page Properties dialog box. If you no longer need the page, deleting the page also removes any links to and from it.

Steps

1. Click Select Page.
2. Right-click the page where the link that you want to delete starts from, and click Properties.
3. Click the Links tab.
4. Click the link that you want to delete.
   The page that this link goes to is listed in the Page column.

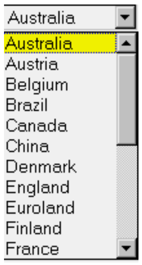
5. Click Delete and then click OK.
   The link is removed from the workspace.

Related Topics

- "Add a Link by Dragging" (p. 16)
- "Add a Link by Using the Dialog Box" (p. 16)
- "Add a Link Condition" (p. 17)
- "Delete a Page" (p. 14)
- "Link Pages: Overview" (p. 15)

Dynamos: Overview

You create a dynamo to format the result of an SQL query against a logical database as defined in Database Definition Manager. For example, you can create a dynamo that retrieves countries from a data source. The same dynamo may contain different countries, depending on what is contained in the data source or as the data source is updated. The list appears when the report consumer views the HTML page that contains the App.RunDynamo method.



For a dynamo, you specify:

- Data Source -- This is the logical database you want to query.
- SQL -- This is the SQL code that defines the query.
- Definition -- This is the JavaScript code that formats the result set.

Before you can query a database, you must already have a logical database as defined in your Cognos.ini file or in Access Manager. If you want to use ODBC connections, create a logical database for the ODBC connection or use ADO directly by using the Server.CreateObject JavaScript method. Also, old PowerPrompts 6.0 applications that use ODBC connections work in PowerPrompts Series 7.

For more information about creating logical databases, see the Database Definition Manager help.

Dynamo Manager Dialog Box

Use the Dynamo Manager dialog box to add, edit, and delete dynamos. Each dynamo you create has an entry in the Name and Data Source columns and may have an entry in the Description column. You can copy dynamos from this dialog box and paste them into your HTML pages.

Note: You may receive an error message when trying to add a data source in a dynamo on Windows 2000 or Windows 2000 Professional. If so, search your local drive for the file, MSSTDFMT.DLL. Ensure that the file is located in the \system32 directory on your computer. If you cannot find this file, contact your system administrator.

Tip: Instead of creating a dynamo, you can write JavaScript code on the HTML page to return values from a SQL query. For an example using JavaScript code instead of a dynamo, see *Discovering PowerPrompts*.

Related Topics

- "Create a Dynamo Using the Wizard" (p. 19)
- "Manually Create a Dynamo" (p. 20)
- "Create a New Application" (p. 12)
- "Create a PowerPrompts Application" (p. 7)

Create a Dynamo Using the Wizard

Description

Use the Dynamo wizard to guide you when you create a dynamo that uses a data source. It is easier to create a dynamo with the wizard than to create one manually.

Steps

1. Click Dynamo Manager.
2. Click Dynamo wizard and follow the instructions until the Select Data Source page.
3. If you want to provide the user ID and password, select the Use The Following Authentication Information check box and then type the user ID and password you want.
4. In the Logical Database box, select the logical database you want and click Next.
5. If you want only unique values retrieved, select the Use Only Unique Values check box.
6. Select the table to provide the data for the dynamo and click Next.
7. If you are creating a list, radio button, or check box, specify the HTML control name, HTML control value, and Control label and click Next.

Control	Purpose
HTML Control Name	The Name used in conditions. In the HTML listed below, the HTML Control Name is FilterType. <INPUT type="radio" NAME="FilterType" VALUE="ByTime">>
HTML Control Value	The Value used in conditions. In the HTML listed below, the HTML Control Value is ByTime. <INPUT type="radio" NAME="FilterType" VALUE="ByTime">
Control Label	The data actually displayed by this dynamo. This may be the same as the Value but does not have to be.

   If you are creating a table, in the HTML Title column, select the columns to show in the table, use the Move Up and Move Down buttons to set the display order, and click Next.

8. In the Summary box, ensure that the information is correct, and then click Finish.
9. Right-click your dynamo and then click Copy.
10. Paste the dynamo into the <form> section of your HTML page and save the page.
11. Close the Dynamo Manager dialog box.

Related Topics

- "Manually Create a Dynamo" (p. 20)
- "Dynamos: Overview" (p. 18)

Manually Create a Dynamo

Description

You can manually create dynamos. You can write more complicated SQL when you manually create the dynamo than when you use the Dynamo wizard.

Tip: You can also create dynamos that do not use a logical database. For example, if you have many pages in your application, you can create a dynamo to insert a title for each page. If you want to change the title, change only the dynamo definition, and each page that uses that title will change. To do this, create a dynamo that does not use a data source, and then insert the title and HTML formatting into the Dynamo Definition dialog box. On each page where you want that title to appear, add the `<%=App.RunDynamo("dynamo name")%>` tag.

Steps to Specify the Dynamo Data Source

1. Click Dynamo Manager.
2. Click New.
   A new dynamo row appears.
3. In the Name column, type the name you want.
4. Click Edit and then click Data Source.
   The Dynamo Data Source dialog box appears.
5. Click No Data Connectivity if your dynamo is not to retrieve any data.
6. If you want to dynamically retrieve data, click User This Logical Database and from the drop-down list, select the name of the logical database.
7. If you want to provide the user ID and password, select the Use The Following Authentication Information check box and then type the user ID and password you want.
8. If you want to limit the number of records retrieved, select the Limit Dynamo To check box and then, in the Records box, set the number you want and then click OK.
   The Dynamo Manager dialog box reappears.

Steps to Specify the Dynamo SQL and Definition

1. Click Edit and then click SQL.
   The Dynamo SQL dialog box appears.
2. Type the SQL to retrieve the data from the data source. You assign your SQL to the SQL property of a Dynamo object as follows:
   ```
   Dynamo.SQL = ...
   ```
   The following example returns ProductLineCode and ProductLine and orders them by ProductLine.
   ```
   Dynamo.SQL = 'Select ProductLineCode, ProductLine from ProductLine order by
   ProductLine';
   ```
   Note: You can drag folders and columns from the Metadata pane into the SQL pane to save you some typing.
3. Click OK.
   The Dynamo Manager dialog box reappears.
4. Click Edit and then click Definition.
   The Dynamo Definition dialog box appears.

5. Type the JavaScript code you want.

   The following example uses ProductLineCode as the HTML output values in a drop-down list, but displays the ProductLine.

```
var sOutput = '<p><select name="ProdLine">';

while (!rs.EOF)
{
        sOutput = sOutput + "<option value='" + rs.Fields("ProductLineCode") + "'>" +
rs.Fields("ProductLine") + "</option>";
        rs.MoveNext();
}

sOutput += '</select></p>';
return sOutput;
```

   The rs.EOF statement tests for the end of the recordset. The whole statement, while (!rs.EOF), directs the statements within the braces to be executed while the end of the recordset hasn't been reached.

6. Click OK to close the Dynamo Definition dialog box.
7. Right-click your dynamo and then click Copy.
8. Close the Dynamo Manager dialog box.
9. Paste the dynamo into the <form> section of your HTML page and save the page.

Related Topics

- "Create a Dynamo Using the Wizard" (p. 19)
- "Dynamos: Overview" (p. 18)

Script: Overview

A script changes the report that is associated with the PowerPrompts application, based on the selections that the report consumers make. For PowerPrompts Series 7, script is JavaScript code. In PowerPrompts 6.0, script was the combination of PowerPrompts methods and Impromptu Report methods.

Script Editor Dialog Box

Use the Script Editor dialog box to create, edit, and delete scripts in your PowerPrompts application.

You cannot validate the script until you run the application. If there are script errors, they will be shown only in the Web browser.

Related Topics

- "Add a Script" (p. 21)
- "Create a New Application" (p. 12)
- "Create a PowerPrompts Application" (p. 7)

Add a Script

Description

You use the Script Editor dialog box to add a script. A script is JavaScript code that changes the report associated with the PowerPrompts application. For example, if a report consumer selects a country, a report filter can be created that shows only records for that country. For an example of a script, see *Discovering PowerPrompts*.

Steps

1. Click Script Editor.
2. Type the script containing the methods and constants you want and click OK.
   For example, the following script filters the report by using the value of Salescountrycode provided by the report consumer
   ```
   if (App.Variables("CountryCd"))
   {
           GetQuery().AndFilterBy("[\\Country\\Salescountrycode] = " +
   App.Variables("CountryCd"));
   }
   ```
   Tip: To insert Impromptu Report methods, and constants, right-click the Script Editor box and then click the item you want. To insert table and column references, open Impromptu and copy these from the Folders dialog box and then paste them into the Script Editor dialog box.
3. Close the Script Editor dialog box.

When the report consumer makes a selection that sets this value, your script is executed before the report opens.

Related Topics

- "Script: Overview" (p. 21)

Test and Deploy Your Application: Overview

To test your PowerPrompts application, select an Impromptu report to test it with. Then you verify and test the application as well as fix any script errors. When you have the application working, you can distribute it.

Related Topics:

- "Fix Script Errors" (p. 24)
- "Notify Your Impromptu Web Reports Administrator" (p. 25)
- "Select a Report to Test Your Application" (p. 22)
- "Test the Application" (p. 23)
- "Verify the Application" (p. 23)
- "View the Generated Script" (p. 24)

Select a Report to Test Your Application

Description

A PowerPrompts application can work with different reports as long as the scripts you created apply to the report. To test the scripts in an application, you must specify a report.

Steps

1. From the File menu, click Properties.
   The General tab of the Application Properties dialog box appears.
2. Click Browse, select the report, and then click Open.
   The report you selected appears in the Report to Run box.
3. Click OK.

Related Topics

- "Fix Script Errors" (p. 24)
- "Notify Your Impromptu Web Reports Administrator" (p. 25)
- "Test the Application" (p. 23)
- "Verify the Application" (p. 23)

Verify the Application

Description

Each time you save an application, the PowerPrompts Developer Studio verifies the application and shows you errors and warnings. However, you can also use the Verify Application command to check the path and links in your application without saving your application.

Note: To open a help topic about an application error, right-click the error and then click Show Help.

Steps

1. Open the application to verify.
2. From the File menu, click Verify.
   The Verify Application dialog box appears showing a list of any navigational errors, missing HTML pages, and redundant links.
   Note: You can keep the Verify Application dialog box open while you fix the errors on the list.
3. Fix your application errors and verify again.

After all application errors have been fixed, a message appears that tells you the application is deployable.

Related Topics

- "Fix Script Errors" (p. 24)
- "Notify Your Impromptu Web Reports Administrator" (p. 25)
- "Select a Report to Test Your Application" (p. 22)
- "Test the Application" (p. 23)
- "View the Generated Script" (p. 24)

Test the Application

Description

After you complete your application and check that there are no application errors, you can test your PowerPrompts application. The application and the resulting report appear in your Web browser as if you were a report consumer.

To test an application, you must have the following software installed on your computer:

- a Web server
- a Web browser
- a PDF viewer, such as Adobe Acrobat Reader

You must also specify your catalog and connection information. You might have to type your User Class, User Class Password, Database User Id, or Database Password depending on what security source you are using.

Why Would PowerPrompts Not Be Able to Connect to a Catalog?

If the current catalog logon information provided is incorrect, is incomplete or hasn't been specified yet.

Steps

1. Start your Web server.
2. Click Run.
   The application Entry page appears in your Web browser.
3. Follow the instructions that you created in your application and specify the information you want to see in the report.

4. On the final page, click Run Report.
   If there were no script errors and PowerPrompts successfully connects to the catalog, your report opens in .pdf format in your Web browser.

Related Topics

- "Fix Script Errors" (p. 24)
- "Notify Your Impromptu Web Reports Administrator" (p. 25)
- "Select a Report to Test Your Application" (p. 22)
- "Verify the Application" (p. 23)
- "View the Generated Script" (p. 24)

Fix Script Errors

Description

When you test your application, it may have script errors. These are shown in the Web browser after the application is run with a report.

Tip: Check scripts for matching quotation marks. Also, check the spelling of Impromptu catalog folder and column names. For more information, see "Impromptu Expressions in PowerPrompts" (p. 94).

Steps

1. Run the PowerPrompts application.
2. In the Web browser, read any error descriptions.
3. Fix the error and run the application again.

Related Topics

- "Select a Report to Test Your Application" (p. 22)
- "Notify Your Impromptu Web Reports Administrator" (p. 25)
- "Test the Application" (p. 23)
- "Verify the Application" (p. 23)
- "View the Generated Script" (p. 24)

View the Generated Script

Description

You can also generate a script to be applied to the report given the selections that you made while running the application. This provides another way to test your PowerPrompts application and to debug your script.

Steps

1. From the File menu, click Properties.
2. Clear the Report to Run box, and click OK.
   That report is no longer associated with the PowerPrompts application.
3. From the File menu, click Save.
   The Verify Application dialog box appears and tells you that there is no report to run with your application.
4. Click Save.
5. Click Run.
6. Select the information that you want in the report.
7. On the Exit page, click Run Report.

A script with the values that you selected appears in your Web browser. If you included comments for the scripts, they are also included in this list.

Tips

- To not see the Verify Application dialog box, from the Tools menu, click Options, and clear the Ignore Warnings While Verifying Application check box.
- To not be prompted to save changes before you test your PowerPrompts application, from the Tools menu, click Options, and click Save Changes.

Related Topics

- "Fix Script Errors" (p. 24)
- "Notify Your Impromptu Web Reports Administrator" (p. 25)
- "Select a Report to Test Your Application" (p. 22)
- "Test the Application" (p. 23)
- "Verify the Application" (p. 23)

Notify Your Impromptu Web Reports Administrator

When you complete and test your PowerPrompts application, notify the Impromptu Web Reports administrator. When report consumers open a report with a PowerPrompts application associated with it, the application runs in their Web browser, and they are prompted about what they want to see in the report.

For more information about what the report administrator must know after you complete the PowerPrompts application, see the Impromptu Web Reports *Report Administrator's Guide*.

Related Topics

- "Fix Script Errors" (p. 24)
- "Select a Report to Test Your Application" (p. 22)
- "Test the Application" (p. 23)
- "Verify the Application" (p. 23)
- "View the Generated Script" (p. 24)

Chapter 2: How To Topics

Filter a Report for Multiple Values

Description

You can filter a report for more than one value. For example, you may want to see the sales figures for Canada and USA combined.

Steps

1. Add the <select> HTML control with the multiple attribute to your page or into a dynamo, for example,

   ```
   <select name="Countries" multiple>...</select>
   ```

2. Write the script to add the filter using GetQuery().AndFilterBy() method. For non-string data, the script is easy, for example,

   ```
   GetQuery().AndFilterBy("[Countrycd] in (" + App.Variables("Countrycd") + ")");
   ```

   Note: Use "in" instead of "=" in the script because "in" handles multiple values.

   If the data is a string, enclose each individual value in single quotes. To do this, use the Join property of the StringList object, for example,

   ```
   GetQuery().AndFilterBy("[City] in (’" +  App.Variables("Cities").Join("','") +
   "')");
   ```

Example

This example adds a drop-down list for countries on a page and then filters the report by the countries you selected.

HTML Page

```
<select name="CountryCd" multiple>
<%
var rs = new Recordset();
rs.Open("Select SalesCountryCode,Country from Country", "GOS");
while(!rs.EOF)
{
    Response.Write("<option value='" + rs.Fields("SalesCountryCode") + "'>" +
rs.Fields("Country") + "</option>");
    Response.Write("<br>");
    rs.MoveNext();
}
rs.Close();
%>
</select>
```

Script

```
if (App.Variables("Countries"))
{
    GetQuery().AndFilterBy("[\\Country\\Salescountrycode] in (" +
App.Variables("CountryCd") + ")");
}
```

Related Topics

- "Add Impromptu User Class Filter to Picklists" (p. 29)
- "Create a Cascading Picklist on a Single Page" (p. 30)
- "Create a Summary of User Choices" (p. 28)
- "Create a Type-in Picklist" (p. 29)

Save User Choices Between Sessions

Description

You can save the choices users make and reuse them when they run the application again.

Save users choices by writing them to a database using the PowerPrompts Connection object or ADO if running under Windows.

Steps

1. Write the App.Variables values into the database at the top of each HTML page in your application. A simple database would be a table with 3 columns.
2. Each time a page is generated, execute a select query to determine if any values exist in the database for prompts on this page. If they do, initialize the controls to the correct values.

Example

This example creates a simple database that stores three values: 1) user name, 2) variable name and 3) value.

```
<%
var conn = new Connection();
for (var sName in App.Variables)
{
    conn.Execute("INSERT INTO UserState (UserName, VariableName, Value) VALUES ( '"+
User.UserName + "','" + sName + "','" + App.Variables(sName) + "')");
}
%>
```

Related Topics

- "Create a Different Look for Different Upfront Themes" (p. 32)
- "Create a Summary of User Choices" (p. 28)

Create a Summary of User Choices

Description

You can create a summary of choices that the user has already made.

Steps

- On the Exit page of the PowerPrompts application, add JavaScript that shows the variables that the user set. The following code shows, in an HTML table, the country that a user selected to view,

```
<br><b>Table of Variables Selected</b>
<table>
<table border=2 bgcolor=#80FFFF>
<tr width="15%">
<th>Variable Name</th>
<th>Value</th>
</tr>
<%
if (App.Variables("CountryCd"))
{
    var rs = new Recordset;
    rs.Open("SELECT Country FROM Country WHERE SalesCountryCode=" +
App.Variables("CountryCd"),"GOS");
    if (!rs.EOF)
    {
        Response.Write("<tr><td>Country</td><td>" + rs.Fields("Country") +
"</td></tr>");
    }
}
...
```

Note: Chapter 2 of the Discovering PowerPrompts has a complete JavaScript example of showing user choices. The code for that example is also in the pwp_html.txt file which is located the *installation_location*\Samples\PowerPrompts folder.

Related Topics

- "Create a Different Look for Different Upfront Themes" (p. 32)
- "Save User Choices Between Sessions" (p. 28)

Add Impromptu User Class Filter to Picklists

Description

Your catalogs may contain existing user class filters. You can use this information in your PowerPrompts applications. For example, if a catalog had a user class for each country, you can filter the data in each report based on which user class the current user is a member of. Therefore when creating a PowerPrompts application for a report from this catalog, you may want a drop-down list of cities that includes only those cities for their user class. You can implement this with the Query object.

Steps

- Use the Query object to send SQL to the Recordset object with the appropriate filter based on the current user class, as in the following example.

Example

This example creates a drop-down list of the cities based on the user class filter.

```
<%
var myQuery = new Query();
myQuery.AddColumn("[\\Country\\City]","City");
var sql = myQuery.SQL;
var rs = new Recordset();
rs.Open(sql,"GOSALES");
Response.Write("<select name=city>");
while (!rs.EOF)
{
    Response.Write("<option>" + rs.Fields(0) + "</option>");
    rs.MoveNext();
}
Response.Write("</select>");
%>
```

Related Topics

- "Create a Cascading Picklist on a Single Page" (p. 30)
- "Create a Type-in Picklist" (p. 29)
- "Filter a Report for Multiple Values" (p. 27)

Create a Type-in Picklist

Description

You can create a type in picklist by adding a text input control to an HTML page. After PowerPrompts retrieves the value from the input control, the value can either be used in the final script or in an Impromptu expression.

Steps

1. You can create a type in picklist by adding a text input control to an HTML page.
2. Retrieve the value from the input control, the value can either be used in the final script or in an Impromptu expression.

Example

This example filters the final report by order code which is typed in by the consumer.

HTML Page

```
<html>
<body>
<h1>Enter Order Code</h1>
Please enter an order code.<p>
<form method="post">
Order Code<br>
<input type="text" name="order">
<input type="submit" name="Enter">
</form>
</body>
</html>
```

Script

```
GetQuery().AndFilterBy("[Order Code] = " + App.Variables("order"));
```

Related Topics

- "Add Impromptu User Class Filter to Picklists" (p. 29)
- "Create a Cascading Picklist on a Single Page" (p. 30)
- "Filter a Report for Multiple Values" (p. 27)

Create a Cascading Picklist on a Single Page

Description

You can filter a picklist by your selection from another picklist, all on the same page. For example, you could filter a city picklist by first selecting a country.

Steps

1. Write the client-side JavaScript to create the two picklists.
2. Add two links from the picklist page.
3. Specify that the first link has the condition, btn=Next, and leads to the next page in the application.
4. Specify that the default link goes to its own page. This a self-referencing link.
   In the application, making a selection in the country select submits the page and generates the correct list of cities. Pressing the Next button causes the next page to be shown.

Example

This example filters a list of cities by your selection in a list of countries.

HTML Page

```
<html>
<body>
<h1>Cascading Picklist</h1>
<form method="post">

Country<br>
<select name="country" OnChange="document.forms[0].submit()">
<%
// Fetch the list of countries
var rs = new Recordset();
rs.Open("SELECT country FROM country ORDER BY country","GOS");
```

```
// Write out each country in the list
var sSelectedText;
var sPreviousCountry = App.Variable("country");
while (!rs.EOF)
{
    // This code preselects the previously chosen country
    if (sPreviousCountry == rs.Fields(0))
    {
        sSelectedText = " selected ";
    }
    else
    {
        sSelectedText = "";
    }

    Response.Writeln("<option " + sSelectedText + " >" + rs.Fields(0));
    rs.MoveNext();
}
%>
</select>
<p>
City<br>
<select name="city">
<%
// Fetch the list of cities filtering by country
var rs2 = new Recordset();
rs2.Open("SELECT city FROM country WHERE country = '" + App.Variables("country") + "'
ORDER BY city","GOS");
while (!rs2.EOF)
{
    Response.Writeln("<option>" + rs2.Fields(0));
    rs2.MoveNext();
}
%>
</select>
<p>
<input type=submit name=btn value=Next>
</form>
</body>
</html>
```

Related Topics

- "Add Impromptu User Class Filter to Picklists" (p. 29)
- "Create a Type-in Picklist" (p. 29)
- "Filter a Report for Multiple Values" (p. 27)

Using ADO from PowerPrompts

Description

You can use Microsoft ActiveX Data Objects (ADO) in PowerPrompts applications to access an ODBC data source. Use the Server.CreateObject method to instantiate a COM object, and then use its properties and methods.

Note: JavaScript does not support the concept of default properties and methods so, you must explicitly call each ADO method.

Steps

1. Write the code to create an ADO object.
2. Add that code to one of your HTML pages, save the change, and run the application.

Example

This example creates an ADO connection object, connects to an ODBC data source, populates a record set, and then writes the first field to your HTML page.

```
<%
var oCmd = Server.CreateObject("ADODB.Command");
oCmd.ActiveConnection = "DB_GREATOUTDOORS";
oCmd.CommandText = "SELECT * FROM Country";
var oRS = oCmd.Execute();
while (!oRS.EOF)
{
    Response.Writeln(oRS.Fields("Country").Value + "<br>");
    oRS.MoveNext();
}
%>
```

Retrieve Database User ID and Password from Access Manager

Description

You need not add database Id and Password information directly into your PowerPrompts application. You can retrieve these from Access Manager. To do this, you must first set up Access Manager correctly. Ensure that each user who will be using the PowerPrompts application has access to the required database and has an appropriate sign on for each database. Please see the Access Manager documentation for information on how to do this.

After doing that, do not supply a user id and password when calling the Recordset.Open method.

Steps

1. Set up Access Manager.
2. Call the Recordset.Open method without supplying a user id or password.
3. When creating dynamos, clear the "Use this Authentication Information" checkbox to retrieve user id and passwords from Access Manager.

Example

This example asks Access Manager if there is a database sign on for the current user. If one is found, it will be used.

```
rs.Open("SELECT * FROM COUNTRY","Database");
```

Create a Different Look for Different Upfront Themes

Description

Upfront supports different themes, each theme can have a different look. Your consumers can select the look they want. By default, PowerPrompt applications have the same look for each theme. However, you can create your applications so that they look different based on the theme selected by the consumer. To do this, access to the Upfront.Theme property and change the page based on this property.

Steps

- Write code that changes your HTML pages depending on which Upfront theme the user selects.

Example

This example creates two themes called "Corporate" and "Flashy".

HTML Page

```
<html>
<head>
<%
    switch (Upfront.Theme)
    {
        case "Corporate":
%>
    <link rel="stylesheet" type="text/css" href="/corp.css">
<%
        break;

        case "Flashy":
%>
    <link rel="stylesheet" type="text/css" href="/flashy.css">
<%
        break;
    }
%>
</head>
<body>
<%
    switch (Upfront.Theme)
    {
        case "Corporate":
%>
    <h1>Corporate Theme</h1>
    This is the corporate theme with the corporate look.
<%
        break;

        case "Flashy":
%>
    <h1>Flashy Theme</h1>
    This is the flashy theme with the flashy look.
<%
        break;
    }
%>

</body>
</html>
```

Create a Single PowerPrompts Application for Multiple Languages

You can author your PowerPrompts application to appear with text in the appropriate language for the consumer.

1. On your HTML page, instead of putting text on the page, create a variable in JavaScript for each word or groups of words.
2. Use the Response.Write method to output the text of the variable wherever you want the text to appear on the page.
3. At the top of the page, query Upfront for the currently selected language, and set each variable to the appropriate text.

Example

This example returns text based on the language the consumer selects.

HTML Page

```
<html>
<head>
<%
```

```
// Get the language and save it
var slang = Upfront.Language;
var sTitle;
var sCountry;
var SelectText;
if (sLang == "en")
{
    sTitle = "Welcome Page";
    sCountry = "Country";
    sSelectText = "Select a Country";
}
else if (slang == "fr")
{
    sTitle = "Page de Bienvenue";
    sCountry = "Pays";
    sSelectText = "Choisir un Pays";
}
else
{
    // By default, just use English
    sTitle = "Welcome Page";
    sCountry = "Country";
    sSelectText = "Select a Country";
}
%>
</head>
<body>
<h1><%=sTitle%></h1>
<%=sSelectText%><p>
<%=sCountry%>
<select name=Country>
</select>
</body>
</html>
```

Chapter 3: JavaScript in PowerPrompts

You can use client-side and server-side JavaScript in your HTML pages.

Client-side JavaScript is executed by the browser. See your client-side JavaScript documentation for information about its syntax rules. Server-side JavaScript is executed by PowerPrompts and use to generate the contents of a page as well as access PowerPrompts and Impromptu Web Reports objects.

Described below are some rules that should help you write server-side JavaScript code.

Server-Side JavaScript

- Server-side JavaScript is fully ECMA-262 compliant.
- You must enclose all server-side JavaScript in angle brackets (<%...%>) to have the PowerPrompts server recognize it.
- Comments in your server-side JavaScript must be indicated by the C++ (//) or C (/*...*/) style comments.

  For example,

```
<%
/* Execute a dynamo that returns a number and assign it to a string */
var sReturn = App.RunDynamo("MyDynamo2");
if (sReturn > 0)
{
    Response.Write("<b>Returned Orders:" + sReturn +"</b>");
}
%>
```

- JavaScript is case-sensitive. Response.Write is correct while RESPONSE.write is not.
- An equal sign (=) enclosed in the angle brackets (<%...%>) is shorthand for Response.Write.

  For example,

```
<%=Request.Variables%>
```

  is the same as

```
<%Response.Write(Request.Variables)%>
```

Master Table of Server-Side JavaScript Objects

Use the following server-side objects to generate the contents of a page as well as interact with PowerPrompts and Impromptu Web Reports components. For more information on server-side JavaScript, see "JavaScript in PowerPrompts" (p. 35).

Object	Properties	Methods
"App Object" (p. 39)	"BackURL Property" (p. 39) "CurrentPage Property" (p. 40) "Errors Property" (p. 40) "FinalURL Property" (p. 41) "IsTestMode Property" (p. 41)	"RunDynamo Method" (p. 43)

Object	Properties	Methods
	"Path Property" (p. 42)	
	"ReportScript Property" (p. 42)	
	"Variables Property" (p. 42)	
"Connection Object" (p. 43)		"Execute Method" (p. 44)
"Field Object" (p. 44)	"HTMLEncodedValue Property" (p. 45)	"Operator() Method (Field)" (p. 49)
	"Index Property" (p. 45)	
	"Name Property" (p. 46)	
	"Type Property" (p. 47)	
	"Value Property" (p. 48)	
"Query Object" (p. 49)	"SQL Property" (p. 50)	"AddColumn Method" (p. 51)
		"AndFilterBy Method" (p. 51)
		"AndSummaryFilterBy Method" (p. 52)
		"AssociateColumn Method" (p. 52)
		"GroupBy Method" (p. 53)
		"OrFilterBy Method" (p. 53)
		"OrSummaryFilterBy Method" (p. 54)
		"RemoveColumn Method" (p. 54)
		"SetDistinct Method" (p. 54)
		"SetPromptValue Method" (p. 55)
		"SortBy Method" (p. 55)
"Recordset Object" (p. 56)	"CurrentRecordIndex Property" (p. 56)	"Close Method" (p. 59)
	"EOF Property" (p. 57)	"MoveNext Method" (p. 60)
	"Fields Property" (p. 58)	"Open Method" (p. 60)
	"MaxRecords Property" (p. 58)	
"Request Object" (p. 62)	"Cookies Property" (p. 62)	

Object	Properties	Methods
	"ServerVariables Property" (p. 62)	
	"Variables Property" (p. 42)	
"Response Object" (p. 64)	"ContentType Property" (p. 64)	"AppendCookie Method" (p. 65)
		"AppendHeader Method" (p. 65)
		"Clear Method" (p. 65)
		"ClearContent Method" (p. 66)
		"ClearHeaders Method" (p. 66)
		"Redirect Method" (p. 66)
		"Write Method" (p. 66)
		"WriteFile Method" (p. 67)
		"Writeln Method" (p. 67)
"Server Object" (p. 69)	"ScriptTimeout Property" (p. 69)	"CreateObject Method" (p. 70)
		"FormatNumber Method" (p. 70)
		"HTMLEncode Method" (p. 71)
		"URLDecode Method" (p. 71)
		"URLEncode Method" (p. 72)
"StringList Object" (p. 72)	"Count Property" (p. 73)	"Contains Method" (p. 73)
		"Item Method" (p. 74)
		"Join Method" (p. 75)
		"Operator() Method (StringList)" (p. 75)
		"toString Method" (p. 76)
"Upfront Object" (p. 76)	"Language Property" (p. 76)	"ExecuteCommand Method" (p. 78)
	"Locale Property" (p. 77)	"GetPageFragment Method" (p. 78)
	"Theme Property" (p. 77)	
"User Object" (p. 78)	"Description Property" (p. 79)	
	"Email Property" (p. 79)	
	"Telephone Property" (p. 80)	

Object	Properties	Methods
	"UserClass Property" (p. 80)	
	"UserClasses Property" (p. 80)	
	"UserName Property" (p. 81)	

#include Statement and Other Preprocessor Directives

#include statement

Use the #include statement to include other JavaScript code onto the page. You can use this statement to share code between pages in your application or even between applications. The path specified is always relative to the application directory. In Example1 Use the relative path when you want a common location for shared JavaScript files on a server.

Example1

This example shows using the #include statement when the SaveState.js is in the same folder as the application.

```
<%
#include "SaveState.js"
%>
```

Example2

This example shows using the #include statement when the SaveState.js in a CommonScripts folder that is one folder below the application.

```
<%
#include "../CommonScripts/SaveState.js"
%>
```

Preprocessor Directives

In addition to the #include statement, you can use other preprocessor directives

- #define
- #ifdef
- #else
- #endif

The #define statement is used to declare global variables, for example:

```
#define COMPANY_NAME Cognos
```

The other three are used to create conditional define statements.

Example

This example shows a conditional define that checks for UNIX.

```
#ifdef _UNIX_
    Response.Write("Not supported on UNIX");
#else
    var ocom = Server.CreateObject("ADODB.Connection");
#endif
```

Pre-Defined Constants

There are two pre-defined constants that you can use to help test applications

- TEST_MODE
- _UNIX_

TEST_MODE is defined if you are testing PowerPrompts application using Impromptu client. _UNIX_ is defined if a UNIX server tries to run a PowerPrompts application.

App Object

Returns the current user's state in the PowerPrompts application.

A non-creatable object.

Property	Description
BackURL Property	Returns the URL to Upfront. Allows you to return to Upfront from your PowerPrompts application. Use this property as part of an action on a form.
CurrentPage Property	Returns the name of the current page.
Errors Property	Returns a list of error messages for the current server request.
FinalURL Property	Returns the URL to Upfront after exiting a PowerPrompts application. Use this method in the Exit page as part of an action on a form.
IsTestMode Property	Returns whether the PowerPrompts application is being run in test mode. Returns true if the application is being run from PowerPrompts Developer Studio and false in all other cases. Use when debugging your application.
Path Property	Returns the folder of the PowerPrompts application file (.xxm). This is a useful property for the Response.WriteFile method.
ReportScript Property	Returns the script that will be sent to Impromptu. This property is available only in test mode and only on the exit page. In all other circumstances, it will return an empty string. Use when debugging your application.
Variables Property	Returns the state of the application and is a collection of variables.

Method	Description
RunDynamo Method	Runs the specified dynamo to generate a string. Use this method in your HTML pages.

BackURL Property

Returns the URL to Upfront. Allows you to return to Upfront from your PowerPrompts application. This property works only from deployed applications running under Impromptu Web Reports. This property doesn’t work in test mode since there is no Upfront to return to.

Use this property as part of an action on a form.

This property is read-only.

Type

String

Example

This example returns the user to Upfront when they click the Cancel button.

```
<form method="post" action="<%=App.BackURL%>">
<input type="submit" value="Cancel">
</form>
```

Applies To

App Object

CurrentPage Property

Returns the name of the current page.

This property is read-only.

Type

String

Example

This example shows the page name in a Head 1.

```
<!--Use the page name as the header-->
<h1><%=App.CurrentPage%></h1>
```

Sample Output

```
SelectCountry
```

Applies To

App Object

Errors Property

Returns a list of error messages for the current server request.

This property is read-only and works only on the Error Page.

Type

StringList

Example

```
This example shows each error separated by a horizontal line.
<%
// Only works on the Error page.
for ( var i = 0; i < App.Errors.Count; i++ )
{
    if ( i > 0 )
    {
        Response.Writeln( "<hr>" );
    }
Response.Write(<p><pre>Server.HTMLEncode( App.Errors( i ) )</pre></p>)
}
%>
```

Sample Output

```
Error on page [SelectCountry] line 10

<%=App.RunDynamo("Countries")%>
----------------------------------------------------------------
Method call [App.RunDynamo] failed

Error near no filename:10 [RunDynamo()].
      from no filename:10 [:Global Initialization:()]
----------------------------------------------------------------
Error in Dynamo [Countries] on line 5 while executing the definition.
----------------------------------------------------------------
TypeError 1406: Variable is not a function type.
Error near no filename:5 [Global Code].
```

Applies To

App Object

FinalURL Property

Returns the URL to Upfront after exiting a PowerPrompts application. Use this method in the Exit page as part of an action on a form.

This property is read-only.

Type

String

Example

This example creates a button that runs the report.

```
<form action="<%=App.FinalURL%>">
<input type = "submit" value = "Finish">
</form>
```

Applies To

App Object

IsTestMode Property

Returns whether the PowerPrompts application is being run in test mode. Returns true if the application is being run from PowerPrompts Developer Studio and false in all other cases. Use when debugging your application.

This property is read-only.

Type

Boolean

Example

This example creates an empty text box if the application is being tested.

```
<!--Only display the report script if the application is being tested-->
<%
if (App.IsTestMode);
{
%>
<textarea>
<%=Server.HTMLEncode(App.ReportScript)%>
</textarea>
<%
}
%>
```

Applies To

App Object

Path Property

Returns the folder of the PowerPrompts application file (.xxm). This path includes the trailing path separator. The path separator is either a forward or backward slash, whichever is appropriate for the operating system that PowerPrompts is running in. This is a useful property for the Response.WriteFile method.

This property is read-only.

Type

String

Example

This example shows the path to the application.

```
<%Response.Write(App.Path);%>
```

Sample Output

```
C:\PowerPrompts Apps\SalesRep\
```

Applies To

App Object

ReportScript Property

Returns the script that will be sent to Impromptu. This property is available only in test mode and only on the exit page. In all other circumstances, it will return an empty string. Use when debugging your application.

This property is read-only.

Type

String

Example

This example creates a text box which shows the script.

```
<!--Display the report script in a text area-->
<textarea>
<%=Server.HTMLEncode( App.ReportScript )%>
</textarea>
```

Applies To

App Object

Variables Property

Returns the state of the application and is a collection of variables. Application variables persist between requests. Request variables do not. Use this property to retrieve variables set in the application. Use the Request.Variables property only if you require the unprocessed form name/value pairs.

This property is read-only.

Type

StringList

Example

This example returns the CountryCd for the country you select on the SelectCountry page.

```
<%
// Output a text control with the text of CountryCd if it has a value, otherwise
// put the text Default
if (App.Variables("CountryCd"))
{
    Response.Writeln("<input type='text' name='text1' value='" +
App.Variables("CountryCd") + "'");
}
else
{
    Response.Writeln("<input type='text' name='text1' value='Default'>");
}
%>
```

Sample Output

```
16
```

Applies To

App Object

RunDynamo Method

Runs the specified dynamo to generate a string. Use this method in your HTML pages. This cannot be used method in the Dynamo Definition or Dynamo SQL dialog boxes.

Parameters

DynamoName: String, Mandatory

Return Type

String

Example1

This example executes MyDynamo1 and writes its output in the page.

```
<!--Run a dynamo that was created with the dynamo wizard-->
<%=App.RunDynamo("MyDynamo1")%>
```

Example2

This example executes MyDynamo2 and if the value it returns is greater than zero, a message is written out.

```
<%
var iReturn = parseInt(App.RunDynamo("MyDynamo2"));

if (iReturn > 0 )
{
    Response.Write("Return Orders: " + iReturn);
}
%>
```

Sample Output

```
Returned Orders: 4
```

Applies To

App Object

Connection Object

An object that changes your database using SQL statements. This allows you to connect to a data source and execute SQL without returning a result set.

A creatable object.

Method	Description
Execute Method	Executes the SQL statement that changes your database.

Execute Method

Executes the SQL statement that changes your database.

If this method fails, the error page is shown.

Parameters

The SQL parameter is the SQL to be executed. The dbName parameter is the name of the Cognos logical database.

There are two optional string parameters. The dbUserID parameter is the database user ID to use when connecting to the database. The dbPassword is the database password for this user ID.

SQL: String, Mandatory

dbName: String, Mandatory

dbUserID: String, Optional

dbPassword: String, Optional

Example

This example does adds the values John Smith and Programmer to the database.

```
<%
var conn = new Connection();
conn.Execute("INSERT INTO User(UserName,UserDescription) VALUES('John
Smith','Programmer')", "User");
Response.Write("<b>Query successful</b>");
%>
```

Applies To

Connection Object

Field Object

A Recordset Object has a Fields collection consisting of Field objects. Each Field Object corresponds to one column of data in the Recordset.

A non-creatable object.

Property	Description
HTMLEncodedValue Property	Returns the value of the column in the result set for the current row, except that it is HTML encoded.
Index Property	Returns the position of the column in the result set. This is a zero-based value.
Name Property	Returns the name of the column in the result set.
Type Property	Returns the data type of the field.
Value Property	Returns the value of the column in the result set for the current row.

Method	Description
Operator() Method (Field)	Specifies the index operator used to access items in the collection. You can specify either a numeric index or a string key.

HTMLEncodedValue Property

Returns the value of the column in the result set for the current row, except that it is HTML encoded. The equivalent JavaScript is

```
Server.HTMLEncode(rs.Fields(0).Value)
```

This is the default property of the Field object. This property is string type regardless of the original database type. To use as a numeric, use any of the standard JavaScript methods for converting string type to numeric. If the value in the database is NULL, then the JavaScript null is returned.

This property is read-only.

Type

String

Example1

```
<%=rs.Fields(0).HTMLEncodedValue%>
```

Example2

```
<%=rs.Fields(0)%>
```

Applies To

Field Object

Index Property

Returns the position of the column in the result set. This is a zero-based value which means that the first column has an index of 0, the second column an index of 1, and so on. For example, the SQL "SELECT Country, SalesCountryCode FROM Country" has two fields. Country has an index of 0, and SalesCountryCode, an index of 1.

This property is read-only.

Type

Number

Example

```
<%
// Output the value of the column, plus its name and index
var rs = new Recordset();
rs.Open("Select Country,SalesCountryCode from Country","GOS");
// Only output a table if the result set is not empty
if (!rs.EOF)
{
    Response.Writeln("<table><tr>");
    // Write out the table headers
    for (var i = 0; i < rs.Fields.Count; i++)
    {
        Response.Write("<td>" + rs.Fields(i).Name + " Index: " + rs.Fields(i).Index +
"</td>");
    }
    Response.Writeln("</tr>");
    while (!rs.EOF)
    {
        Response.Write("<tr>");
        for (var j = 0; j < rs.Fields.Count; j++)
        {
            Response.Write("<td>" + rs.Fields(j).Value + "</td>");
            rs.MoveNext();
        }
        Response.Writeln("</tr>");
    }
    Response.Writeln("</table>");
}
else
{
    Response.Writeln("No data.<p>");
}
%>
```

Sample Output

```
Country Index: 0 SalesCountryCode Index: 1
France 2
United States 4
Austria 6
Netherlands 8
England 10
Japan 12
Korea 14
Australia 17
Denmark 19
Mexico 21
Finland 23
```

Applies To

Field Object

Name Property

Returns the name of the column in the result set. For example, the SQL, "SELECT Country, Country_Cd AS Code FROM Country" you has two fields. The first one is named Country, and the second one is named Code.

This property is read-only.

Type

String

Example

```
<%
// Output the value of the column, plus its name and index
var rs = new Recordset();
rs.Open("Select Country,SalesCountryCode from Country","GOS");
// Only output a table if the result set is not empty
if (!rs.EOF)
{
    Response.Writeln("<table><tr>");
    // Write out the table headers
    for (var i = 0; i < rs.Fields.Count; i++)
    {
        Response.Write("<td>" + rs.Fields(i).Name + " Index: " + rs.Fields(i).Index +
"</td>");
    }
    Response.Writeln("</tr>");
    while (!rs.EOF)
    {
        Response.Write("<tr>");
        for (var j = 0; j < rs.Fields.Count; j++)
        {
            Response.Write("<td>" + rs.Fields(j).Value + "</td>");
            rs.MoveNext();
        }
        Response.Writeln("</tr>");
    }
    Response.Writeln("</table>");
}
else
{
    Response.Writeln("No data.<p>");
}
%>
```

Sample Output

```
Country Index: 0 SalesCountryCode Index: 1
France 2
United States 4
Austria 6
Netherlands 8
England 10
Japan 12
Korea 14
Australia 17
Denmark 19
Mexico 21
Finland 23
```

Applies To

Field Object

Type Property

Returns the data type of the field. The following numeric constants can be used:

- ftDate
- ftDateTime
- ftInterval
- ftNumeric
- ftString
- ftTime
- ftUnknown

This property is read-only.

Type

Number

Example

```
<%
// Output the value of the column, plus its name and index
var rs = new Recordset();
rs.Open("Select * from Country","GOS");
// Output the names of the columns and their data types
for (var i = 0; i < rs.Fields.Count; i++)
{
var sText = rs.Fields(i).Name + ": ";
    switch(rs.Fields(i).Type)
    {
        case ftDate:
            sText += "Date";
            break;
        case ftDateTime:
            sText += "Date/Time";
            break;
        case ftInterval:
            sText += "Interval";
            break;
        case ftNumeric:
            sText += "Numeric";
            break;
        case ftString:
            sText += "String";
            break;
        case ftTime:
            sText += "Time";
            break;
        case ftUnknown:
            sText += "Unknown";
            break;
    }
Response.Writeln(sText + "<br>");
}
%>
```

Sample Output

```
SalesCountryCode: Numeric
Country: String
ISOThreeLetterCode: String
ISOTwoLetterCode: String
ISOThreeDigitCode: String
CurrencyName: String
EuroInUseSince: Date/Time
```

Applies To

Field Object

Value Property

Returns the value of the column in the result set for the current row. If the value in the database is NULL, the JavaScript null is returned.

This property is string type regardless of the original database type. To use as a numeric, use any of the standard JavaScript methods for converting string type to numeric.

This property is read-only.

Type

String

Example

```
<%
// Write out the field value or the string "NULL" if the field has no value
var rs = new Recordset();
rs.Open("Select SalesCountryCode from Country","GOS");
var sValue = rs.Fields( 0 ).Value;
Response.Write( ( sValue == null ) ? "NULL" : sValue );
%>
```

Sample Output

```
1
```

Applies To

Field Object

Operator() Method (Field)

Specifies the index operator used to access items in the collection. You can specify either a numeric index or a string key.

Parameters

Index: Numeric, Mandatory

or

Key: String, Mandatory

Return Type

An item in a collection

Example

```
<%
// Output the value of the column, plus its name and index
var rs = new Recordset();
rs.Open("Select * from Country","GOS");
// Show both ways to use the Operator() Method
Response.Write(rs.Fields(0) + " is the value of the first column in the recordset.");
Response.Write("<br>");
Response.Write(rs.Fields("Country") + " is the value of a column named Country in the
recordset.");
%>
```

Sample Output

```
1 is the value of the first column in the recordset.
France is the value of a column named Country in the recordset.
```

Applies To

Field Object

Query Object

Use to retrieve SQL for an Impromptu report. The SQL will contain any user class filters and database qualifiers defined in the catalog. The SQL can then be used in a dynamo or picklist to populate user controls. Any denied tables that you specify in the Impromptu catalog are ignored.

You can construct the Query object by using the full path to an Impromptu report. This means that the Query object returns the main query for that report. PowerPrompts cannot access sub-report queries. If you want to start with a blank query, do not pass any objects to the constructor statement for the query, for example

```
var myQuery = new Query()
```

A creatable object.

Property	Description
SQL Property	Returns the SQL for the Query object, including the Impromptu catalog filters, based on the user class of the current user.

Method	Description
AddColumn Method	Adds an Impromptu catalog column or calculation to the query.
AndFilterBy Method	Adds a detail filter expression to the query. If a detail filter already exists, this filter is appended to it using an AND.
AndSummaryFilterBy Method	Adds a summary filter expression to the query. If a summary filter already exists, this filter is appended to it using an AND.
AssociateColumn Method	Adds an association for a column in the query.
GroupBy Method	Adds grouping to the query for a specified column.
OrFilterBy Method	Adds a detail filter expression to the query. If a detail filter already exists, this filter is appended to it using an OR.
OrSummaryFilterBy Method	Adds a summary filter expression to the query. If a summary filter already exists, this filter is appended to it using an OR.
RemoveColumn Method	Removes a column from the query and the report.
SetDistinct Method	Specifies that the query contains only distinct items.
SetPromptValue Method	Sets a value for a named prompt in the query.
SortBy Method	Sorts the query by the specified column, in addition to any existing sorting in the query.

SQL Property

Returns the SQL for the Query object, including the Impromptu catalog filters, based on the user class of the current user.

The returned SQL always begins with an exclamation mark (!). This mark tells the Recordset object that the SQL was generated by the Query object.

This property is read-only.

Type

String

Example

This example opens the Impromptu report pwp_tutorial_js.imr, which is located in the same folder as the PowerPrompts application, and then returns the SQL of that report to the Recordset object. Any user class filters are applied before the SQL is returned.

```
<%
var rs = new Recordset();
var myQuery = new Query("pwp_tutorial_js.imr");
var rs = new Recordset();
rs.Open(myQuery.SQL, "GOS");
%>
```

Applies To

Query Object

AddColumn Method

Adds an Impromptu catalog column or calculation to the query. For more information about creating Impromptu catalog expressions, see "Impromptu Expressions in PowerPrompts" (p. 94).

The Name parameter is the name of the column (or calculation) and must be unique within the query. The Expression parameter is the Impromptu catalog expression that defines the column.

Parameters

Name: String, Mandatory

Expression: String, Mandatory

Return Type

String

Example

This example adds a new column to the report before before returning the SQL.

```
<%
var rs = new Recordset();
var myQuery = new Query("pwp_tutorial_js.imr");
myQuery.AddColumn("Year","[year([Orders\\OrderDate])]");
rs.Open(myQuery.SQL, "GOS");
%>
```

Applies To

Query Object

AndFilterBy Method

Adds a detail filter expression to the query. If a detail filter already exists, this filter is appended to it using an AND. For more information about creating Impromptu catalog expressions, see "Impromptu Expressions in PowerPrompts" (p. 94).

The detailFilter parameter is the detail filter expression to be added.

Parameters

detailFilter: String, Mandatory

Return Type

String

Example

This example filters the report by year before returning the SQL.

```
<%
var rs = new Recordset();
var myQuery = new Query("pwp_tutorial_js.imr");
myQuery.AndFilterBy("year([Orders\\Orderdate]) in (1999,2000)");
rs.Open(myQuery.SQL, "GOS");
%>
```

Applies To

Query Object

AndSummaryFilterBy Method

Adds a summary filter expression to the query. If a summary filter already exists, this filter is appended to it using an AND. For more information about creating Impromptu catalog expressions, see "Impromptu Expressions in PowerPrompts" (p. 94).

The summaryFilter parameter is the summary filter expression to be added.

Parameters

summaryFilter: String, Mandatory

Return Type

String

Example

This example filters the report by year before returning the SQL.

```
<%
var rs = new Recordset();
var myQuery = new Query("pwp_tutorial_js.imr");
myQuery.AndSummaryFilterBy("year([Orders\\Orderdate]) in (1999,2000)");
rs.Open(myQuery.SQL, "GOS");
%>
```

Applies To

Query Object

AssociateColumn Method

Adds an association for a column in the query.

The namedColumn parameter is the name of the column to be associated. The groupColumn parameter is the name of the grouped column that the first column is to be associated with.

Parameters

namedColumn: String, Mandatory

groupColumn: String, Mandatory

Return Type

String

Example

This example associates the column Total Sales with the column Country before returning the SQL.

```
<%
var rs = new Recordset();
var myQuery = new Query("pwp_tutorial_js.imr");
myQuery.AssociateColumn("Total Sales","Country");
rs.Open(myQuery.SQL, "GOS");
%>
```

Applies To

Query Object

GroupBy Method

Adds grouping to the query for a specified column.

The groupColumn parameter is the name of the column to group. The sortDirection parameter is either "ascend" or "descend" to set how the group column is sorted.

This method appends a group to any existing groups in the report.

Parameters

groupColumn: String, Mandatory

sortDirection: String (ascend or descend), Mandatory

Return Type

String

Example

This example groups the column Vendorname before returning the SQL.

```
<%
var rs = new Recordset();
var myQuery = new Query("pwp_tutorial_js.imr");
myQuery.GroupBy("Vendorname","descend");
rs.Open(myQuery.SQL, "GOS");
%>
```

Applies To

Query Object

OrFilterBy Method

Adds a detail filter expression to the query. If a detail filter already exists, this filter is appended to it using an OR. For more information about creating Impromptu catalog expressions, see "Impromptu Expressions in PowerPrompts" (p. 94).

The detailFilter parameter is the detail filter expression to be added.

Parameters

detailFilter: String, Mandatory

Return Type

String

Example

This example adds a detail filter before returning the SQL.

```
<%
var rs = new Recordset();
var myQuery = new Query("pwp_tutorial_js.imr");
myQuery.OrFilterBy("[\\Products\\Product] startswith 'GO'");
rs.Open(myQuery.SQL, "GOS");
%>
```

Applies To

Query Object

OrSummaryFilterBy Method

Adds a summary filter expression to the query. If a summary filter already exists, this filter is appended to it using an OR. For more information about creating Impromptu catalog expressions, see "Impromptu Expressions in PowerPrompts" (p. 94).

The summaryFilter parameter is the summary filter expression to be added.

Parameters

summaryFilter: String, Mandatory

Return Type

String

Example

This example adds a summary filter before returning the SQL.

```
<%
var rs = new Recordset();
var myQuery = new Query("pwp_tutorial_js.imr");
myQuery.OrSummaryFilterBy("[\\Products\\Product] startswith 'GO'");
rs.Open(myQuery.SQL, "GOS");
%>
```

Applies To

Query Object

RemoveColumn Method

Removes a column from the query and the report.

The columnName parameter is the name of the column to remove from the query.

Parameters

columnName: String, Mandatory

Return Type

String

Example

This example removes the Vendorname column from the query before returning the SQL.

```
<%
var rs = new Recordset();
var myQuery= new Query("pwp_tutorial_js.imr");
myQuery.RemoveColumn("Vendorname");
rs.Open(myQuery.SQL, "GOS");
%>
```

Applies To

Query Object

SetDistinct Method

Specifies that the query contains only distinct items.

Parameters

onSwitch: Boolean, Mandatory

Return Type

String

Example

This example removes any duplicate values before returning the SQL.

```
<%
var rs = new Recordset();
var myQuery= new Query("pwp_tutorial_js.imr");
myQuery.SetDistinct("true");
rs.Open(myQuery.SQL, "GOS");
%>
```

Applies To

Query Object

SetPromptValue Method

Sets a value for a named prompt in the main query. This method works only if the query is created within an existing Impromptu report. All prompts must have values before you can retrieve the SQL for the query. To ensure that the Impromptu report accepts multiple prompt values, use an "In" operator instead of an equals (=) operator in the filter statement of that Impromptu report.

Tip: Use a comma to separate multiple values. If a comma is required as part of a value, precede that comma with the escape character. By default, a caret (^) is the escape character, but you can change it by updating the Separator Escape Character entry in the [Startup Options] section of your Impromptu.ini file.

Parameters

The promptName parameter is the name of the prompt within the query. Use the prompt name from the Available Prompts box of the Prompt Manager dialog box.

The promptValue parameter is the value you want the prompt to have.

promptName: String, Mandatory

promptValue: String, Mandatory

Return Type

String

Example

This example sets the value Canada for the Country prompt before returning the SQL.

```
<%
var rs = new Recordset();
var myQuery = new Query("pwp_tutorial_js.imr");
myQuery.SetPromptValue("Country", "Canada");
rs.Open(myQuery.SQL, "GOS");
%>
```

Applies To

Query Object

SortBy Method

Sorts the query by the specified column, in addition to any existing sorting in the query.

Parameters

The sortColumn parameter is the name of the column to sort. The sortDirection parameter is either "ascend" or "descend" to set how the sort column is sorted.

sortColumn: String, Mandatory

sortDirection: String (ascend or descend), Mandatory

Return Type

String

Example

This example sorts the query by the Producttype column before returning the SQL.

```
<%
var myQuery = new Query("pwp_tutorial_js.imr");
myQuery.SortBy("Producttype","ascend");
rs.Open(myQuery.SQL, "GOS");
%>
```

Applies To

Query Object

Recordset Object

Fetches data from Cognos logical databases. The Recordset object provides methods to connect to databases, execute SQL, and fetch back the data rows.

A creatable object.

Property	Description
CurrentRecordIndex Property	Returns the current record index which is zero based. This value is incremented after each call to the MoveNext method. This property is valid only if the EOF property is false.
EOF Property	Tests for the end of the Recordset.
Fields Property	A collection of field objects. Each field represents a column in the result set of the query.
MaxRecords Property	Sets or returns the maximum number of records to return.

Method	Description
Close Method	Closes an open Recordset.
MoveNext Method	Advances to the next row in the Recordset.
Open Method	Populates the Recordset with the rows returned from executing the SQL.

CurrentRecordIndex Property

Returns the current record index which is zero based. This value is incremented after each call to the MoveNext method. This property is only valid if the EOF Property is false.

This property is read-only.

Type

Integer

Example

This example writes out the name of each country on a separate row preceded by the row number.

```
<%
var rs = new Recordset();
rs.Open("Select Country from Country","GOS");
while (!rs.EOF)
{
    Response.Writeln("Row: " + rs.CurrentRecordIndex + " " + rs.Fields("country"));
    Response.Write("<br>");
    rs.MoveNext();
}
%>
```

Sample Output

```
Row: 0 France
Row: 1 Germany
Row: 2 United States
Row: 3 Canada
Row: 4 Austria
Row: 5 Italy
Row: 6 Netherlands
Row: 7 Switzerland
Row: 8 England
Row: 9 Sweden
Row: 10 Japan
Row: 11 Taiwan
Row: 12 Korea
Row: 13 China
Row: 14 Australia
Row: 15 Belgium
Row: 16 Denmark
Row: 17 Spain
Row: 18 Mexico
Row: 19 Brazil
Row: 20 Finland
Row: 21 Euroland
```

Applies To

Recordset Object

EOF Property

Tests for the end of the Recordset. EOF stands for End Of File. If the resulting query contains no rows, EOF is true.

This property is read-only.

Type

Boolean

Example

This example writes out the name of each country on a separate row.

```
<%
var rs = new Recordset();
rs.Open("Select SalesCountryCode,Country from Country", "GOS");
while(!rs.EOF)
{
    Response.Write("<option value='" + rs.Fields("SalesCountryCode") + "'>" +
rs.Fields("Country") + "</option>");
    Response.Write("<br>");
    rs.MoveNext();
}
rs.Close();
%>
```

Sample Output

```
France
Germany
United States
Canada
Austria
Italy
Netherlands
Switzerland
England
Sweden
Japan
Taiwan
Korea
China
Australia
Belgium
Denmark
Spain
Mexico
Brazil
Finland
Euroland
```

Applies To

Recordset Object

Fields Property

A collection of Field objects. Each field represents a column in the result set of the query.

This property is read-only.

Type

Collection

Example

```
<%
var rs = new Recordset();
rs.Open("Select * from Country","GOS");
Response.Write("Number of fields in the result set: " + rs.Fields.Count);
%>
```

Sample Output

```
Number of fields in the result set: 7
```

Applies To

Recordset Object

MaxRecords Property

Sets or returns the maximum number of records to return.

To work properly, you must set this property before calling the Recordset.Open method.

This property is read-write.

Type

Integer

Default Value

0 -- Returns zero records

-1 -- Unlimited number of records

Example

```
<%
var rs = new Recordset();
rs.MaxRecords = 10;
rs.Open("Select SalesCountryCode,Country from Country", "GOS");
while(!rs.EOF)
{
    Response.Write("<option value='" + rs.Fields("SalesCountryCode") + "'>" +
rs.Fields("Country") + "</option>");
    Response.Write("<br>");
    rs.MoveNext();
}
%>
```

Sample Output

```
France
Germany
United States
Canada
Austria
Italy
Netherlands
Switzerland
England
Sweden
```

Applies To

Recordset Object

Close Method

Closes the Recordset. If a Recordset is not explicitly closed, PowerPrompts closes it at the end of the page. Use this method to reuse existing Recordset objects.

Parameters

None

Return Type

None

Example

This example writes each country name and then each product line.

```
<%
var rs = new Recordset();
rs.Open("Select SalesCountryCode,Country from Country", "GOS");
// Must close the recordset before the recordset object can be reused
while(!rs.EOF)
{
    Response.Write("<option value='" + rs.Fields("SalesCountryCode") + "'>" +
rs.Fields("Country") + "</option>");
    Response.Write(",&nbsp;");
    rs.MoveNext();
}
Response.Write("<br><br>");
rs.Close();
rs.Open("Select Productline from Productline","GOS");
while(!rs.EOF)
{
    Response.Write("<option>" + rs.Fields("Productline") + "</option>");
    Response.Write(",&nbsp;");
    rs.MoveNext();
}
%>
```

Sample Output

```
France, Germany, United States, Canada, Austria, Italy, Netherlands, Switzerland,
England, Sweden, Japan, Taiwan, Korea, China, Australia, Belgium, Denmark, Spain,
Mexico, Brazil, Finland, Euroland,
Camping Equipment, Mountaineering Equipment, Personal Accessories, Outdoor Protection,
Golf Equipment,
```

Applies To

Recordset Object

MoveNext Method

Advances to the next row in the Recordset.

Parameters

None

Return Type

None

Example

This example writes each country name and currency on its own row.

```
<%
var rs= new Recordset();
rs.Open("Select CurrencyName,Country from Country order by Country", "GOS");
while(!rs.EOF)
{
    Response.Writeln(rs.Fields("Country") + " " + rs.Fields("CurrencyName"));
    Response.Write("<br>");
    rs.MoveNext();
}
rs.Close();
%>
```

Sample Output

```
Australia dollars
Austria schillings
Belgium francs
Brazil reals
Canada dollars
China renminbi
Denmark kroner
England pounds
Euroland euros
Finland markka
France francs
Germany marks
Italy lira
Japan yen
Korea won
Mexico pesos
Netherlands guilders
Spain pesetas
Sweden krona
Switzerland francs
Taiwan new dollar
United States dollars
```

Applies To

Recordset Object

Open Method

Populates the Recordset with the rows returned from executing the SQL.

Parameters

The SQL parameter is the SQL to be executed. The dbName parameter is the name of the Cognos logical database.

There are two optional string parameters. The dbUserID parameter is the database user ID to use when connecting to the database. The dbPassword is the database password for this user ID.

SQL: String, Mandatory

dbName: String, Mandatory

dbUserID: String, Optional

dbPassword: String, Optional

Return Type

None

Example1

This example shows opening a Recordset to a logical database that does not require a database signon.

```
<%
var rs = new Recordset();
rs.Open("Select ProductLineCode, ProductLine from ProductLine order by ProductLine",
"GOS");
while(!rs.EOF)
{
    Response.Write("<option value='" + rs.Fields("ProductLineCode") + "'>" +
rs.Fields("ProductLine") + "</option>");
    Response.Write("<br>");
    rs.MoveNext();
}
rs.Close();
%>
```

Sample Output1

```
Camping Equipment
Golf Equipment
Mountaineering Equipment
Outdoor Protection
Personal Accessories
```

Example2

This example demostrates using SQL from an Impromptu report.

The sample code is the same as Example1 except replace line 2 (rs.Open…) with the following lines:

```
var myQuery = new Query("pwp_tutorial_js.imr");
rs.Open(myQuery.SQL, "GOS");
```

Example3

This example demostrates using SQL from a string.

The sample code is the same as Example1 except replace line 2 (rs.Open…) with the following lines:

```
var sSQL = new String;
sSQL = "Select Country from Country";
rs.Open(sSQL.SQL, "GOS");
```

Applies To

Recordset Object

Request Object

Provides information about the client's current request.

Property	Description
Cookies Property	Returns all incoming cookies as a StringList.
ServerVariables Property	Returns all server variables as a StringList.
Variables Property	Returns the variables for the current request, which differ from application variables that persist between requests as a StringList.

Cookies Property

Returns all incoming cookies as a StringList. A cookie is a name/value pair sent by the browser. The cookies are ordered such that those with more specific path attributes precede those that are less specific.

This property is read-only.

Type

StringList

Example1

This example shows a list of all the cookies and their corresponding values.

```
<%
// List all the cookies
for (var myCookie in Request.Cookies)
{
    Response.Write(myCookie + " = " + Request.Cookies(myCookie) + "<br/>");
}
%>
```

Example2

This example shows the value of the cookie named CookieName.

```
<% =Request.Cookies("CookieName") %>
```

Applies To

Request Object

ServerVariables Property

Returns all server variables as a StringList. You cannot index server variables by number since there is no order to these variables.

The following server variables can be returned:

AUTH_TYPE	AUTH_USER
CONTENT_LENGTH	CONTENT_TYPE
GATEWAY_INTERFACE	HTTP_ACCEPT
HTTP_ACCEPT_LANGUAGE	HTTP_COOKIE
HTTP_USER_AGENT	HTTP_ACCEPT_CHARSET
HTTP_HOST	HTTP_RANGE

HTTP_REFERER	HTTPS
LOGON_USER	PATH_INFO
PATH_TRANSLATED	QUERY_STRING
REMOTE_ADDR	REMOTE_HOST
REMOTE_IDENT	REMOTE_USER
REQUEST_METHOD	SCRIPT_NAME
SERVER_NAME	SERVER_PORT
SERVER_PROTOCOL	SERVER_SOFTWARE

This property is read-only.

Type

StringList

Example

This example shows the value of the server variable called SERVER_NAME.

```
<%=Request.ServerVariables("SERVER_NAME")%>
```

Applies To

Request Object

Variables Property

Returns the variables for the current request, which differ from application variables that persist between requests as a StringList. Only use this property if you require the unprocessed form variable pairs. Use the App.Variables property to retrieve variables set in the application.

This property is read-only.

Type

StringList

Example

This example assigns the value of the CountryCd variable to the JavaScript variable myValue. The CountryCd variable must be set on a previous page.

```
<%
var myValue = Request.Variables("CountryCd");
%>
```

Applies To

Request Object

Response Object

Sends content back to the browser.

Property	Description
ContentType Property	Specifies the HTTP content type for the response.

Method	Description
AppendCookie Method	Sets a cookie.
AppendHeader Method	Adds a field to the HTTP header of the response.
Clear Method	Clears both the content and HTTP header fields.
ClearContent Method	Clears all the content for the response.
ClearHeaders Method	Clears all HTTP header fields for the response.
Redirect Method	Redirects the browser to another URL address.
Write Method	Inserts a string into the response that the browser receives.
WriteFile Method	Inserts the contents of a file into the response the browser receives.
Writeln Method	Inserts a string (followed by a newline) into the response that browser receives.

ContentType Property

Specifies the HTTP content type for the response. For information about for MIME Type information, please refer to the HTTP standard.

If the page is HTML, there is no need to set this property because HTML is the default. Only set this property once per page.

This property is read-write.

Type

String

Default Value

Text/HTML

Example1

This example shows how to send plain text to the browser.

```
<%
Response.ContentType = "text/plain";
Response.Write("This is a plain text message.");
%>
```

Sample Output1

```
This is a plain text message.
```

Example2

This example shows how to send an Excel workbook to the browser.

```
<%
Response.ContentType = "application/vnd.ms-excel";
Response.AppendHeader("Content-Disposition", "inline; filename=file1.xls");
%>
<table>
<tr>
<td>Product Number</td>
<td>Product Name</td>
<td>Product Cost</td>
</tr>
<tr>
<td align=left>105</td>
<td>Hailstorm Titanium Woods Set</td>
<td align=left>$555.90</td>
</tr>
</table>
```

Applies To

Response Object

AppendCookie Method

Sets a cookie. For more information about cookies, please see the cookie specification from Netscape.

Parameters

CookieName: String, Mandatory

Example

```
<% Response.AppendCookie( "MyCookie=Laventus3; path=/" ), %>
```

Applies To

Response Object

AppendHeader Method

Adds a field to the HTML header of the response. This method is an advanced feature and is not required for normal use.

Parameters

HeaderField: String, Mandatory

Example

```
<%Response.AppendHeader("Content-Type: text/plain");%>
```

Applies To

Response Object

Clear Method

Clears both the content and HTTP header fields. This method has the same effect as calling both the Response.ClearContent and Response.ClearHeader methods. This method is an advanced feature and is not required for normal use.

Example

```
<% Response.Clear(); %>
```

Applies To

Response Object

ClearContent Method

Clears all the content for the response. This method is an advanced feature and is not required for normal use.

Example

```
<%Response.ClearContent();%>
```

Applies To

Response Object

ClearHeaders Method

Clears all HTTP header fields for the response. This method is an advanced feature not required for normal use.

Example

```
<%Response.ClearHeaders()%>
```

Applies To

Response Object

Redirect Method

Redirects the browser to another URL address.

Parameters

URL: String, Mandatory

Example

This example redirects the browser to the Cognos Web site.

```
<%Response.Redirect( "www.cognos.com" );%>
```

Applies To

Response Object

Write Method

Inserts a string into the response that the browser receives. This method automatically converts numeric data to a string.

Parameters

aString: String, Mandatory

Return Type

None

Example1

This example sends text to the browser.

```
<%
// Write a string to the browser
Response.Write("<b>This is a bold string.</b>");
%>
```

Example2

This example sends a drop-down list containing Products to the browser.

```
<select name="ProdType">
<%
var rs = new Recordset;
rs.Open("Select ProductTypeCode, ProductType from ProductType where ProductLineCode = "
+ App.Variables("ProdLine"), "GOS");
while(!rs.EOF)
{
    Response.Write("<option value='" + rs.Fields("ProductTypeCode") + "'>" +
rs.Fields("ProductType") + "</option>");
    rs.MoveNext();
}
rs.Close();
%>
</select>
```

Sample Output

```
Irons
Woods
Putters
Golf Accessories
```

Applies To

Response Object

WriteFile Method

Inserts the contents of a file into the response the browser receives.

This text is not interpreted by the server-side JavaScript engine.

Parameters

Filename: String, Mandatory

Example

This example writes the contents of the file CompanyPageFooter.txt to the browser.

```
<% Response.WriteFile( App.Path + "CompanyPageFooter.txt" ); %>
```

Applies To

Response Object

Writeln Method

Inserts a string (followed by a newline) into the response that browser receives.

Note: If the content type is HTML, and you want a new line to appear in the browser, use the <br> tag.

Parameters

aString: String, Mandatory

Return Type

None

Example1

This example writes the names of each country and their currency on their own line.

```
<%
var rs= new Recordset();
rs.Open("Select CurrencyName,Country from Country order by Country", "GOS");
while(!rs.EOF)
{
    Response.Writeln(rs.Fields("Country") + " " + rs.Fields("CurrencyName"));
    Response.Write("<br>");
    rs.MoveNext();
}
rs.Close();
%>
```

Sample Output1

```
Australia dollars
Austria schillings
Belgium francs
Brazil reals
Canada dollars
China renminbi
Denmark kroner
England pounds
Euroland euros
Finland markka
France francs
Germany marks
Italy lira
Japan yen
Korea won
Mexico pesos
Netherlands guilders
Spain pesetas
Sweden krona
Switzerland francs
Taiwan new dollar
United States dollars
```

Example2

This example writes the Body tag on a new line in the browser.

```
Response.Writeln('<BODY BGCOLOR="#FFFFFF"><table width="100%" bgcolor="#cddfff"
cellpadding="0" cellspacing="0"><tr><td height="1" bgcolor="#003366"
width="100%"></td></tr><tr><td><H1>Sales Rep Details</H1></td></tr></table>')
```

Applies To

Response Object

Server Object

Provides access to methods and properties on the server. Most of these methods and properties serve as utility functions.

Property	Description
ScriptTimeout Property	Sets or returns the number of seconds the server waits before returning the "Script Timeout reached" error.

Method	Description
CreateObject Method	Creates an instance of a COM (Component Object Model) object given a ProgID. Only works under Windows.
FormatNumber Method	Formats a number.
HTMLEncode Method	Applies HTML encoding to the specified string.
URLDecode Method	Applies URL decoding rules. This method is the opposite of the URLEncode method.
URLEncode Method	Applies URL encoding rules. This method is the opposite of the URLDecode method.

ScriptTimeout Property

Sets or returns the number of seconds the server waits before returning the "Script Timeout reached" error. Use this property to prevent an infinite loop on the server.

This property is read-write.

Type

Integer

Default Value

90

Example

```
<%
// Returns the error page after 20 seconds
Server.ScriptTimeout = 20;
var i = 0;
while (true)
{
    i++
}
%>
```

Sample Output

```
Error
Server.ScriptTimeout exceeded (20 seconds) while executing line number line number of
page page name.
i++
Check the server event log for more details.
```

Applies To

Server Object

CreateObject Method

Creates an instance of a COM (Component Object Model) object given a ProgID. This method works only in Windows because there is no COM support under UNIX.

Parameters

ProgID: String

Return Type

Object

Example

This example shows how to fetch data using ADO. For more information about ADO, see your appropriate Microsoft documentation.

```
<%
var oConn = Server.CreateObject("ADODB.Connection");
// This is any valid ODBC DSN
oConn.Open("GOScer3");
// This is any value SQL for the above DSN
var oRs = oConn.Execute("Select Country,SalesCountryCode from Country order by
Country");
Response.Writeln("<select name='country_cd'>");
while (!oRs.EOF)
{
    Response.Writeln("<option value='" + oRs.Fields("SalesCountryCode").Value + "'>" +
oRs.Fields("Country").Value + "</option>");
    oRs.MoveNext();
}
Response.Writeln("</select>");
%>
```

Applies To

Server Object

FormatNumber Method

Formats a number. For more information about numeric formats in Impromptu, see Numeric Format Symbols in the Impromptu User online help.

Parameters

Number: String, Mandatory

Format: String, Mandatory

Return Type

String

Example

```
<%
var myNumber = "67.0000012";
Response.Write(Server.FormatNumber(myNumber,"#,#.00"));
%>
```

Sample Output

```
67.00
```

Applies To

Server Object

HTMLEncode Method

Applies HTML encoding to the specified string. For example, this method changes any less than (<) to &lt;, greater than (>) to &gt; and ampersand (&) to &amp;. For more information about HTML encoding, see to your HTML documentation.

Parameters

Value: String

Return Type

String

Example

```
<%
var myString;
myString = "<>&'\"";
Response.Write(Server.HTMLEncode(myString));
%>
```

Sample Output

```
<>&'"
```

Browser Sequence

```
&lt;&gt;&amp;&apos;&quot;
```

Applies To

Server Object

URLDecode Method

Applies URL decoding rules. For example, this method changes hex values to their character equivalents. This method is the opposite of the URLEncode method.

Parameters

Value: String

Return Type

String

Example

This example encodes and then decodes a string.

```
<%
var myString;
myString= " &^*@";
var myString2;
myString2 = ""
Response.Write(Server.URLEncode(myString));
Response.Write("<br>")
myString2 = Server.URLEncode(myString);
Response.Write(Server.URLDecode(myString2));
%>
```

Sample Output

```
%20%26%5E%2A%40
&^*@
```

Applies To

Server Object

URLEncode Method

Applies URL encoding rules. For example, this method changes any non-alphanumeric characters to the equivalent hex values. This method is the opposite of the URLDecode method.

Parameters

Value: String

Return Type

String

Example

This example encodes and then decodes a string.

```
<%
var myString;
myString= " &^*@";
var myString2;
myString2 = ""
Response.Write(Server.URLEncode(myString));
Response.Write("<br>")
myString2 = Server.URLEncode(myString);
Response.Write(Server.URLDecode(myString2));
%>
```

Sample Output

```
%20%26%5E%2A%40
&^*@
```

Applies To

Server Object

StringList Object

A collection of string values.

Property	Description
Count Property	Returns the number of string values in the StringList.

Method	Description
Contains Method	Tests whether the StringList contains a value.
Item Method	Retrieves an indexed string value.
Join Method	Returns all values concatenated. An optional separator can be specified. The default is a comma (,).
Operator() Method (StringList)	Specifies the index operator used to access items in the collection. You can specify a numeric index.
toString Method	Returns all values concatenated. This method is the same as the Join property except that it takes no parameters.

Count Property

Returns the number of string values in the StringList.

This property is read-only.

Type

Integer

Example

This example adds a drop-down list for countries on a page and then, on the next page, prints out the number of countries you selected.

Page 1

```
<select name="Countries" multiple>
<%
var rs = new Recordset;
rs.Open("Select Country from Country order by Country", "GOS");
while(!rs.EOF)
{
    Response.Write("<option>" + rs.Fields("Country") + "</option>");
    rs.MoveNext();
}
rs.Close();
%>
</select>
```

Page 2

```
<%
var myStringList = App.Variables("Countries");
Response.Write("There are " + myStringList.Count + " string items.");
%>
```

Sample Output

```
There are 22 string items.
```

Applies To

StringList Object

Contains Method

Tests whether the StringList contains a value.

Parameters

ValueToFind: String, Mandatory

Return Type

Boolean

Example

This example adds a drop-down list for countries on a page and then, on the next page, test if Canada was one of the countries you selected.

Page 1

```
<select name="Countries" multiple>
<%
var rs = new Recordset;
rs.Open("Select Country from Country order by Country", "GOS");
while(!rs.EOF)
{
    Response.Write("<option>" + rs.Fields("Country") + "</option>");
    rs.MoveNext();
}
rs.Close();
%>
</select>
```

Page 2

```
<%
var myStringList = App.Variables("Countries");
// Need to populate StringList object
if (myStringList.Contains("Canada"))
{
    Response.Write("Canada has been picked.");
}
else
{
    Response.Write("Canada hasn’t been picked.");
}
%>
```

Sample Output

```
Canada has been picked.
```

Applies To

StringList Object

Item Method

Retrieves an indexed string value. The index is zero-based meaning it starts at zero.

Parameters

Index: Integer, Mandatory

Return Type

String

Example

This example adds a drop-down list for countries on a page and then, on the next page, shows the second item in the string.

Page 1

```
<select name="Countries" multiple>
<%
var rs = new Recordset;
rs.Open("Select Country from Country order by Country", "GOS");
while(!rs.EOF)
{
    Response.Write("<option>" + rs.Fields("Country") + "</option>");
    rs.MoveNext();
}
rs.Close();
%>
</select>
```

Page 2

```
<%
var myStringList = App.Variables("Countries");
var x = new String();
x = myStringList.Item(1);
// Index starts at 0 so the second item is shown
Response.Write(x + " is the second item in the string.");
%>
```

Sample Output

```
Austria is the second item in the string.
```

Applies To

StringList Object

Join Method

Returns all values concatenated. An optional separator can be specified. The default is a comma (,).

Parameters

Separator: String, Mandatory

Return Type

String

Example

This example adds a drop-down list for countries on a page and then, on the next page, shows the countries you selected with a comma between each name.

Page 1

```
<select name="Countries" multiple>
<%
var rs = new Recordset;
rs.Open("Select Country from Country order by Country", "GOS");
while(!rs.EOF)
{
    Response.Write("<option>" + rs.Fields("Country") + "</option>");
    rs.MoveNext();
}
rs.Close();
%>
</select>
```

Page 2

```
<%
var myStringList = App.Variables("Countries").Join(", ");
Response.Write(myStringList);
%>
```

Sample Output

```
Australia, Austria, Belgium
```

Applies To

StringList Object

Operator() Method (StringList)

Specifies the index operator used to access items in the collection. The index is zero-based meaning it starts at zero.

Parameters

You can specify a numeric index.

Index: Numeric

Return Type

An item in a collection

Example

This example returns the first country in the StringList.

```
<%=App.Variables("Country")(0)%>
```

Applies To

StringList Object

toString Method

Returns all values concatenated. This method is the same as the Join property except that it takes no parameters.

This is the default parameter of StringList.

Parameters

None

Return Type

String

Example

```
<%
switch(App.Variables("OrderMonth").toString())
{
    case "1" :
        Response.Write("<tr><td>Order Month</td><td>" + "January" + "</td></tr>");
        break;
...
}
%>
```

Applies To

StringList Object

Upfront Object

Provides integration with the Upfront server so that PowerPrompts applications can use themes and languages.

Property	Description
Language Property	Returns the current language selected in Upfront.
Locale Property	Returns the current locale selected in Upfront.
Theme Property	Returns the name of the current theme selected in Upfront.

Method	Description
ExecuteCommand Method	Used to pass an XML command to the Upfront server. The return value is an XML document indicating success or failure as well as any results of the command.
GetPageFragment Method	Returns an HTML page fragment in Upfront based on the current theme and language. The return value is the page contents as processed by the Upfront template engine.

Language Property

Returns the current language selected in Upfront.

This property is read-only.

Type

String

Example

```
<%
// Let the user know that this application is only available in English
if (Upfront.Language != "en")
{
    Response.Write("<b>You have selected a language other than English. This application
is only available in English.</b>");
}
else
{
    Response.Write("<b>Upfront likes English.</b>");
}
%>
```

Sample Output

```
Upfront likes English.
```

Applies To

Upfront Object

Locale Property

Returns the current locale selected in Upfront.

This property is read-only.

Type

String

Example

```
<%
// Let the user know which locale is currently selected
Response.Write("Locale: " + Upfront.Locale);
%>
```

Sample Output

```
Locale: en-us
```

Applies To

Upfront Object

Theme Property

Returns the name of the current theme selected in Upfront.

This property is read-only.

Type

String

Example

```
<!-- Let the user know what theme is currently selected -->
Theme: <%=Upfront.Theme%>
```

Sample Output

```
Theme: standard70
```

Applies To

Upfront Object

ExecuteCommand Method

Used to pass an XML command to the Upfront server. The return value is an XML document indicating success or failure as well as any results of the command.

The appropriate XML namespace must be specified for the root element of the command. The namespace is http://developer.cognos.com/schemas/upfront.

Parameters

XMLCommand: String, Mandatory

Return Type

String

Example

```
<%=Server.HTMLEncode(Upfront.ExecuteCommand("<DescribeUser
xmlns=\"http://developer.cognos.com/schemas/upfront/\"/>"))%>
```

Applies To

Upfront Object

GetPageFragment Method

Returns an HTML page fragment in Upfront based on the current theme and language. The return value is the page contents as processed by the Upfront template engine.

Parameters

PageFragmentName: String, Mandatory

Return Type

String

Example

```
<%=Upfront.GetPageFragment("header.html")%>
```

Applies To

Upfront Object

User Object

Provides the user information for a valid ticket in Access Manager.

Property	Description
Description Property	Returns the description of the current user set in Access Manager.
Email Property	Returns the email address of the current user set in Access Manager.
Telephone Property	Returns the telephone number of the current user set in Access Manager.

Property	Description
UserClass Property	Returns the user class name for the current user. UserClass is the user class that the report is being run under.
UserClasses Property	Lists the available user classes for this user. UserClasses is the list of all user classes of which the user is currently a member of in Access Manager.
UserName Property	Returns the name of the current user set in Access Manager.

Description Property

Returns the description of the current user set in Access Manager.

This property is read-only.

Type

String

Example

This example shows the user name, user class, email, telephone number and description of the user in a table.

```
<%
// Display the user's properties in a table
Response.Writeln("<table>");
Response.Writeln("<tr><td>UserName:</td><td>" + User.UserName + "</td></tr>");
Response.Writeln("<tr><td>UserClass:</td><td>" + User.UserClass + "</td></tr>");
Response.Writeln("<tr><td>Email:</td><td>" + User.Email + "</td></tr>");
Response.Writeln("<tr><td>Telephone:</td><td>" + User.Telephone + "</td></tr>");
Response.Writeln("<tr><td>Description:</td><td>" + User.Description + "</td></tr>");
Response.Writeln("</table>");
%>
```

Applies To

User Object

Email Property

Returns the email address of the current user set in Access Manager.

This property is read-only.

Type

String

Example

This example shows the user name, user class, email, telephone number and description of the user in a table.

```
<%
// Display the user's properties in a table
Response.Writeln("<table>");
Response.Writeln("<tr><td>UserName:</td><td>" + User.UserName + "</td></tr>");
Response.Writeln("<tr><td>UserClass:</td><td>" + User.UserClass + "</td></tr>");
Response.Writeln("<tr><td>Email:</td><td>" + User.Email + "</td></tr>");
Response.Writeln("<tr><td>Telephone:</td><td>" + User.Telephone + "</td></tr>");
Response.Writeln("<tr><td>Description:</td><td>" + User.Description + "</td></tr>");
Response.Writeln("</table>");
%>
```

Applies To

User Object

Telephone Property

Returns the telephone number of the current user set in Access Manager.

This property is read-only.

Type

String

Example

This example shows the user name, user class, email, telephone number and description of the user in a table.

```
<%
// Display the user's properties in a table
Response.Writeln("<table>");
Response.Writeln("<tr><td>UserName:</td><td>" + User.UserName + "</td></tr>");
Response.Writeln("<tr><td>UserClass:</td><td>" + User.UserClass + "</td></tr>");
Response.Writeln("<tr><td>Email:</td><td>" + User.Email + "</td></tr>");
Response.Writeln("<tr><td>Telephone:</td><td>" + User.Telephone + "</td></tr>");
Response.Writeln("<tr><td>Description:</td><td>" + User.Description + "</td></tr>");
Response.Writeln("</table>");
%>
```

Applies To

User Object

UserClass Property

Returns the user class name for the current user. UserClass is the user class that the report is being run under.

This property is read-only.

Type

String

Example

This example shows the user name, user class, email, telephone number and description of the user in a table.

```
<%
// Display the user's properties in a table
Response.Writeln("<table>");
Response.Writeln("<tr><td>UserName:</td><td>" + User.UserName + "</td></tr>");
Response.Writeln("<tr><td>UserClass:</td><td>" + User.UserClass + "</td></tr>");
Response.Writeln("<tr><td>Email:</td><td>" + User.Email + "</td></tr>");
Response.Writeln("<tr><td>Telephone:</td><td>" + User.Telephone + "</td></tr>");
Response.Writeln("<tr><td>Description:</td><td>" + User.Description + "</td></tr>");
Response.Writeln("</table>");
%>
```

Applies To

User Object

UserClasses Property

Lists the available user classes for this user. UserClasses is the list of all user classes of which the user is currently a member of in Access Manager.

This property is read-only.

Type

StringList

Example

This example joins together all the user classes that are available for this user.

```
<%=UserClasses.Join()%>
```

Applies To

User Object

UserName Property

Returns the name of the current user set in Access Manager.

This property is read-only.

Type

String

Example

This example shows the user name, user class, email, telephone number and description of the user in a table.

```
<%
// Display the user's properties in a table
Response.Writeln("<table>");
Response.Writeln("<tr><td>UserName:</td><td>" + User.UserName + "</td></tr>");
Response.Writeln("<tr><td>UserClass:</td><td>" + User.UserClass + "</td></tr>");
Response.Writeln("<tr><td>Email:</td><td>" + User.Email + "</td></tr>");
Response.Writeln("<tr><td>Telephone:</td><td>" + User.Telephone + "</td></tr>");
Response.Writeln("<tr><td>Description:</td><td>" + User.Description + "</td></tr>");
Response.Writeln("</table>");
%>
```

Applies To

User Object

JavaScript Report Alteration Methods

Use the JavaScript Report Alteration Methods to alter Impromptu reports. Use the Script Editor dialog box to add these methods to your PowerPrompts application. You cannot use these methods in your HTML pages. For more information about a script, see "Script: Overview" (p. 21).

Syntax Rules

- Report methods require a GetReport() method prefixed to them.
- Report Object methods require GetReportObject() method prefixed to them.
- Column methods require GetColumnTitle() method prefixed to them.
- Query methods require GetQuery() method prefixed to them
- JavaScript is case-sensitive. AddDataItem is correct but ADDdataITEM is not.
- Use double quotation marks to enclose Impromptu expressions passed to these methods.
- Use single quotes to enclose string values within Impromptu expressions.

  For an example that uses these rules, see "Add a Script" (p. 21).

Note

To open the help topic for a method, in the Script Editor dialog box, select the method name and press F1.

GetColumnTitle Method

Returns the column title for the specified column.

Syntax

```
GetColumnTitle(columnName)
```

Parameters

columnName: String

Example

This example changes the Product column title to Product Title.

```
GetColumnTitle("Product").SetText("Product Title");
```

Related Topics

- "ApplyStyle Column Title Method" (p. 89)
- "GetQuery Method" (p. 82)
- "GetReport Method" (p. 82)
- "GetReportObject Method" (p. 83)
- "SetText Column Title Method" (p. 89)
- "SetTextJustification Column Title Method" (p. 89)

GetQuery Method

Returns the main query of the report.

Syntax

```
GetQuery()
```

Example

```
GetQuery().AddColumn("Total Qty","total([Qty])");
```

Related Topics

- "AddColumn Query Method" (p. 90)
- "AndFilterBy Query Method" (p. 90)
- "AssociateColumn Query Method" (p. 91)
- "GetColumnTitle Method" (p. 82)
- "GetReport Method" (p. 82)
- "GetReportObject Method" (p. 83)
- "GroupBy Query Method" (p. 91)
- "SetMaxRows Query Method" (p. 93)

GetReport Method

Returns the report with the specified name.

Syntax

```
GetReport()
```

Example

This example adds the catalog column Qty to the final report.

```
GetReport().AddDataItem("Qty","[\\Orders\\Order Details\\Qty]");
```

Related Topics

- "ApplyTemplate Report Method" (p. 84)
- "GetColumnTitle Method" (p. 82)
- "GetQuery Method" (p. 82)
- "GetReportObject Method" (p. 83)
- "RemoveReportObject Report Method" (p. 84)
- "SetListInsertCursor Report Method" (p. 85)
- "SetPrimaryFrame Report Method" (p. 85)

GetReportObject Method

Returns the report object with the specified name.

Syntax

```
GetReportObject(repObjName)
```

Parameters

repObjName: String

Example

This example applies the style named UglyRed to the column Col1.

```
GetReportObject("Col1").ApplyStyle("UglyRed");
```

Related Topics

- "ApplyStyle Report Object Method" (p. 86)
- "GetColumnTitle Method" (p. 82)
- "GetQuery Method" (p. 82)
- "GetReport Method" (p. 82)
- "SetText Report Object Method" (p. 87)
- "VerticalAlign Report Object Method" (p. 88)

AddDataItem Report Method

Adds the specified expression as a column into the query and inserts a corresponding report object. The object is inserted into the default position of the primary frame.

Syntax

```
AddDataItem(repObjName,data)
```

Parameters

repObjName: String

data: String

Example

This example adds the catalog column Qty to the final report.

```
GetReport().AddDataItem("Qty","[\\Orders\\Order Details\\Qty]");
```

Related Topics

- "ApplyTemplate Report Method" (p. 84)
- "GetReport Method" (p. 82)
- "GetReportObject Method" (p. 83)
- "RemoveReportObject Report Method" (p. 84)
- "SetListInsertCursor Report Method" (p. 85)
- "SetPrimaryFrame Report Method" (p. 85)

ApplyTemplate Report Method

Applies the specified template to the report.

Syntax

```
ApplyTemplate(templateName)
```

Parameters

templateName: String

Example1

This example applies a template named Laventus.imt to the report. Laventus.imt is located in C:\PowerPrompts Apps.

```
GetReport().ApplyTemplate("C:\\PowerPrompts Apps\\Laventus.imt");
```

Example2

This example applies a template named Laventus.imt to the report. Laventus.imt is located in a folder named Templates which is beneath the folder that contains the PowerPrompts application.

```
GetReport().ApplyTemplate(".\\Templates\\Laventus.imt");
```

Note: You must always specify the template name, but you can specify an absolute or relative path. Relative paths are relative to the report location. You can get the path to the application using the App.Path property.

Related Topics

- "AddDataItem Report Method" (p. 83)
- "GetReport Method" (p. 82)
- "RemoveReportObject Report Method" (p. 84)
- "SetListInsertCursor Report Method" (p. 85)
- "SetPrimaryFrame Report Method" (p. 85)

RemoveReportObject Report Method

Removes the named report object from the report.

This method does not work against hidden columns in an Impromptu report.

Syntax

```
RemoveReportObject(repObjName)
```

Parameters

repObjName: String

Example

This removes the report object named Country from the final report.

```
GetReport().RemoveReportObject("Country");
```

Related Topics

- "AddDataItem Report Method" (p. 83)
- "ApplyTemplate Report Method" (p. 84)
- "GetReport Method" (p. 82)
- "SetListInsertCursor Report Method" (p. 85)
- "SetPrimaryFrame Report Method" (p. 85)

SetListInsertCursor Report Method

Sets where columns are added to list frames. By default, new columns are added to the right of existing columns.

Specifying	Places the new columns
-1	to the right of the existing columns
0	to the left of the existing columns
1	right of the first existing column
2	right of the second existing column
N (where N is a positive integer)	right of the Nth existing column

Syntax

```
SetListInsertCursor(columnInsertIndex)
```

Parameters

columnInsertIndex: Integer

Example

```
GetReport().SetListInsertCursor(0);
```

Note: In the Script Editor dialog box, this method must appear above the AddDataItem method because the script must set the insertion point before the new columns are added.

Related Topics

- "AddDataItem Report Method" (p. 83)
- "ApplyTemplate Report Method" (p. 84)
- "GetReport Method" (p. 82)
- "RemoveReportObject Report Method" (p. 84)
- "SetPrimaryFrame Report Method" (p. 85)

SetPrimaryFrame Report Method

Sets the primary frame for the report. This primary frame is where Impromptu adds new report objects.

Syntax

```
SetPrimaryFrame(frameName)
```

Parameters

frameName: String

Example

This example sets the Product Type Footer as the primary frame for the report.

```
GetReport().SetPrimaryFrame("Product Type Footer");
```

Related Topics

- "AddDataItem Report Method" (p. 83)
- "ApplyTemplate Report Method" (p. 84)
- "GetReport Method" (p. 82)
- "RemoveReportObject Report Method" (p. 84)
- "SetListInsertCursor Report Method" (p. 85)

AddConditionalFormat Report Object Method

Adds a conditional format object to the report object. A conditional format is made up of a named condition and a named style.

Syntax

```
AddConditionalFormat(namedCondition,styleName)
```

Parameters

namedCondition: String

styleName: String

Example

This example adds conditional format to the Qty object.

```
GetReportObject("Qty").AddConditionalFormat("Small Amount","Bad");
```

Related Topics

- "ApplyStyle Report Object Method" (p. 86)
- "GetReportObject Method" (p. 83)
- "HorizontalAlign Report Object Method" (p. 86)
- "HorizontalAlignToColumn Report Object Method" (p. 87)
- "SetText Report Object Method" (p. 87)
- "SetTextJustification Report Object Method" (p. 88)
- "VerticalAlign Report Object Method" (p. 88)

ApplyStyle Report Object Method

Applies the named style to the report object. The style must exist in the Impromptu.ini file on the server that the PowerPrompts application is deployed to.

Syntax

```
ApplyStyle(styleName)
```

Parameters

styleName: String

Example

```
GetReportObject("Product").ApplyStyle("Seeing Red 1");
```

Related Topics

- "AddConditionalFormat Report Object Method" (p. 86)
- "GetReportObject Method" (p. 83)
- "HorizontalAlign Report Object Method" (p. 86)
- "HorizontalAlignToColumn Report Object Method" (p. 87)
- "SetText Report Object Method" (p. 87)
- "SetTextJustification Report Object Method" (p. 88)
- "VerticalAlign Report Object Method" (p. 88)

HorizontalAlign Report Object Method

Horizontally aligns the report object relative to its parent.

Syntax

```
HorizontalAlign(alignType)
```

Parameters

alignType: String (left or center or right)

Example

```
GetReportObject("Title").HorizontalAlign("center");
```

Related Topics

- "AddConditionalFormat Report Object Method" (p. 86)
- "ApplyStyle Report Object Method" (p. 86)
- "GetReportObject Method" (p. 83)
- "HorizontalAlignToColumn Report Object Method" (p. 87)
- "SetText Report Object Method" (p. 87)
- "SetTextJustification Report Object Method" (p. 88)
- "VerticalAlign Report Object Method" (p. 88)

HorizontalAlignToColumn Report Object Method

Horizontally aligns the report object relative to the specified column. For example, use for summaries within a footer.

Syntax

```
HorizontalAlignToColumn(alignToColumnName,alignType)
```

Parameters

alignToColumnName: String

alignType: String (left or center or right)

Example

```
GetReportObject("Total Qty").HorizontalAlignToColumn("Qty","right");
```

Related Topics

- "AddConditionalFormat Report Object Method" (p. 86)
- "ApplyStyle Report Object Method" (p. 86)
- "GetReportObject Method" (p. 83)
- "HorizontalAlign Report Object Method" (p. 86)
- "SetText Report Object Method" (p. 87)
- "SetTextJustification Report Object Method" (p. 88)
- "VerticalAlign Report Object Method" (p. 88)

SetText Report Object Method

Sets the text for a text frame.

Syntax

```
SetText(text)
```

Parameters

text: String

Example

```
GetReportObject("Title").SetText("Financial Report");
```

Related Topics

- "AddConditionalFormat Report Object Method" (p. 86)
- "ApplyStyle Report Object Method" (p. 86)
- "GetReportObject Method" (p. 83)
- "HorizontalAlign Report Object Method" (p. 86)
- "HorizontalAlignToColumn Report Object Method" (p. 87)
- "SetTextJustification Report Object Method" (p. 88)
- "VerticalAlign Report Object Method" (p. 88)

SetTextJustification Report Object Method

Aligns the text within the text frame.

Syntax

```
SetTextJustification(justifyType)
```

Parameters

justifyType: String (left or center or right)

Example

```
GetReportObject("Product").SetTextJustification("right");
```

Related Topics

- "AddConditionalFormat Report Object Method" (p. 86)
- "ApplyStyle Report Object Method" (p. 86)
- "GetReportObject Method" (p. 83)
- "HorizontalAlign Report Object Method" (p. 86)
- "HorizontalAlignToColumn Report Object Method" (p. 87)
- "SetText Report Object Method" (p. 87)
- "VerticalAlign Report Object Method" (p. 88)

VerticalAlign Report Object Method

Vertically aligns the report object relative to its parent frame.

Syntax

```
VerticalAlign(alignType)
```

Parameters

alignType: String (top or center or bottom)

Example

```
GetReportObject("Total Qty").VerticalAlign("center");
```

Related Topics

- "AddConditionalFormat Report Object Method" (p. 86)
- "ApplyStyle Report Object Method" (p. 86)
- "GetReportObject Method" (p. 83)
- "HorizontalAlign Report Object Method" (p. 86)
- "HorizontalAlignToColumn Report Object Method" (p. 87)
- "SetText Report Object Method" (p. 87)
- "SetTextJustification Report Object Method" (p. 88)

ApplyStyle Column Title Method

Applies the named style to the column title.

Syntax

```
ApplyStyle(styleName)
```

Parameters

styleName: String

Example

```
GetColumnTitle("Product").ApplyStyle("Seeing Red 1");
```

Related Topics

- "GetColumnTitle Method" (p. 82)
- "SetText Column Title Method" (p. 89)
- "SetTextJustification Column Title Method" (p. 89)

SetText Column Title Method

Sets the text for a column title.

Syntax

```
SetText(text)
```

Parameters

text: String

Example

```
GetColumnTitle("Product Type").SetText("Type of the Product");
```

Related Topics

- "ApplyStyle Column Title Method" (p. 89)
- "GetColumnTitle Method" (p. 82)
- "SetTextJustification Column Title Method" (p. 89)

SetTextJustification Column Title Method

Aligns the text within the text frame.

Syntax

```
SetTextJustification(justifyType)
```

Parameters

justifyType: String (left or center or right)

Example

```
GetColumnTitle("Product").SetTextJustification("right");
```

Related Topics

- "ApplyStyle Column Title Method" (p. 89)
- "GetColumnTitle Method" (p. 82)
- "SetText Column Title Method" (p. 89)

AddColumn Query Method

Adds a new column to the query with the specified name and expression.

Syntax

```
AddColumn(colName,data)
```

Parameters

colName: String

data: String

Example

```
GetQuery().AddColumn("Total Qty","total([Qty])");
```

Related Topics

- "AssociateColumn Query Method" (p. 91)
- "GetQuery Method" (p. 82)
- "Impromptu Expressions in PowerPrompts" (p. 94)
- "RemoveColumn Query Method" (p. 92)

AddNamedCondition Query Method

Adds a named condition to the query. Use this method to define conditional formatting.

Syntax

```
AddNamedCondition(conditionName,condition)
```

Parameters

conditionName: String

condition: String

Example

```
GetQuery().AddNamedCondition("Poor Sales","[\\Products\\Sales History\\Sales 95] <
5000");
```

Related Topics

- "GetQuery Method" (p. 82)

AndFilterBy Query Method

Adds the specified condition to the current filter. If the current filter is not empty, the condition is added by using an AND.

Note: This method does not work for Impromptu reports where you have written the SQL for the query.

Syntax

```
AndFilterBy(condition)
```

Parameters

condition: String

Example

```
GetQuery().AndFilterBy("year([Order Date]) in (95,96)");
```

Related Topics

- "AndSummaryFilterBy Query Method" (p. 91)
- "GetQuery Method" (p. 82)
- "OrFilterBy Query Method" (p. 92)

AndSummaryFilterBy Query Method

Adds the specified condition to the current summary filter. If the current filter is not empty, the condition is added by using an AND.

Note: This method does not work for Impromptu reports where you have written the SQL for the query.

Syntax

```
AndSummaryFilterBy(condition)
```

Parameters

condition: String

Example

```
GetQuery().AndSummaryFilterBy("[Total Sales] > 10000");
```

Related Topics

- "AndFilterBy Query Method" (p. 90)
- "GetQuery Method" (p. 82)
- "OrSummaryFilterBy Query Method" (p. 92)

AssociateColumn Query Method

Associates the specified column with the specified group column.

Syntax

```
AssociateColumn(colName,groupColName)
```

Parameters

colName: String

groupColName: String

Example

```
GetQuery().AssociateColumn("Total Sales","Country");
```

Related Topics

- "AddColumn Query Method" (p. 90)
- "GetQuery Method" (p. 82)
- "RemoveColumn Query Method" (p. 92)

GroupBy Query Method

Adds the named column to the end of the list of grouped columns. The column is sorted ascending or descending if the sort type is set to ascend or descend respectively.

Syntax

```
GroupBy(colName,sortType)
```

Parameters

colName: String

sortType: String (ascend or descend)

Example

```
GetQuery().GroupBy("Customer","descend");
```

Related Topics

- "AddColumn Query Method" (p. 90)
- "GetQuery Method" (p. 82)
- "SortBy Query Method" (p. 94)

OrFilterBy Query Method

Adds the specified condition to the current filter. If the current filter is not empty, the condition is added by using an OR.

Note: This method does not work for Impromptu reports where you have written the SQL for the query.

Syntax

```
OrFilterBy(condition)
```

Parameters

condition: String

Example

```
GetQuery().OrFilterBy("[\\Products\\Product] startswith('GO')");
```

Related Topics

- "AndFilterBy Query Method" (p. 90)
- "GetQuery Method" (p. 82)
- "OrSummaryFilterBy Query Method" (p. 92)

OrSummaryFilterBy Query Method

Adds the specified condition to the current summary filter. If the current filter is not empty, the condition is added by using an OR.

Note: This method does not work for Impromptu reports where you have written the SQL for the query.

Syntax

```
OrSummaryFilterBy(condition)
```

Parameters

condition: String

Example

```
GetQuery().OrSummaryFilterBy("[Country] = 'Canada'");
```

Related Topics

- "AndSummaryFilterBy Query Method" (p. 91)
- "GetQuery Method" (p. 82)
- "OrFilterBy Query Method" (p. 92)

RemoveColumn Query Method

Removes the specified column from the query and removes any report objects that reference the column.

Syntax

```
RemoveColumn(colName)
```

Parameters

colName: String

Example

```
GetQuery().RemoveColumn("Product Type");
```

Related Topics

- "AddColumn Query Method" (p. 90)
- "GetQuery Method" (p. 82)
- "AssociateColumn Query Method" (p. 91)

SetMaxRows Query Method

Sets the maximum number of rows that the query returns in the result set.

Syntax

```
SetMaxRows(maxRows)
```

Parameters

maxRows: Integer

Example

```
GetQuery().SetMaxRows(3000);
```

Related Topics

- "GetQuery Method" (p. 82)

SetPromptValue Query Method

Sets the values for the prompt in the Impromptu report that is associated with the PowerPrompts application. To ensure that the Impromptu report accepts multiple prompt values, use an "In" operator instead of an equals (=) operator in the filter statement of that report.

Tip: Use a comma to separate the values. If a comma is required as part of a value, precede that comma with the escape character. By default, a caret (^) is the escape character, but you can change it by updating the Separator Escape Character entry in the [Startup Options] section of your Impromptu.ini file.

Syntax

```
SetPromptValue(promptName,value1,...,valueN)
```

Parameters

For the promptName parameter, use the prompt name from the Available Prompts box of the Prompt Manager dialog box.

promptName: String

value1-valueN: Variant

Examples

In this example below, "Filter Date" is the prompt name and "1999-02-28" is the prompt value.

```
GetQuery().SetPromptValue("Filter Date","1999-02-28")
GetQuery().SetPromptValue("Country Code","App.Variables("myVar")");
```

Related Topics

- "GetQuery Method" (p. 82)

SortBy Query Method

Adds the named column to the end of the list of sorted columns. The column is sorted ascending or descending if the sort type is set to ascend or descend respectively.

Syntax

```
SortBy(colName,sortType)
```

Parameters

colName: String

sortType: String (ascend or descend)

Example

```
GetQuery().SortBy("Country","ascend")
```

Related Topics

- "AddColumn Query Method" (p. 90)
- "GetQuery Method" (p. 82)
- "GroupBy Query Method" (p. 91)

Impromptu Expressions in PowerPrompts

Impromptu expressions modify a query and are used in calculations, filters, and conditions. Impromptu expressions are similar but not identical between PowerPrompts and Impromptu.

In PowerPrompts, you type Impromptu expressions directly into the Script Editor dialog box but the code is JavaScript. If you want to create complex Impromptu expressions, create them in Impromptu first (by using an expression editor) and then recreate them in PowerPrompts with the following JavaScript modifications:

- Remove spaces from function names, for example, 'starts with' becomes 'startswith'.
- Use underscores instead of dashes for function names, for example, 'date-to-string' becomes 'date_to_string'.
- Enclose catalog references in square brackets and use a double backslash to start them, for example, [\\Admin.\\Country\\Country Cd].
- Distinguish query references from catalog references by not starting them with a slash, for example, [Col 1].
- Enclose prompts in the report with question marks, for example, ?PromptName?.
- Reference prompts that are stored in the catalog as you would any other catalog column.

The following table shows two Impromptu expressions and their equivalents in PowerPrompts:

PowerPrompts expression	Impromptu expression
[\Orders\Order Date] >= 1996-01-01	Order Date >= 1996-01-01
[\Date] startswith '1999'	Date starts with '1996'

When you include any of the following characters in an Impromptu expression you must precede the character with a backslash:

- a single quotation mark (or an apostrophe) in a string literal
- a backslash in a string literal
- a square bracket in a catalog reference

Notes

- Names are not case-sensitive, for example, 'startswith' is the same as 'StartsWith'.
- Spaces between operators are not required, for example, 'x+3*abs(y)' is the same as 'x + 3 * abs (y) '.
- Numeric constants must start with a digit, for example, '.3' becomes '0.3'.

For more information about Impromptu expressions, see the Reference section of Impromptu online help.

JavaScript Client Methods

JavaScript Client methods come with the JavaScript Client library.

The following methods can be used only in your HTML pages:

- FormatUserVar Method
- GetUserVar Method
- GetUserVarValues Method

Do not enclose this method in angle brackets (<%...%>) because these are client-side JavaScript methods.

Step

- To use these methods, you must type the following in your HTML file

  ```
  <script language= "javascript"
  src="/cognos/PowerPrompts/PowerPromptLib.js"></script>
  ```

FormatUserVar Method

Use to format the values of a variable according to the specified format string and list separator. The user sets this variable as part of completing the PowerPrompts application. The percentage sign (%) acts as the substitution character. Type the percentage sign twice (%%) to represent a percentage sign (%).

Syntax

```
FormatUserVar(varName,format,separator)
```

Parameters

varName: string

format: string

separator: string

Example

```
<html>
<head>
<script language="javascript" src="/cognos/PowerPrompts/PowerPromptLib.js"></script>
</head>
<body>
<form>
</form>
Show me values in Country:
<script language="javascript">
document.write(FormatUserVar("StateList","state = '%'"," AND "));
</script>
</body>
</html>
```

returns:

```
state = 'AK' AND state = 'VT' AND state = 'AL'
```

Related Topics

- "GetUserVar Method" (p. 96)
- "GetUserVarValues Method" (p. 96)
- "JavaScript Client Methods" (p. 95)

GetUserVar Method

Returns the value of a variable that the user sets as part of completing the PowerPrompts application. If there is more than one value associated with the variable, the values are separated with a comma.

Syntax

```
GetUserVar(varName)
```

Parameters

varName: string

Example

```
<html>
<head>
<script language="javascript" src="/cognos/PowerPrompts/PowerPromptLib.js"></script>
</head>
<body>
<form>
</form>
Show me values in Country:
<script language="javascript">
document.write(GetUserVar("country"));
</script>
</body>
</html>
```

Related Topics

- "FormatUserVar Method" (p. 95)
- "GetUserVarValues Method" (p. 96)
- "JavaScript Client Methods" (p. 95)

GetUserVarValues Method

Returns an array of the values for the named variable.

Syntax

```
GetUserVarValues(varName)
```

Parameters

varName: string

Example

```
<html>
<head>
<script language="javascript" src="/cognos/PowerPrompts/PowerPromptLib.js"></script>
</head>
<body>
<form>
</form>
Show me values in Country:
<script language="javascript">
document.write(GetUserVarValues("country code"));
</script>
</body>
</html>
```

Related Topics

- "FormatUserVar Method" (p. 95)
- "GetUserVar Method" (p. 96)
- "JavaScript Client Methods" (p. 95)

Index

Symbols

#else statement, 38
#endif statement, 38
#ifdef statement, 38
#include statement, 38

A

Access Manager
- retrieving user ID and password, 32

AddColumn, 90
AddConditionalFormat, 86
AddDataItem, 83
adding
- a script, 21
- graphics, 10
- Impromptu user class filter, 29
- link conditions, 17
- links, 16
- pages, 13

AddNamedCondition, 90
ADO, 31
AndFilterBy, 90
AndSummaryFilterBy, 91
apostrophe, 94
App object, 39
App.BackURL, 39
App.CurrentPage, 40
App.Errors, 40
App.FinalURL, 41
App.IsTestMode, 41
App.Path, 42
App.ReportScript, 42
App.RunDynamo, 43
App.Variables, 42
AppendCookie method, 65
AppendHeader method, 65
application
- testing, 23
- verifying, 23

ApplyStyle, 86, 89
ApplyTemplate, 84
AssociateColumn, 91

B

BackURL property, 39

C

catalog login, 23
changing
- default HTML editor, 11

Clear method, 65
ClearContent method, 66
ClearHeaders method, 66
Close method, 59
Cognos logical database, 18, 20
column title methods
- ApplyStyle, 89
- SetText, 89
- SetTextJustification, 89

conditions, 17
configuring
- Microsoft Web server, 10

connecting
- to the catalog, 23

Connection object, 43
Connection.Execute, 44
Contains method, 73
ContentType property, 64
Cookies property, 62
copyright, 2
Count property, 73
CreateObject method, 70
creating
- cascading picklist, 30
- different look for Upfront themes, 32
- dynamos, 19, 20
- new application, 12
- summary of choices, 28
- type-in picklist, 29

CurrentPage property, 40
CurrentRecordIndex property, 56

D

data source, 18, 20
database definition, 18
databases
- accessing using ADO, 31

default HTML editor, 11
default link, 15
deleting
- links, 17
- pages, 14

Description property, 79
document
- version, 2

drawing area, 12
dynamos, 18, 19, 20

E

editing pages, 14
Email property, 79
Entry page, 13
EOF property, 57
Error page, 13
errors
- application, 23
- script, 24

Errors property, 40
Execute method, 44
ExecuteCommand method, 78
Exit page, 13

F

Field object, 44
Fields property, 58
Fields.HTMLEncodedValue, 45
Fields.Index, 45
Fields.Name, 46
Fields.Type, 47
Fields.Value, 48
filtering
  reports for multiple values, 27
FinalURL property, 41
fixing script errors, 24
FormatNumber method, 70
FormatUserVar (JavaScript), 95

G

GetColumnTitle, 82
GetPageFragment method, 78
GetQuery, 82
GetReport, 82
GetReportObject, 83
GetUserVar, 96
GetUserVarValues, 96
GroupBy, 91

H

HorizontalAlign, 86
HorizontalAlignToColumn, 87
HTML editor, 11
HTML template, 11
HTMLEncode method, 71
HTMLEncodedValue property, 45

I

importing pages, 13
Impromptu expressions, 94
Impromptu user class filter, 29
include statement, 38
index operator, 49, 75
Index property, 45
information for PowerPrompts 6.0 users, 8
install PowerPrompts, 10
introduction to PowerPrompts, 7
IsTestMode property, 41
Item method, 74

J

JavaScript Client methods
  FormatUserVar, 95
Join method, 75

L

Language property, 76
link conditions, 17
links, 16
  deleting, 17
  overview, 15
Locale property, 77
logical database, 18, 20

M

MaxRecords property, 58
methods
  AddColumn, 51
  AndFilterBy, 51
  AndSummaryFilterBy, 52
  AppendCookie, 65
  AppendHeader, 65
  AssociateColumn, 52
  Clear, 65
  ClearContent, 66
  ClearHeaders, 66
  Close, 59
  Contains, 73
  CreateObject, 70
  Execute, 44
  ExecuteCommand, 78
  FormatNumber, 70
  GetColumnTitle, 82
  GetPageFragment, 78
  GetQuery, 82
  GetReport, 82
  GetReportObject, 83
  GetUserVar, 96
  GetUserVarValues, 96
  GroupBy, 53
  HTMLEncode, 71
  Item, 74
  Join, 75
  MoveNext, 60
  Open, 60
  Operator, 49, 75
  OrFilterBy, 53
  OrSummaryFilterBy, 54
  Redirect, 66
  RemoveColumn, 54
  RunDynamo, 43
  SetDistinct, 54
  SetPromptValue, 55
  SortBy, 55
  toString, 76
  URLDecode, 71
  URLEncode, 72
  Write, 66
  WriteFile, 67
  Writeln, 67
MoveNext method, 60
multiple values
  filter reports for, 27

N

Name property, 46
new application, 12

O

objects
  App, 39
  Connection, 43
  Field, 44
  Query, 49
  Recordset, 56
  Request, 62
  Response, 64
  Server, 69
  StringList, 72
  Upfront, 76
  User, 78
ODBC, 18, 20

Open method, 60
Operator method, 49, 75
OrFilterBy, 92
OrSummaryFilterBy, 92
overviews
  dynamos, 18
  links, 15
  script, 21

P

pages
  adding, 13
  deleting, 14
  editing, 14
  importing, 13
password, 32
Path property, 42
picklists
  adding Impromptu user class filter, 29
  creating cascading, 30
  creating type-in, 29
PowerPrompts, 7, 8, 10
PowerPrompts 6.0, 8
pre-defined constants, 38
preprocessor directives, 38
product
  version, 2
properties
  BackURL, 39
  ContentType, 64
  Cookies, 62
  Count, 73
  CurrentPage, 40
  CurrentRecordIndex, 56
  Description, 79
  Email, 79
  EOF, 57
  Errors, 40
  Fields, 58
  FinalURL, 41
  HTMLEncodedValue, 45
  Index, 45
  IsTestMode, 41
  Language, 76
  Locale, 77
  MaxRecords, 58
  Name, 46
  Path, 42
  ReportScript, 42
  ScriptTimeOut, 69
  ServerVariables, 62
  SQL, 50
  Telephone, 80
  Theme, 77
  Type, 47
  UserClass, 80
  UserClasses, 80
  UserName, 81
  Value, 48
  Variables, 42, 63

Q

Query, 49
query methods
  AddColumn, 90
  AddNamedCondition, 90
query methods *(cont'd)*
  AndFilterBy, 90
  AndSummaryFilterBy, 91
  AssociateColumn, 91
  GroupBy, 91
  OrFilterBy, 92
  OrSummaryFilterBy, 92
  RemoveColumn, 92
  SetMaxRows, 93
  SetPromptValue, 93
  SortBy, 94
Query object, 49
Query.AddColumn, 51
Query.AndFilterBy, 51
Query.AndSummaryFilterBy, 52
Query.AssociateColumn, 52
Query.GroupBy, 53
Query.OrFilterBy, 53
Query.OrSummaryFilterBy, 54
Query.RemoveColumn, 54
Query.SetDistinct, 54
Query.SetPromptValue, 55
Query.SortBy, 55
Query.SQL, 50

R

Recordset object, 56
Recordset.Close, 59
Recordset.CurrentRecordIndex, 56
Recordset.EOF, 57
Recordset.Fields, 58
Recordset.MaxRecords, 58
Recordset.MoveNext, 60
Recordset.Open, 60
Redirect method, 66
RemoveColumn, 92
RemoveReportObject, 84
report methods
  AddDataItem, 83
  ApplyTemplate, 84
  RemoveReportObject, 84
  SetListInsertCursor, 85
  SetPrimaryFrame, 85
report object methods
  AddConditionalFormat, 86
  ApplyStyle, 86
  HorizontalAlign, 86
  HorizontalAlignToColumn, 87
  SetText, 87
  SetTextJustification, 88
  VerticalAlign, 88
ReportScript property, 42
Request object, 62
Request.Cookies, 62
Request.ServerVariables, 62
Request.Variables, 63
requirements for PowerPrompts, 8
Response object, 64
Response.AppendCookie, 65
Response.AppendHeader, 65
Response.Clear, 65
Response.ClearContent, 66
Response.ClearHeaders, 66
Response.ContentType, 64
Response.Redirect, 66
Response.Write, 66
Response.WriteFile, 67

Response.Writeln, 67
retrieving
  user ID and password, 32
RunDynamo method, 43
running the application, 23

S

saving
  choices between sessions, 28
script, 24
  adding, 21
  overview, 21
script errors, 24
ScriptTimeOut property, 69
selecting a report, 22
Server object, 69
Server.CreateObject, 70
Server.FormatNumber, 70
Server.HTMLEncode, 71
Server.ScriptTimeOut, 69
Server.URLDecode, 71
Server.URLEncode, 72
ServerVariables property, 62
SetListInsertCursor, 85
SetMaxRows, 93
SetPrimaryFrame, 85
SetPromptValue, 93
SetText, 87, 89
SetTextJustification, 88, 89
setting
  default HTML editor, 11
  HTML template, 11
sharing
  JavaScript code, 38
single quotation mark, 94
SortBy, 55, 94
SQL property, 50
starting PowerPrompts, 10
string literal
  apostrophe, 94
  single quotation mark, 94
StringList object, 72
StringList.Contains, 73
StringList.Count, 73
StringList.Item, 74
StringList.Join, 75
StringList.toString, 76

T

Telephone property, 80
testing the application, 23
Theme property, 77
toString method, 76
Type property, 47

U

UNIX, 18
Upfront object, 76
Upfront themes
  creating different look for, 32
Upfront.ExecuteCommand, 78
Upfront.GetPageFragment, 78
Upfront.Language, 76
Upfront.Locale, 77
Upfront.Theme, 77
URLDecode method, 71
URLEncode method, 72
user choices
  creating summary, 28
  saving between sessions, 28
user ID, 32
User object, 78
User.Description, 79
User.Email, 79
User.Telephone, 80
User.UserClass, 80
User.UserClasses, 80
User.UserName, 81
UserClass property, 80
UserClasses property, 80
UserName property, 81
using
  ADO, 31

V

Value property, 48
Variables property, 42, 63
verify the application, 23
version
  product, 2
VerticalAlign, 88
viewing a generated script, 24
virtual directories, 10

W

web server, 10
Write method, 66
WriteFile method, 67
Writeln method, 67