SYBASE®

JSP Target Reference

PowerBuilder®

11.0

DOCUMENT ID: DC37778-01-1100-01

LAST REVISED: May 2007

Sybase, Inc., One Sybase Drive, Dublin, CA 94568.

Contents

About This Book

Audience

This book is for programmers who will be using PowerBuilder® to build Java Server Pages (JSP) applications.

How to use this book

This book describes classes that make up the Web Target object model and its 4GL extensions that you can use with JSP applications. The descriptions includes syntax, usage notes, and examples for the object model class methods. Server-side events from which you can call these methods are also described.

Related documents

Working with JSP Targets
DataWindow Reference Guide

Other sources of information

Use the Sybase Getting Started CD, the SyBooks CD, and the Sybase Product Manuals Web site to learn more about your product:

- The Getting Started CD contains release bulletins and installation guides in PDF format, and may also contain other documents or updated information not included on the SyBooks CD. It is included with your software. To read or print documents on the Getting Started CD, you need Adobe Acrobat Reader, which you can download at no charge from the Adobe Web site using a link provided on the CD.
- The SyBooks CD contains product manuals and is included with your software. The Eclipse-based SyBooks browser allows you to access the manuals in an easy-to-use, HTML-based format.

  Some documentation may be provided in PDF format, which you can access through the PDF directory on the SyBooks CD. To read or print the PDF files, you need Adobe Acrobat Reader.

  Refer to the *SyBooks Installation Guide* on the Getting Started CD, or the *README.txt* file on the SyBooks CD for instructions on installing and starting SyBooks.
- The Sybase Product Manuals Web site is an online version of the SyBooks CD that you can access using a standard Web browser. In addition to product manuals, you will find links to EBFs/Maintenance, Technical Documents, Case Management, Solved Cases, newsgroups, and the Sybase Developer Network.

To access the Sybase Product Manuals Web site, go to Product Manuals at http://www.sybase.com/support/manuals/.

If you need help

Each Sybase installation that has purchased a support contract has one or more designated people who are authorized to contact Sybase Technical Support. If you cannot resolve a problem using the manuals or online help, please have the designated person contact Sybase Technical Support or the Sybase subsidiary in your area.

CHAPTER 1

JSP Target Classes and Objects

About this chapter

This chapter describes the classes in the Web Target server-side object model, including information about their properties and methods. Unless a target type is specifically mentioned in a description, the classes and objects are available to 4GL and non-4GL JSP targets.

Contents

PSArgClass

Produces a parameter string with name and value pairs that you can use in a psPage.Redirect call with 4GL targets. This class uses a default constructor with no arguments.

Methods

PSArgClass method	Description
addArg	Adds a name and value pair to be passed as a parameter. The value passed can be of the following datatypes: boolean, byte, char, double, int, float, long, Object, short, or String.
getCharacterEncoding	Returns the name of charset used by the PSArgClass object.
GetParameterString	Returns the URL encoded value for the string to be passed as a parameter.
setCharacterEncoding	Sets the name of the charset used by the PSArgClass object.

PSButtonClass

Objects of this class are server-side representations of buttons on the client side.

If you enable the 4GL server-side event model for your JSP Web pages, you can bind parameters or components to objects of this class in the PowerBuilder HTML editor. The Server Side Scriptable check box on the button control property sheet must also be selected.

Properties

PSButtonClass property	Datatype	Description
name	String	The name of the control. Because this is a read-only property, changing the name in a server-side script has no effect other than potentially causing confusion.
enabled	boolean	Whether or not the control allows focus. This property works only in browsers that support the DISABLED attribute.
value	String	The label for the button control.
visible	boolean	Whether or not the client control is generated. If not visible, there is no access to the client control.

Events

PSButtonClass event	Description
ServerAction	This event is triggered when a button action was the trigger for a page refresh. This event happens after all validation and data binding has occurred.

PSCheckBoxClass

Objects of this class are server-side representations of CheckBox controls on the client side.

If you enable the 4GL server-side event model for your JSP Web pages, you can bind parameters or components to objects of this class in the PowerBuilder HTML editor. The Server Side Scriptable check box on the CheckBox control property sheet must also be selected.

Properties

PSCheckBoxClass property	Datatype	Description
name	String	The name of the control. Because this is a read-only property, changing the name in a server-side script has no effect other than to potentially cause confusion.
enabled	boolean	Whether or not the control can be edited. This property works only in browsers that support the DISABLED attribute.
value	boolean	The state of the check box. This is not the string value that is a required attribute of the Input element in an HTML form.
visible	boolean	Whether or not the client control is generated. If not visible, there is no access to the client control.

Events

PSCheckBoxClass event	Description
ItemChanged	This event is triggered when the value of the control has changed and passed validation.

PSCommandClass

Provides a mechanism for storing and re-executing a SQL statement. You must assign a variable of the PSCommandClass type before you can create an instance of the object or call methods on it.

Syntax

PSCommandClass (*strSql*, *conn*)

Constructors

PSCommandClass constructor	Datatype	Description
strSql	String	SQL statement you want to execute
conn	PSConnectionClass	Object you use to connect to the database where you want to execute the SQL statement

Methods

PSCommandClass method	Description
Execute	Executes a SQL command
SetSQL	Obsolete method that was previously used for PowerDynamo or ASP targets

PSConnectionClass

Allows you to connect to a database, obtain or clear database errors, and create database cursors. You must assign a variable of the PSConnectionClass type before you can create an instance of the object or call methods on it.

Syntax

Syntax with user name and password

PSConnectionClass (pageContext, *Driver*, *URL*, *user*, *password*, {*bTrace*})

Syntax with database properties

PSConnectionClass (pageContext, *Driver*, *URL*, *Properties*, {*bTrace*})

Constructors

PSConnectionClass constructor	Datatype	Description
pageContext	PageContext (javax.servlet .jsp class)	The implicit pageContext object available to JSP targets.

PSConnectionClass constructor	Datatype	Description
Driver	String	The name of the JDBC driver used to connect to the database.
URL	String	The location of the database to which you want to connect. The database URL is obtained from the database JDBC driver documentation.
user	String	The user name that the object uses to connect to the specified database.
password	String	The password that the object uses to connect to the specified database.
Properties	String	Any properties that your JDBC driver uses to connect to the database. If properties are defined, you must also define the user ID and password in the properties that you list.
bTrace (Optional)	boolean	Allows tracing if set to true. The default is false.

Methods

PSConnectionClass method	Description
ClearError	Clears the list of error objects
CreateCommand	Creates a named object that represents a SQL statement
CreateCursor	Creates a database cursor
GetError	Returns the first error object

PSConnectionParmsClass

Specifies the database connection parameters required for a Web DataWindow control to connect to a database. The object does not connect to the database.

Unless you use the two-argument syntax, you need to be familiar with the connection parameters for your database system before using this object. The *Connecting to Your Database* book provides information about making database connections.

Syntax

Syntax with single argument

PSConnectionParmsClass(*connectString*)

Syntax with two arguments

PSConnectionParmsClass(*DBProfile*, *Prop*)

Syntax with multiple arguments

PSConnectionParmsClass(*connectString*, *username*, *password*, *dbms*, *lock*, *database*, *serverName*)

Constructors

PSConnectionParms Class constructor	Datatype	Description
connectString	String	The connection parameters required to connect to the database. This string is specific to the database driver for your database DBMS. It must not include spaces that are not part of the string value.
DBProfile	String	The connection parameters defined in the *database.properties* file in the \WEB-INF\classes directory. This file is created when you build the JSP target and its content is taken from the database profiles defined in PowerBuilder.
Prop	Boolean	This value is always “true”. It is used to distinguish the constructor syntax with the DBProfile argument from the syntax with the *connectString* argument.
username	String	The user name that the object uses to connect to the specified database.
password	String	The password that the object uses to connect to the specified database.
dbms	String	The connection mechanism used to connect to the database.
lock	String	The lock value required by your database to prevent concurrent transactions for interfering with each other and compromising the data in the database.
database	String	The name of the database.
serverName	String	The name of the server that runs the database.

Examples

The following example creates the connection object dbConn using the syntax with a single *connectString* argument:

```
PSConnectionParmsClass  dbConn = new
PSConnectionParmsClass("ConnectString='DSN=EAS Demo DB
V11;UID=dba;PWD=sql',ConnectOption='SQL_DRIVER_CONNECT
,SQL_DRIVER_NOPROMPT'" );
```

PSCursorClass

Provides access to a SQL result set.

Syntax

PSCursorClass(*ResSet*)

Constructors

PSCursorClass constructor	Datatype	Description
ResSet	ResultSet	Result set object returned by the SQL query

Methods

PSCursorClass method	Description
EOF	Determines whether the end of a cursor has been reached
GetColumnCount	Retrieves the number of columns in a cursor
GetColumn<DataType>	Retrieves the value of a column in a cursor
GetColumnLength	Retrieves the length of a column in a cursor
GetColumnName	Retrieves the name of a column in a cursor
GetColumnType	Retrieves the SQL type of a column in a cursor
GetColumnTypeName	Retrieves the database-specific type of a column in a cursor
GetPrecision	Retrieves the number of decimal digits of a column in a cursor
GetResultSet	Retrieves the result set for a cursor
GetResultSetMetaData	Retrieves the metadata result set
GetRowCount	Retrieves the number of rows in a cursor
GetScale	Retrieves the number of digits to right of the decimal point for a column in a cursor

PSCursorClass method	Description
GetValue	Obsolete method that was previously used for PowerDynamo or ASP targets
Move	Moves to an absolute row in a cursor
MoveFirst	Moves to the first row in a cursor
MoveLast	Moves to the last row in a cursor
MoveNext	Moves to the next row in a cursor
MovePrevious	Moves to the previous row in a cursor

PSDataWindowClass

Creates a new object for a Web DataWindow control. This object lets you add a Web DataWindow object (that you create in PowerBuilder or InfoMaker) to your page.

Adding a Sybase Web DataWindow DTC to an HTML page creates an object of type PSDataWindowClass. If the page is 4GL-enabled, an object of type PSWebDataWindowClass is created instead.

Syntax

PSDataWindowClass(pageContext, *request*, *objectName*, *ServerSideStateManagement*, *jaguarConnection*, *sourceLocation*, *dbConnection*, {*lPageSize*})

Constructors

PSDataWindowClass constructor	Datatype	Description
pageContext	PageContext	Implicit object created by the JSP server to handle page requests.
request	HttpServletRequest	Object created by servlet container for HTTP requests.
objectName	String	The name of the client-side control. If you do not specify a name, htmlDW is used as the default object.

PSDataWindowClass constructor	Datatype	Description
ServerSideStateManagement	boolean	Specifies where the database state is managed. Values are: • **true** The server manages the database state. A reference to the server component is saved and retrieved from the session object (based on the name of the Web DataWindow object). • **false** (default) The client manages the database state.
jaguarConnection	String	The connection information needed to connect to EAServer.
sourceLocation	String	The location of the DataWindow object. If this property is null (default), the server component must encapsulate the identity of the source.
dbConnection	String	The database connection properties. If this property is null (default), the server component must encapsulate the database connection properties.
lPageSize (Optional)	String	The size of the page: • **0** Indicates that all rows retrieved from the database will be generated. • **-1** (default) Indicates that the size of the page is specified in the definition for the Web DataWindow control. • **Any positive integer** Specifies the number of rows that will be passed to, and therefore contained in, the Web DataWindow control.

Properties

PSDataWindowClass property	Datatype	Description
Component	Object	Represents a reference to a server component for a Web DataWindow control. For information about the server component, see the *DataWindows Programmer's Guide*.
RetrievalArgs [ ]	String	An array of arguments used to retrieve data from the database. You can specify these retrieval arguments or use the FillRetrievalArgs method to do so.

Methods

PSDataWindowClass method	Description
FillRetrievalArgs	Fills in the array that stores the retrieval arguments.
Generate	Generates the DataWindow as HTML.
GenerateXHTML	Generates the DataWindow as XHTML.
GenerateXMLWeb	Generates the DataWindow as XML.
SetColumnLink	Establishes a link for a column that is passed to the Web DataWindow control.
SetWeight	Identifies the type of functionality included on your HTML page. (As you include more functionality on your page, the size of the control increases.)

Examples

The following example shows how to define a new Web DataWindow object named webDW. The Web DataWindow object uses a client-side control named webDW, and a previously defined EAServer connection object named jagConn:

```
PSJaguarConnection jagConn = new
PSJaguarConnection("my-desktop:9000", "jagadmin", "",
"DataWindow/HTMLGenerator110", false);

PSDataWindowSourceClass dwSource = new
PSDataWindowSourceClass("d:\\test\\appl.pbl",
"dw_dept");
```

```
PSConnectionParmsClass  dbConn = new
PSConnectionParmsClass("ConnectString='DSN=EAS Demo DB
V11;UID=dba;PWD=sql',ConnectOption='SQL_DRIVER_CONNECT
,SQL_DRIVER_NOPROMPT'" );

PSDataWindowClass webDW = new PSDataWindowClass
(pageContext, request, "webDW", false, jagConn,
dwSource, dbConn, 10);
```

PSDataWindowSourceClass

Creates a new source parameter object. The object specifies an existing definition of a Web DataWindow control.

Syntax

PSDataWindowSourceClass(*sourceFileName*, *dwName*, *stringSourceURL*)

Constructors

PSDataWindowSourceClass constructor	Datatype	Description
sourceFileName	String	The URL of the source format for a the definition of a Web DataWindow control that is deployed on a Web server. The object retrieves this definition and passes it to the server component. A null setting indicates that the source format is deployed on a Web server.
dwName	String	The name of the file on the server that stores the definition for the Web DataWindow control. If a PSR or SRD file stores the definition, you do not need to specify the name for the definition of the DataWindow control. If a PBD or PBL stores the definition for the DataWindow object, you must specify the *dwName* property. The server component of the Web DataWindow control uses the path information from the *dwName* property to locate the file.

PSDataWindowSourceClass constructor	Datatype	Description
stringSourceURL	String	The name of a DataWindow control stored in a PBD or PBL. Required if *dwName* has a PBD or PBL extension, but ignored for file names that have other extensions.

Examples

The following example creates a new source parameter object named dwSource that uses the *appl.pbl* library on a local drive and the DataWindow control dw_dept:

```
PSJaguarConnection jagConn = new
PSJaguarConnection("my-desktop:9000", "jagadmin", "",
"DataWindow/HTMLGenerator90", false);

PSDataWindowSourceClass dwSource = new
PSDataWindowSourceClass("d:\\test\\appl.pbl",
"dw_dept");

PSConnectionParmsClass  dbConn = new
PSConnectionParmsClass("ConnectString='DSN=EAS Demo DB
V11;UID=dba;PWD=sql',ConnectOption='SQL_DRIVER_CONNECT
,SQL_DRIVER_NOPROMPT'" );

PSDataWindowClass webDW = new
PSDataWindowClass(pageContext, request, "webDW", false,
jagConn, dwSource, dbConn, 10);
```

PSDocumentClass

Represents the current document in a Web application.

Unique object
A unique instance of the class called psDocument is created for you automatically when you deploy your application. Therefore, you do not need to instantiate PSDocumentClass. In your scripts, you will always refer to psDocument.

Methods

PSConnectionClass method	Description
File	Returns the file name for the document.
GetEnv	Retrieves the value of a server environment variable.
GetParam	Retrieves a parameter passed to the current page.
Path	Returns the path portion of the URL for the document.
Redirect	Redirects the current request to another URL.
Site	Returns the domain name for the document.
Write	Writes an output string to the document.
WriteLn	Writes an output string to the document that ends with a line break. The line break affects the HTML source, not the final HTML output.

PSDropDownListClass

Objects of this class are server-side representations of DropDownListBox controls on the client side.

If you enable the 4GL server-side event model for your JSP Web pages and if the Server Side Scriptable check box on the control property sheet is selected, you can bind parameters or components to objects of this class in the PowerBuilder HTML editor.

Properties

PSDropDownListClass property	Datatype	Description
name	String	The name of the control. Because this is a read-only property, changing the name in a server-side script has no effect other than potentially causing confusion.
enabled	boolean	Whether or not the control can be edited. This property works only in browsers that support the DISABLED attribute.

PSDropDownListClass property	Datatype	Description
value	String	The label for the drop-down list control.
visible	boolean	Whether or not the client control is generated. If not visible, there is no access to the client control.

Events

PSDropDownListClass event	Description
ItemChanged	This event is triggered when the value of the control has changed and passed validation.

PSErrorClass

Provides access to errors captured by the application server. The error information provided is server specific.

Syntax

PSErrorClass (*code*, *info*)

Constructors

PSErrorClass constructor	Datatype	Description
code	int	The error code returned from the connection object
info	String	The error message returned from the connection object

Methods

PSConnectionClass method	Description
GetCode	Returns the code associated with the current error object
GetMessage	Returns the message associated with the current error object

PSConnectionClass method	Description
GetNextError	Returns the next error object, if one exists

PSImageClass

Objects of this class are server-side representations of images on the client side.

If you enable the 4GL server-side event model for your JSP Web pages and if the Server Side Scriptable check box on the Input Properties dialog box is selected, you can bind parameters or components to objects of this class in the PowerBuilder HTML editor.

Properties

PSImageClass property	Datatype	Description
enabled	boolean	Whether or not the control allows focus. This property works only in browsers that support the DISABLED attribute.
value	String	The label for the Image Button control.
visible	boolean	Whether or not the client control is generated. If not visible, there is no access to the client control.

Events

PSImageClass event	Description
ServerAction	This event is triggered on an Image Button control when a user action is the trigger for a page refresh. This event happens after all validation and data binding has occurred.

PSJaguarConnection

Specifies the connection information used to connect to a component on EAServer. This component provides interoperability between the Web DataWindow control and page servers that support Java.

Syntax

Syntax specifying server name and properties

PSJaguarConnection(*serverName*, *userId*, *password*, *componentName*, *bOneTrip*)

Syntax specifying an EAServer profile

PSJaguarConnection(*profileName*, *componentName*, *bOneTrip*)

Constructors

PSJaguarConnection constructor	Datatype	Description
serverName	String	The name of the server that runs the component for your Web DataWindow control. The syntax for this entry is serverName:port.
profileName	String	The server connection parameters defined in the *jaguar.properties* file in the \WEB-INF\classes directory. This file is created when you build the JSP target and its content is taken from the EAServer profiles defined in PowerBuilder.
userId	String	The username that the object uses to connect to the specified EAServer. The default is Jaguar.
password	String	The password that the object uses to connect to the specified EAServer. The default is guest.
componentName	String	The name of the Web DataWindow server component on EAServer that uses this connection. The default is DataWindow/nv_html_data_window.

PSJaguarConnection constructor	Datatype	Description
bOneTrip	String	Specifies how many round trips are made to the server. Lets you specify that one method, rather then several, perform the setup and generate the HTML for your Web DataWindow. • **true** One trip is made to the server to set up and generate HTML. If set to true, the DataWindow must have a name configured. Setting this parameter can help increase performance for a custom component that has already loaded a DataWindow object. • **false** (default) Methods for setting up your DataWindow and generating the HTML are coded separately and require more than one trip to the server.

Examples

The following example defines a connection to the EAServer named "my-desktop" using the port 9000:

```
PSJaguarConnection jagConn = new
PSJaguarConnection("my-desktop:9000", "jagadmin", "",
"DataWindow/HTMLGenerator100", false);
```

PSLinkClass

Objects of this class are server-side representations of client-side hyperlinks or HTML “A” tags.

If you enable the 4GL server-side event model for your JSP Web pages and if the Server Side Scriptable check box on the control property sheet is selected, you can bind parameters or components to objects of this class in the PowerBuilder HTML editor.

Properties

PSLinkClass property	Datatype	Description
value	String	The destination URL

PSNamedConnectionParmsClass

Specifies the database connection information required to connect to a named database. The object does not connect to the database.

Obsolete method
This method was designed for use with ASP and PowerDynamo Web targets that are no longer supported.

psPage

The psPage object is a global object that resides on the server for 4GL JSP targets. It controls the event model and encapsulates the server-side object model, including a representation of all the form controls in the object model. You must enable the 4GL Web server-side event model in order to create the psPage object.

Properties

psPage property	Datatype	Description
errors[ ]	Vector	The Vector datatype is a collection of PageError objects. PageError is a value class with 3 string attributes: location, cause, and message.
showErrorsOnPage	boolean	Specifies whether errors contained in the errors[ ] array are displayed on the page when the page is generated.

psPage property	Datatype	Description
showErrorsAtTop	boolean	Specifies whether errors are displayed at the top or the bottom of the page.
showErrorsInAlert	boolean	Specifies whether errors are displayed in a client-side alert box after the page has completed loading.
pageName	String	The current name of the page.
firstTime	boolean	Indicates whether this is the first time the page was called.
hadValidationError	boolean	Indicates whether a validation error occurred. If an action fails validation and then you change this property to false, the action will occur despite the validation error.
didRedirect	boolean	Indicates whether psPage.Redirect has been called. If it has, generation will not occur.
doTrace	boolean	Indicates whether tracing is enabled. This property is obsolete. Use the SetTrace and IsTrace methods instead.

Properties you should not change

Changes to the firstTime, hadValidationError, and didRedirect property values are generated dynamically. It is not recommended that you change these values directly in code.

Events

psPage event	Occurs	Order of occurrence
RequestStart	At the beginning of a request	1
FirstTime	The first time a page is loaded	2
BeforeBinding	Just before binding starts	3
Validate	To allow whole page validation	4
ValidationError	If any validate on the page fails	5
AfterBinding	Just after binding completes	6
BeforeAction	Just before the action is performed	7
AfterAction	Just after the action is performed	8
BeforeGenerate	Just before the page is generated	9
AfterGenerate	After all generation is complete	10

psPage event	Occurs	Order of occurrence
RequestFinish	After all generation is complete	11
ServerError	When ReportError() is called	When invoked

Validation and ItemChanged events for controls on the page occur between the psPage BeforeBinding and AfterBinding events. ServerAction events for controls on the page occur between the psPage BeforeAction and AfterAction events.

Methods

psPage method	Description
Alert	Causes client-side alert box to display when the page is finished loading
IsTrace	Indicates whether tracing is enabled
Redirect	Redirects the client's browser to another page
ReportError	Indicates whether a server-side error has occurred
SetTrace	Turns tracing on and off
TestCompError	Tests for errors on EAServer component methods
Trace	Adds a message to the internal trace buffer
TraceIndent	Increases the indent level to indicate nesting
TraceOutdent	Decreases the indent level to indicate nesting
WriteErrorsToDocument	Writes current errors at the current place in the page

PSPasswordClass

Objects of this class are server-side representations of text box controls on the client side.

If you enable the 4GL server-side event model for your JSP Web pages and if the Server Side Scriptable check box on the text box control property sheet is selected, you can bind parameters or components to objects of this class in the PowerBuilder HTML editor.

Properties

PSPasswordClass property	Datatype	Description
name	String	The name of the control. Because this is a read-only property, changing the name in the server-side script has no effect other than potentially causing confusion.
enabled	boolean	Whether or not the control can be edited. This property works only in browsers that support the DISABLED attribute.
value	String	The text in the password control.
visible	boolean	Whether or not the client control is generated. If not visible, there is no access to the client control.

Events

PSPasswordClass event	Description
ItemChanged	This event is triggered when the value of the control has changed and passed validation.
Validate	This event occurs when the client changes a value of a text control. The event is passed the new value.
ValidationError	This event is triggered when the Validate event fails. It is passed the user-entered value from the Validate event.

PSRadioGroupClass

Objects of this class are server-side representations of RadioButton controls on the client side.

If you enable the 4GL server-side event model for your JSP Web pages and if the Server Side Scriptable check box on the control property sheet is selected, you can bind parameters or components to objects of this class in the PowerBuilder HTML editor.

Properties

PSRadioGroup Class property	Datatype	Description
enabled	boolean	Whether the control can be edited. This property works only in browsers that support the DISABLED attribute.
value	String	The value of the selected radio button in the group.
visible	boolean	Whether the client control is generated. If not visible, there is no access to the client control.

Events

PSRadioGroup Class event	Description
ItemChanged	This event is triggered when the value of a control has changed and passed validation.

PSServerClass

Provides a variety of basic services for a Web application.

Unique object
A unique instance of the class called psServer is created automatically when you deploy your application. Therefore, you do not need to instantiate psServer. In your scripts, always refer to psServer as the object instance of this class.

Methods

PSServerClass method	Description
CreateConnection	Creates a new database connection

PSServerClass method	Description
GetConnection	Obsolete method that was previously used for PowerDynamo or ASP targets
MapPath	Obsolete method that was previously used for PowerDynamo or ASP targets
ObjectModelType	Identifies the application server
ObjectModelVersion	Returns the version of the Web Target object model you are using
Type	Identifies the Web server
URLEncode	Applies URL encoding rules to a string
Version	Returns the version of the Web server

PSSessionClass

Manages data that needs to persist across pages in a Web application.

Unique object
A unique instance of the class called psSession is created automatically when you deploy your application. Therefore, you do not need to instantiate psSession. In your scripts, always refer to psSession as the object instance of this class.

Methods

PSServerClass method	Description
Abandon	Causes a session object to be discarded
GetValue	Retrieves the value of a session variable
SetValue	Sets the value of a session variable

PSStaticTextClass

Objects of this class allow an arbitrary piece of text to be manipulated from server-side scripts in 4GL JSP targets. The value of the text cannot be changed on the client side. You can insert an object of this class only on a page that is server scriptable.

Properties

PSStaticTextClass property	Datatype	Description
name	String	The name of the control. Because this is a read-only property, changing the name in a server-side script has no effect other than potentially causing confusion.
value	String	The text to be displayed (can be HTML).
visible	boolean	Whether or not the client control is generated. If not visible, there is no access to the client control.

Events

PSStaticTextClass event	Description
ServerAction	Scripting this event causes a form submit to be scripted for an onClick event. Not scripting this event turns this control into an HTML SPAN tag.

PSTextAreaClass

Objects of this class are server-side representations of TextArea controls on the client side.

If you enable the 4GL server-side event model for your JSP Web pages and if the Server Side Scriptable check box on the control property sheet is selected, you can bind parameters or components to objects of this class in the PowerBuilder HTML editor.

Properties

PSTextAreaClass property	Datatype	Description
name	String	The name of the control. Because this is a read-only property, changing the name in a server-side script has no effect other than potentially causing confusion.
enabled	boolean	Whether or not text in the control can be edited. This property works only in browsers that support the DISABLED attribute.
value	String	The text in the edit control.
visible	boolean	Whether or not the client control is generated. If not visible, there is no access to the client control.

Events

PSTextAreaClass event	Description
ItemChanged	This event is triggered when the value of the control has changed and passed validation.
Validate	This event occurs when the client changes a value of a text control. The event is passed the new value.
ValidationError	This event is triggered when the Validate event fails. It is passed the user-entered value from the Validate event.

PSTextClass

Objects of this class are server-side representations of SingleLineEdit controls on the client side.

If you enable the 4GL server-side event model for your JSP Web pages and if the Server Side Scriptable check box on the control property sheet is selected, you can bind parameters or components to objects of this class in the PowerBuilder HTML editor.

Properties

PSTextClass property	Datatype	Description
name	String	The name of the control. Because this is a read-only property, changing the name in a server-side script has no effect other than potentially causing confusion.
enabled	boolean	Whether or not the text in the control can be edited. This property works only in browsers that support the DISABLED attribute.
value	String	The text in the edit control.
visible	boolean	Whether or not the client control is generated. If not visible, there is no access to the client control.

Events

PSTextClass event	Description
ItemChanged	This event is triggered when the value of the control has changed and passed validation.
Validate	This event occurs when the client changes a value of a text control. The event is passed the new value.
ValidationError	This event is triggered when the Validate event fails. It is passed the user-entered value from the Validate event.

PSWebDataWindowClass

Objects of this class are server-side representations of Web DataWindow controls on 4GL Web pages. For non-4GL Web pages, use PSDataWindowClass objects instead.

Syntax

PSWebDataWindowClass(*objectName*, *ServerSideStateManagement*, *jaguarConnection*, *sourceLocation*, *dbConnection*, {*lPageSize*})

Constructors

PSWebDataWindowClass constructor	Datatype	Description
objectName	String	The name of the client-side control. By default, the name is set to dw_1.
ServerSideStateManagement	boolean	Specifies where the database state is managed: • **true** The server manages the database state. A reference to the server component is saved and retrieved from the session object (based on the name of the Web DataWindow object). • **false** (default) The client manages the database state.
jaguarConnection	String	The connection information needed to connect to EAServer. An EAServer profile must be defined.
sourceLocation	String	The location of the DataWindow object. If this property is null (default), the server component must encapsulate the identity of the source.
dbConnection	String	The database connection properties. If this property is null (default), the server component must encapsulate the database connection properties.
lPageSize (optional)	String	A positive integer specifies the number of rows that will be passed to, and therefore contained in, the Web DataWindow control.

Events

PSWebDataWindow Class event	Description
AfterAction	Triggered just after the call to SetAction on the server component
AfterRetrieve	Triggered just after the call to Retrieve on the server component
AfterUpdate	Triggered just after the call to Update on the server component

PSWebDataWindow Class event	Description
BeforeAction	Triggered just before SetAction is called on the server component
BeforeRetrieve	Triggered just before Retrieve is called on the server component
BeforeUpdate	Triggered just before Update is called on the server component
OnDBError	Triggered if a database error occurs during processing
Validate	Triggered immediately after the context is restored in the server component
ValidationError	Triggered if the *webdw*.Validate event fails

Methods

See the *DataWindow Reference* for more information about these methods:

PSWebDataWindowClass method	Description
ClearValues	Deletes all items from a value list or code table associated with a DataWindow column
Create	Creates a DataWindow object using DataWindow source code and puts that object in the specified DataWindow control
DeletedCount	Reports the number of rows that have been marked for deletion in the database
DeleteRow	Deletes a row from the DataWindow control
Describe	Reports the values of properties of a DataWindow object and controls within the DataWindow object
Filter	Displays rows in a DataWindow that pass the current filter criteria
FilteredCount	Reports the number of rows that are not displayed in the DataWindow because of the current filter criteria
Find	Finds the next row in a DataWindow in which data meets a specified condition
FindGroupChange	Searches for the next break for the specified group
Generate	Generates the DataWindow as HTML
GenerateXHTML	Generates the DataWindow as XHTML
GenerateXMLWeb	Generates the DataWindow as XML

PSWebDataWindowClass method	Description
GetColumn	Obtains the number of the current column
GetColumnName	Obtains the name of the current column
GetFormat	Obtains the display format assigned to a column in a DataWindow control
GetItemDate	Gets data of type Date from the specified buffer of a DataWindow control
GetItemDateTime	Gets data of type DateTime from the specified buffer of a DataWindow control
GetItemFormattedString	Gets and formats data of type String from the specified buffer of a DataWindow control or DataStore object.
GetItemNumber	Gets numeric data from the specified buffer of a DataWindow control
GetItemStatus	Reports the modification status of a row or a column within a row
GetItemString	Gets data of type String from the specified buffer of a DataWindow control
GetItemTime	Gets data of type Time from the specified buffer of a DataWindow control
GetItemUnformattedString	Gets unformatted data of type String from the specified buffer of a DataWindow control or DataStore object.
GetRow	Reports the number of the current row in a DataWindow control
GetValidate	Obtains the validation rule for a column in a DataWindow
GetValue	Obtains the value of an item in a value list or code table associated with a column in a DataWindow
GroupCalc	Recalculates the breaks in the grouping levels in a DataWindow
ImportString	Inserts data into a DataWindow control from tab-delimited data in a string
InsertRow	Inserts a row in a DataWindow
ModifiedCount	Reports the number of rows that have been modified but not updated in a DataWindow
Modify	Modifies a DataWindow object by applying specifications (given as a list of instructions) that change the DataWindow object's definition

PSWebDataWindowClass method	Description
ReselectRow	Accesses the database to retrieve values for all columns that can be updated and refreshes all timestamp columns in a row in a DataWindow control
Reset	Clears all the data from a DataWindow control
ResetUpdate	Clears the update flags in the primary and filter buffers and empties the delete buffer of a DataWindow
Retrieve	Retrieves rows from the database for a DataWindow control
RowCount	Obtains the number of rows that are currently available in a DataWindow control
RowsDiscard	Discards a range of rows in a DataWindow control
SaveAs	Saves the contents of a DataWindow in the format you specify
SetColumn	Sets the current column in a DataWindow control
SetColumnLink	Specifies information used for constructing hyperlinks for data in a column in generated HTML
SetDetailHeight	Sets the height of each row in the specified range to the specified value
SetDWObject	Specifies the DataWindow library and object that the Web DataWindow server component will use for generating HTML
SetFilter	Specifies filter criteria for a DataWindow control
SetFormat	Specifies a display format for a column in a DataWindow control
SetItem	Sets the value of a row and column in a DataWindow control to the specified value
SetItemDate	Sets the value of a row and column in a DataWindow control to the specified value
SetItemDateTime	Sets the value of a row and column in a DataWindow control to the specified value
SetItemNumber	Sets the value of a row and column in a DataWindow control to the specified value
SetItemStatus	Changes the modification status of a row or a column within a row
SetItemString	Sets the value of a row and column in a DataWindow control to the specified value

PSWebDataWindowClass method	Description
SetItemTime	Sets the value of a row and column in a DataWindow control to the specified value
SetPosition	Moves a control within the DataWindow to another band or changes the front-to-back order of controls within a band
SetRow	Sets the current row in a DataWindow control
SetServerServiceClasses	Tells the server component to trigger custom events defined in user objects for data validation
SetSort	Specifies sort criteria for a DataWindow control
SetSQLSelect	Specifies the SQL SELECT statement for a DataWindow control
SetValidate	Sets the input validation rule for a column in a DataWindow control
SetValue	Sets the value of an item in a value list or code table for a column in a DataWindow control
SetWeight	Specifies the types of JavaScript code that will be included in the generated HTML
Sort	Sorts the rows in a DataWindow control using the DataWindow's current sort criteria
Update	Updates the database with the changes made in a DataWindow control

CHAPTER 2

4GL Web Target Events

About this chapter

This chapter describes server-side events for the psPage object. They are included in the second drop-down list of the script window in the Web target Page view—but only when the Enable 4GL Web Server Side Event Model is selected on the Page tab of the Page Properties dialog box. These events display in blue to distinguish them from client-side events, which are listed in black.

Contents

AfterAction

Description	Occurs after all actions have been performed but before generation of the HTML page.
Applies to	psPage object
Arguments	None
Return codes	Boolean
Usage	This event occurs only after a self-navigation, making this event a good place to call the Redirect method if that was not done in a control's ServerAction event. It is also a place where an action method on an EAServer component can be called to change the internal state before the get portion of data binding is run.

Error processing
Because this event is triggered before generation occurs, psDocument.Write cannot be used for reporting errors. Instead, you can use the ReportError method on the psPage object to trigger the ServerError event. The error will then be added to the error log, depending on the ServerError return value.

Examples

This statement in the AfterAction event changes the Web page that displays in the client-side browser:

```
psPage.Redirect("My_WebPage.htm");
```

See also

AfterAction for PSWebDataWindowClass objects
ServerAction

AfterBinding

Description	Occurs after the controls have been bound to the input data and all validation has been done, but before any actions are performed.
Applies to	psPage object
Arguments	None
Return codes	None
Usage	This event is equivalent to the BeforeAction event but occurs before it. This event enables you to semantically separate logic related to post-processing of the data binding from logic related to the pre-processing of actions.

Error processing
Because this event is triggered before generation occurs, you cannot use psDocument.Write to report errors. Instead, use the ReportError method on the psPage object to trigger the ServerError event. The error will then be added to the error log, depending on the ServerError return value.

Examples

This example sets a trace for all events after binding. The trace messages appear at the top of the page in the client browser.

```
psPage.SetTrace (true);
```

See also

BeforeAction
BeforeBinding

AfterGenerate

Description

Occurs after all generation has taken place.

Applies to

psPage object

Arguments

None

Return codes

None

Usage

This event does not occur if a Redirect method is called. If this event occurs, it is followed by the RequestFinish event.

Examples

This example turns off tracing for all events after page generation.

```
psPage.SetTrace (false);
```

See also

RequestFinish

BeforeAction

Description

Occurs after data binding and validation and just before any action is performed.

Applies to

psPage object

Arguments

None

Return codes

Boolean. Returning false stops any further processing; returning true allows processing to continue. You must include a return value in the event script.

Usage

This event enables you to do any required preprocessing before the action is initiated.

Error processing
Because this event is triggered before generation occurs, you cannot use psDocument.Write to report errors. Instead, use the ReportError method on the psPage object to trigger the ServerError event. The error will then be added to the error log, depending on the ServerError return value.

Examples

This script displays a client-side alert box message if the Redirect method is not called for another psPage event:

```
psPage.Alert (MyVar + " in BeforeAction event", true);
return true;
```

See also

AfterBinding
BeforeAction for PSWebDataWindowClass objects

BeforeBinding

Description

Occurs only when doing a self-navigation. It occurs after the server-side objects have been created and the page variables have been filled, but before the controls have been bound to the input data.

Applies to

psPage object

Arguments

None

Return codes

None

Usage

This event allows advanced users to manipulate the object model in advance of data binding. Changing the values of the server-side objects can trigger the ItemChanged event for controls on the page.

Error processing
Because this event is triggered before generation occurs, you cannot use psDocument.Write to report errors. Instead, use the ReportError method on the psPage object to trigger the ServerError event. The error will then be added to the error log, depending on the ServerError return value.

Examples

This script displays a client-side alert box message if the Redirect method is not called for another psPage event:

```
psPage.Alert (MyVar + " in BeforeBinding event", true);
```

See also

AfterBinding
ItemChanged

BeforeGenerate

Description

Occurs before any generation happens. It is triggered both when the page is requested for the first time and when a self-navigation is done. The psPage variable *firstTime* stores whether or not the page is requested for the first time.

Applies to

psPage object

Arguments

None

Return codes

Boolean. If false is returned, generation does not occur. If true is returned, generation occurs normally. You must include a return value in the event script.

Usage

This event is not fired if a Redirect method was called during any previous event. This event is the last chance to modify the data on the server side before generation begins.

Error processing
Because this event is triggered before generation occurs, you cannot use psDocument.Write to report errors. Instead, use the ReportError method on the psPage object to trigger the ServerError event. The error will then be added to the error log, depending on the ServerError return value.

Examples

This script displays a client-side alert box message if the Redirect method is not called for another psPage event:

```
psPage.Alert (MyVar + " in BeforeGenerate event", true);
return true;
```

See also

AfterGenerate

FirstTime

Description

Occurs the first time the page is accessed. Server-side objects are created and page variables filled before this event is triggered.

Applies to

psPage object

Arguments

None

Return codes

Boolean. You must include a return value in the event script.

Usage

This event is the place to include initialization that you want to have occur only the first time the page is accessed. For example, you could use this event to call *webdw*.Retrieve to fetch data, or *webdw*.InsertRow to start off in data entry mode. It is the equivalent of the PowerBuilder Open event.

If binding is not selected for a Web DataWindow control, you should call either Retrieve or Insert in the FirstTime event. If the control is using a stateless server component and Retrieve is called once, the server automatically re-performs the retrieve, using the retrieval arguments that were passed to Retrieve during the binding phase.

Error processing

Because this event is triggered before generation occurs, psDocument.Write cannot be used for reporting errors. Instead, you can use the ReportError method on the psPage object to trigger the ServerError event. The error will then be added to the error log, depending on the ServerError return value.

Examples

This example adds an item to the user's shopping cart if the value of the action page parameter (passed from a linking page) is "add". It then retrieves information from the user's shopping cart in a DataWindow:

```
if (action == "add") {
  n_cart.additem(user, cd_id);
}
dw_cart.Retrieve(user);
```

The following example shows code placed in the FirstTime event of a page that is loaded from a logon screen when the user enters a password that is incorrect. The showErrorsOnPage property is set to false because the error will be displayed at a precise location on the page by calling WriteErrorsToDocument in a server-side script. The error message needs to be displayed only once—in this case, at the location of the server-side script:

```
psPage.ReportError(myLocation, myCause, "Incorrect
       password");
psPage.showErrorsOnPage = false;
```

The following example returns the value of the page variable *myVar* in an alert box the first time the page is accessed:

```
psPage.Alert (myVar + " in FirstTime event", true);
return true;
```

See also

InsertRow in the *DataWindow Reference*
Retrieve in the *DataWindow Reference*

ItemChanged

Description

Occurs when the value of a control has changed and passed validation.

Applies to

PSCheckBoxClass, PSDropDownListClass, PSPasswordClass, PSRadioGroupClass, PSTextAreaClass, and PSTextClass objects

Arguments

None

Return codes

None

Usage

Before this event is called, the new value must be selected (check box, drop-down list, radio button group), typed (text fields) in the client browser, or otherwise placed (through scripts) in the Value property of the control.

Radio buttons are different from other controls because they function as a group. Each button in the group uses the same binding and has the same properties. The Integrated Script editor lets you script the ItemChanged event for a single radio button, but the script you enter applies to the group. If you select the ItemChanged event for other radio buttons in the same group, the script is displayed there as well. (On the Source page of the HTML editor, the script appears in the INPUT tag for only one of the radio buttons.)

Examples

This script displays a client-side alert box message if the Redirect method is not called for a psPage event:

```
psPage.Alert ("Value changed in check box", true);
```

RequestFinish

Description	Last event to occur on the page. It happens after all generation is complete.
Applies to	psPage object
Arguments	None
Return codes	None
Usage	This event allows for any last-minute cleanup to be done. It is also the place where the persistence of any failover data can be done. This event is always triggered, even if Redirect was called.
See also	AfterGenerate RequestStart

RequestStart

Description	Occurs at the beginning of page processing, before server-side objects have been created and before any data binding or variable retrieval.
Applies to	psPage object
Arguments	None
Return codes	Boolean. If false is returned, no other processing occurs. If true is returned, processing continues normally. You must include a return value in the event script.
Usage	This event allows advanced users to short-circuit the normal processing. Input parameters are made available by calling the psDocument.GetParam method. Page variables are not valid during this event.

Error processing
Because this event is triggered before generation occurs, psDocument.Write cannot be used for reporting errors. Instead, you can use the ReportError method on the psPage object to trigger the ServerError event. The error is then added to the error log, depending on the ServerError return value.

See also	FirstTime RequestFinish

ServerAction

Description — Occurs when a user action was the trigger for a page refresh, and after all validation and data binding has taken place.

Applies to — PSButtonClass, PSImageClass, PSStaticTextClass

Arguments — None

Return codes — None

Usage — This event gives you the opportunity to respond to an action that the client performed. One action might be to link to a new page through a client-side redirect.

If the ServerAction event is not scripted for a Static Text control, the control is generated as an HTML SPAN tag. If the ServerAction event is scripted, an onClick event is scripted that causes a form submit.

Examples — This statement in the ServerAction event changes the Web page that displays in the client-side browser:

```
psPage.Redirect("My_WebPage.htm");
```

See also — Redirect

ServerError

Description — Occurs when the ReportError method is called. It can be used to alert you when something goes wrong during processing.

Applies to — psPage object

Arguments

Argument	Description
location	String for *objectName.methodName* passed to ReportError
cause	String for the error cause
message	String for the system error message

Return codes — Boolean. If true is returned, the error is added to the error list. If false is returned, the error is not added to the list. The error list can be used to provide application-specific error processing.

Usage — The arguments are those passed to ReportError. Use this event instead of the psDocument.Write method to report errors that occur before generation.

The following table gives the string values that can be passed in the *cause* and *message* arguments:

Cause	Object (method)	Meaning	Message string
Component call failed	Various (TestCompError)	An EAServer method call failed	Returned message
Could not create component	PSJaguarObjectClass (CreateComponent)	java.CreateComponent failed for declarative EAServer object	Returned message
Restore context failed	PSWebDataWindow Class(BindToInput)	Restoring the context through SetAction call failed	Returned code
Create failed	PSWebDataWindow Class(loadDWObject)	The source for a DataWindow definition failed to compile	Compile error
Source not found	PSWebDataWindow Class(loadDWObject)	The source for a DataWindow definition could not be found	Specified URL
SetDWObject failed	PSWebDataWindow Class(loadDWObject)	The call to SetDWObject failed	Error return code, specified PBL, and DW name
Configuring database parms failed	PSWebDataWindow Class(connectTo Component)	The operation defined in the location parameter failed	Error return value
Operation failed	PSWebDataWindow Class(connectTo Component)	The operation defined in the location parameter failed	Error return value
DB error	PSWebDataWindow Class (triggerOnDBError)	A database error occurred	SQLDB code and SQLERRTEXT

DB error is passed for the *cause* argument only if you do not set the return on the OnDBError event to 1.

See also

ReportError

Validate

The Validate event has different arguments for different objects:

Object	See
psPage	Syntax 1
PSPasswordClass, PSTextAreaClass, PSTextClass	Syntax 2

For use with PSWebDataWindowClass objects, see Validate in the chapter on server-side Web DataWindow events.

Syntax 1 For psPage objects

Description: Occurs after all data binding has taken place and all validation has been performed. It allows you to do page-level validation in 4GL JSP targets.

Arguments: None

Return codes: Boolean. If true is returned, the page is considered valid. If false is returned, the psPage.ValidationError event is triggered. The action that fails validation is not performed. You must include a return value in the event script.

Usage: Use to set a condition for the page or a control on the page.

Examples: This example tests for whether a user-entered password is valid (the same value as the user name). If it is not, the ValidationError event is triggered. If it is valid, the AfterBinding event is triggered and (usually) a Redirect method is called:

```
return userid.value == password.value;
```

See also: ValidationError

Syntax 2 For server-side text controls

Description: Occurs when the client changes the value of a text control in 4GL JSP targets. The value entered by the user is passed to the event.

Applies to: PSPasswordClass, PSTextAreaClass, PSTextClass

Arguments:

Argument	Description
sNewValue	String variable for the new value the user enters

Return codes

Boolean. If true is returned, the validation is considered successful. If false is returned, the *control*.ValidationError event is triggered. The action that fails validation is not performed. You get a compiler error if you do not include a return value in the event script.

Usage

If this event is not scripted, validation is assumed to have succeeded.

This event gives you a chance to do complex validation on the value (for example, by calling an EAServer component to do the validation). Simple validation should preferably be written in client-side JavaScript.

Examples

This example calls the validate method on an EAServer component called "n_creditcard". The method (and event) returns true if the credit card is valid.

```
return
       n_creditcard.validate(amount,
          cc_type,cc_number,cc_expiration);
```

See also

ValidationError

ValidationError

The ValidationError event has different arguments for different objects:

Object	See
psPage	Syntax 1
PSPasswordClass, PSTextAreaClass, PSTextClass	Syntax 2

For use with PSWebDataWindowClass objects, see ValidationError in the chapter on server-side Web DataWindow events.

Syntax 1 For psPage objects

Description

Occurs if the psPage.Validate event returns false or if any control's Validate event returns false.

Arguments

None

Return codes

None

Usage

Use this event to report any validation errors.

One way of reporting validation errors is to place an error message in a Static Text control that you make visible when an error is detected. Another way is to call the Alert method to open a client-side alert box after the page is displayed. A third way is to call ReportError.

The first two ways display validation errors to the client. If you want to plug validation errors into the standard error processing mechanism, use ReportError.

Error processing
Because this event is triggered before generation occurs, psDocument.Write cannot be used for reporting errors. Instead, you can use the ReportError method on the psPage object to trigger the ServerError event. The error is then added to the error log, depending on the ServerError return value.

Examples

This code in the ValidationError event for a logon window makes visible a previously hidden Static Text control (containing text indicating an invalid password). It also sets the *logonValid* variable to false:

```
st_invalid.visible = true;
logonValid = false;
```

See also

Alert

Syntax 2

For server-side text controls

Description

Occurs if the *control*.Validate event fails.

Applies to

PSPasswordClass, PSTextAreaClass, PSTextAreaClass

Arguments

Argument	Description
sNewValue	String variable for the user-entered value that fails the *control*.Validate test

Return codes

None

Usage

Use this event to report validation errors on user-entered data.

See also

Validate

CHAPTER 3

Web DataWindow Events

About this chapter

This chapter describes server-side events for PSWebDataWindowClass objects that represent the 4GL Web DataWindow on the page server.

Contents

Using server-side events with the Web DataWindow DTC

Server-side events are selectable from the second drop-down list in the Web Target integrated Script editor. They are displayable only when a 4GL Web DataWindow object to which they apply is selected in the first drop-down list. These events display in blue to distinguish them from client-side events, which are listed in black.

For information on scripting events, see *Working with JSP Targets*.

For information on client-side events and on server-side methods for Web DataWindow objects, see the *DataWindow Reference*.

AfterAction

Description: Occurs just after the call to SetAction on the server component.

Applies to: PSWebDataWindowClass objects

Arguments: None

Return codes: None

Usage: This event is not triggered if the call to SetAction fails.

See also: AfterAction for psPage object
SetAction in the *DataWindow Reference*

AfterRetrieve

Description: Occurs just after the call to Retrieve on the server component.

Applies to: PSWebDataWindowClass objects

Arguments:

Argument	Description
numRows	Number of rows retrieved by the server component

Return codes: None

Usage: This event is not triggered if the call to Retrieve fails.

This is the equivalent of the RetrieveEnd event for the PowerBuilder DataWindow.

Examples: This example counts the rows retrieved and sends a message to the user if the count is higher than 1000:

```
If (numRows>1000) {
 psPage.Alert ("Please refine your database query.\n"
    +"It currently returns in excess of " + numRows
    +"  rows.", true);
 psPage.Redirect("sendingpage.html");}
```

See also: Retrieve in the *DataWindow Reference*

AfterUpdate

Description	Occurs just after the call to Update on the server component or just after an action of Update is performed.
Applies to	PSWebDataWindowClass objects
Arguments	None
Return codes	None
Usage	This event is not triggered if the call to Update fails. This is the equivalent of the UpdateEnd event for the PowerBuilder DataWindow.
See also	Update in the *DataWindow Reference*

BeforeAction

Description	Occurs just before SetAction is called on the server component.
Applies to	PSWebDataWindowClass objects
Arguments	None
Return codes	Boolean. If true is returned, SetAction is called on the server component during the ServerAction phase of processing. If false is returned, SetAction is not called. You must include a return value in the event script.
Usage	Context restoration and action execution are separated in the server component. The context of the Web DataWindow is restored by the Validate event before this event is triggered.
See also	BeforeAction for psPage object SetAction in the *DataWindow Reference*

BeforeRetrieve

Description	Occurs just before the call to Retrieve on the server component.
Applies to	PSWebDataWindowClass objects
Arguments	None
Return codes	Boolean
Usage	If true is returned, Retrieve is called on the server component. If false is returned, Retrieve is not called. You must include a return value in the event script. This is the equivalent of the RetrieveStart event for the PowerBuilder DataWindow.

Examples	This example sorts a DataWindow called "dw_1" by its fourth column in ascending order before the data is displayed in the client browser:
	
	`dw_1.SetSort ("#4, A");`
	`dw_1.Sort ();`
	`return true;`

See also	Retrieve in the *DataWindow Reference*

BeforeUpdate

Description	Occurs just before the call to Update on the server component or just before an action of Update is performed.
Applies to	PSWebDataWindowClass objects
Arguments	None
Return codes	Boolean
Usage	If true is returned, Update is called on the server component. If false is returned, Update is not called. You must include a return value in the event script. This is the equivalent of the UpdateStart event for the PowerBuilder DataWindow.
See also	Update in the *DataWindow Reference*

OnDBError

Description

Occurs if a database error takes place during processing.

Applies to

PSWebDataWindowClass objects

Arguments

Argument	Description
sqlDBCode	Number corresponding to a database-specific error code. (See your DBMS documentation for the meaning of the code.)
sqlErrText	String with a database-specific error message.
sqlSyntax	String with the full text of the SQL statement being sent to the DBMS when the error occurred.
buffer	String for the buffer containing the row involved in the database activity that caused the error.
row	Number of the row involved in the database activity that caused the error (the row being updated, selected, inserted, or deleted).

Return codes

Boolean. You can set the return code to affect the type of error message displayed. By default, when the DBError event occurs in a DataWindow control, it displays a system error message. You can display your own message and suppress the system message by specifying a return code of true in the DBError event. You must include a return value in the event script.

Usage

This event is the equivalent of the DBError event for the PowerBuilder DataWindow.

Examples

This example redirects users to a more user-friendly error page describing the database error. It passes error parameters to the new page:

```
PSArgClass args = new PSArgClass ( );
args.addArg ("arg1", sqlDBCode);
args.addArg ("arg2", buffer);
args.addArg ("arg3", row);
psPage.Redirect("DBErrorPage.html", args);
return true;
```

See also

DBError in the *DataWindow Reference*
ServerError

Validate

Description	Occurs immediately after the context is restored in the server component.
Applies to	PSWebDataWindowClass objects
Arguments	None
Return codes	Boolean. If true is returned, the validation is considered successful. If false is returned, the *webdw*.ValidationError event is triggered. You get a compiler error if you do not include a return value in the event script.
Usage	The server component action is performed only after the validation succeeds.
Examples	This example makes sure that the salary entered is greater than $20,000.

```
var result;
/* real(gettext() )  > 20000 */
result = (parseFloat(exprCtx.currentText) > 20000);
return result;
```

See also	Validate for other Web Target object model objects ValidationError

ValidationError

Description	Occurs if the *webdw*.Validate event fails.
Applies to	PSWebDataWindowClass objects
Arguments	None
Return codes	None
Usage	You can call Modify on a particular text object in the DataWindow to transmit the validation error message. If you place the Web DataWindow on a 4GL Web page and you want to plug validation errors into the standard error processing mechanism, use psPage.ReportError. To report validation errors to the client, you can place an error message in a Static Text control that you make visible when the error is detected. Another way is to call the psPage.Alert method. These error reporting methods are available only for 4GL Web pages.
Examples	This example gives the validation error if the salary is not greater than $20,000.

```
var result;
/* 'Salary must be greater then $20,000' */
result = "Salary must be greater then $20,000";
return result;
```

See also	Modify in the *DataWindow Reference* Validate ValidationError for other Web Target object model objects

CHAPTER 4

JSP Target Methods

About this chapter

This chapter describes the methods you can call on objects in the Web Target server-side object model.

Contents

Abandon

Description — Causes a session object to be discarded.

Applies to — PSSessionClass object

Syntax — psSession.Abandon( )

Return value — None

Usage — At runtime, Abandon destroys the current session object immediately.

Examples — The following example destroys the current session object:

```
psSession.Abandon();
```

addArg

Description — Use to include name and value pairs as parameters that you redirect to another Web page. This method is for use with 4GL targets only.

Applies to — PSArgClass

Syntax —

ArgObj.addArg (String *argName*, String *Value*)
ArgObj.addArg (String *argName*, boolean *Value*)
ArgObj.addArg (String *argName*, byte *Value*)
ArgObj.addArg (String *argName*, char *Value*)
ArgObj.addArg (String *argName*, double *Value*)
ArgObj.addArg (String *argName*, int *Value*)
ArgObj.addArg (String *argName*, float *Value*)
ArgObj.addArg (String *argName*, long *Value*)
ArgObj.addArg (String *argName*, short *Value*)
ArgObj.addArg (String *argName*, Object *Value*)

Argument	Description
ArgObj	Object of the PSArgClass type to which you add name-value pairs
argName	Name of the argument you want to add
Value	Value of the argument you want to add

Return value — None

Usage — You can use this overloaded method to add parameters of the supported datatype to the PSArgClass object that you include in a psPage.Redirect call.

Examples

This example uses a PSArgClass object with a single parameter. That parameter (param1) is defined on the Parameters tab of the Page Properties dialog box. The value defined for the parameter is added to the PSArgClass object that is included in the psPage.Redirect call:

```
PSArgClass myParam=null;
myParam = new PSArgClass();
myParam.addArg("param1", param1);
psPage.Redirect("page_2.jsp", myParam);
```

See also

Redirect

Alert

Description

Causes a client-side alert box to be displayed when the page is finished loading. This method is for use with 4GL targets only.

Applies to

psPage object

Syntax

psPage.Alert (string *message* {, boolean *appendToCurrent* })

Argument	Description
message	Message to be displayed in the client-side alert box
appendToCurrent (optional)	Indicates whether the passed message should replace any current message (the default) or be appended to the current message

Return value

None

Usage

By using the optional argument *appendToCurrent* and setting its value to true, you can present all validation problems to the user at once.

Calling the Alert method is one way of reporting validation errors. Another way is to place an error message in a Static Text control that you make visible when an error is detected. A third way is to call ReportError.

The first two ways display validation errors to the client. If you want to plug validation errors into the standard error processing mechanism, use ReportError.

Do not use HTML in error messages
Do not use HTML for messages you want to display in a client-side alert box, because any HTML tags that you type for the message text will also be visible in the alert box.

Examples

This call displays the value of the variable *MyVar* with the added text in quotes. If another Alert call has been made previously, the optional argument makes sure that both alert messages are displayed:

```
psPage.Alert (MyVar + " in FirstTime event", true);
return true;
```

ClearError

Description

Clears the list of error objects.

Applies to

PSConnectionClass objects

Syntax

connectionobject.ClearError( )

Argument	Description
connectionobject	A variable that contains a reference to an instance of PSConnectionClass

Return value

None

Usage

At runtime, ClearError dstroys the current error object by setting it to null.

Examples

The following code clears the list of errors for a PSConnectionClass connection object called myconnect:

```
myconnect.ClearError();
```

See also

CreateConnection

CreateCommand

Description — Creates a named object that represents a SQL statement.

Applies to — PSConnectionClass objects

Syntax — *connectionobject*.CreateCommand(*sql*)

Argument	Description
connectionobject	A variable that contains a reference to an instance of PSConnectionClass
sql	A string that specifies the SQL statement

Return value — PSCommandClass object

Usage — CreateCommand allows you to execute the same SQL statement multiple times on a single page. Once you have created the command object, you can execute the command by calling the Execute method.

At runtime, CreateCommand returns a command object that can be used to execute the SQL statement that you pass as an argument.

Examples — The following code creates a SQL command object named mycommand and executes the SQL statement associated with the command:

```
PSCommandClass mycommand;
mycommand = myconnect.CreateCommand("SELECT * FROM
      products");
mycommand.Execute();
```

CreateConnection

Description — Creates a new database connection.

Applies to — PSServerClass object

Syntax — **Syntax with user name and password**

psServer.CreateConnection (pageContext, *Driver*, *URL*, *user*, *password*, {*bTrace*})

Syntax with database properties

psServer.CreateConnection (pageContext, *Driver*, *URL*, *Properties*, {*bTrace*})

Argument	Description
connectionstring	A string that specifies connection parameters (for example, the name of the data source to which you want to connect).
user	The user name for the connection.
password	The password for the specified user name.
pageContext	The implicit pageContext object available to JSP targets.
Driver	The connection mechanism used to connect to the database.
URL	The location of the database to which you want to connect. The database URL is obtained from the database JDBC driver documentation.
Properties	Any properties that your JDBC driver uses to connect to the database. If properties are defined, you must also define the user ID and password in the properties that you list.
bTrace (Optional)	Allows tracing if set to true. The default is false.

Return value

PSConnectionClass object

Usage

CreateConnection defines a set of reusable parameters for connecting to a database.

At runtime, CreateConnection creates a JDBC connection object.

Examples

The following example creates a new connection with the user name “dba” and the password “sql”, and turns on tracing:

```
PSConnectionClass myConnect;
myConnect = psServer.CreateConnection(pageContext,
    "com.sybase.jdbc2.jdbc.SybDriver",
    "jdbc:sybase:Tds:localhost:2638", "dba", "sql",
    true);
```

CreateCursor

Description

Creates a database cursor.

Applies to

PSConnectionClass objects

Syntax

Syntax with a single argument

connectionobject.CreateCursor(*sql*)

Syntax with multiple arguments

connectionobject.CreateCursor(*sql* , *resultSetType*, *resultSetConcurrency*)

Argument	Description
connectionobject	A variable that contains a reference to an instance of PSConnectionClass.
sql	A string that specifies the SQL statement.
resultSetType	An int that specifies the result set type. If the single argument syntax is used in a JSP target, TYPE_SCROLL_INSENSITIVE is passed as a default argument.
resultSetConcurrency	An int that specifies the result set concurrency properties. If the single argument syntax is used in a JSP target, CONCUR_UPDATABLE is passed as a default argument.

Return value

PSCursorClass object

Usage

At runtime, CreateCursor creates a cursor object and executes the SQL statement.

Examples

The following example creates a cursor to retrieve rows from the products table. Once the data has been retrieved, the code in the example writes out each row in the cursor. If an error occurs, it writes out the error code and message text:

```
String myquery;
PSConnectionClass myconnect;
PSCursorClass mycursor;
PSErrorClass errobj;
boolean dberror;

myConnect = psServer.CreateConnection(pageContext,
    "com.sybase.jdbc2.jdbc.SybDriver",
    "jdbc:sybase:Tds:localhost:2638", "dba", "sql",
    true);
...
myquery = "SELECT products.prod_id, products.prod_name
       FROM DBA.products";
```

```
mycursor = myconnect.CreateCursor(myquery);
if ( myconnect.GetError() == null )
       {
       for (i=0; (!mycursor.EOF()); i++)
           {
           psDocument.Write(mycursor.GetValue(0) + " " +
              mycursor.GetValue(1));
           psDocument.Write("<BR>");
           mycursor.MoveNext();
           }
       }
   else {
       dberror = true;
       }
if ( dberror == true )
           {
           errobj = myconnect.GetError();
           str = errobj.GetCode() + " " +
              errobj.GetMessage();
           psDocument.Write ( str );
           }
```

EOF

Description — Determines whether the end of a cursor has been reached.

Applies to — PSCursorClass objects

Syntax — *cursorobject*.EOF( )

Argument	Description
cursorobject	A variable that contains a reference to an instance of PSCursorClass

Return value — Boolean. Returns true if the end of the cursor has been reached, and false if it has not.

Usage — At runtime, EOF calls the isAfterLast method on the ResultSet object to determine if the end of the cursor has been reached.

Examples

The following example uses EOF in a loop to determine whether all of the rows in a cursor have been processed:

```
String myquery;
PSConnectionClass myconnect;
PSCursorClass mycursor;
boolean dberror;

myConnect = psServer.CreateConnection(pageContext,
    "com.sybase.jdbc2.jdbc.SybDriver",
    "jdbc:sybase:Tds:localhost:2638", "dba", "sql",
    true);
...
myquery = "SELECT products.prod_id, products.prod_name
       FROM DBA.products";
mycursor = myconnect.CreateCursor(myquery);

if ( myconnect.GetError() == null )
       {
       for (i=0; (!mycursor.EOF()); i++)
           {
           psDocument.Write(mycursor.GetValue(0) + " " +
               mycursor.GetValue(1));
           psDocument.Write("<BR>");
           mycursor.MoveNext();
           }
else dberror = true;
```

Execute

Description

Executes a SQL command.

Applies to

PSCommandClass objects

Syntax

commandobject.Execute( )

Argument	Description
commandobject	A variable that contains a reference to an instance of PSCommandClass

Return value

PSCursorClass object

Usage

At runtime, Execute creates a new ResultSet object using the SQL statement stored with the PSCommandClass object.

Examples

The following example performs a database query by executing the SQL statement associated with a command object:

```
PSConnectionClass myconnect;
PSCommandClass mycommand;
myConnect = psServer.CreateConnection(pageContext,
    "com.sybase.jdbc2.jdbc.SybDriver",
    "jdbc:sybase:Tds:localhost:2638", "dba", "sql",
    true);
mycommand = myconnect.CreateCommand("SELECT * FROM
       products");
mycommand.Execute();
```

File

Description

Returns the file name and extension for the current document.

Applies to

PSDocumentClass object

Syntax

psDocument.File( )

Return value

String

Usage

At runtime, File extracts the file name and extension from the PATH_INFO server environment variable.

Examples

The following code returns the file name of the current document into a variable called *myfile*. If the URL for the current document were *http://www.sybase.com/index.htm*, the value of *myfile* would be *index.htm* after the call to File:

```
myfile = psDocument.File( );
```

FillRetrievalArgs

Description

Fills in the retrieval arguments array based on names of the page variables that are passed.

Applies to

PSDataWindowClass object

Syntax

DWObject.FillRetrievalArgs(*variable*[ ] arguments)

Argument	Description
DWObject	A variable that contains a reference to an instance of PSDataWindowClass
variable	An array for String variables to use as retrieval arguments

Return value

Retrieval arguments

Usage

At runtime FillRetrievalArgs adds the values of the named variables to the array in the order in which they are passed. If a value for a variable is not found, then an empty string " " is added to the array.

Examples

In the following code, the retrieval argument array is filled with "MyParam1" and "MyParam2" for the DataWindow object named "htmlDwObj1":

```
htmlDwObj1.FillRetrievalArgs("MyParam1", "MyParam2");
```

Generate

Description

Generates the inline HTML for the Web DataWindow. As this method generates the HTML, it also generates inline connection and database errors. Page variables that you want to maintain need to be passed in.

Applies to

PSDataWindowClass and PSWebDataWindowClass objects

Syntax

DWObject.Generate(*page*[ ] variables)

Argument	Description
DWObject	A variable that contains a reference to an instance of PSDataWindowClass
page	An array for passing page variables of the String datatype

Return value

Integer. 1 indicates success, and -1 indicates failure.

Usage

At runtime, Generate performs the tasks required to generate the dynamic HTML including retrieving the action context and generating the HTML inline. Connection errors, including database error messages, are also generated inline.

Examples

In the following code, the DataWindow object htmlDwObj1 generates HTML and maintains the page variables called "MyParam1" and "MyParam2":

```
htmlDwObj1.Generate("MyParam1", "MyParam2");
```

GenerateXHTML

Description

Generates the inline content for the XHTML Web DataWindow.

Applies to

PSDataWindowClass and PSWebDataWindowClass objects

Syntax

DWObject.GenerateXHTML({*page*[ ] variables})

Argument	Description
DWObject	A variable that contains a reference to an instance of PSDataWindowClass or PSWebDataWindowClass.
page	An array for passing page variables for objects of type PSDataWindowClass. The page variables must be defined as String datatypes.

Return value

Integer. 1 indicates success, and -1 indicates failure.

Usage

At runtime, GenerateXHTML performs the tasks required to generate the dynamic XHTML, including retrieving the action context and generating the XHTML inline. Connection errors, including database error messages, are also generated inline.

The following table shows when it is best to use the HTML, XHTML, or XML Web DataWindow:

HTML Web DataWindow use	XHTML Web DataWindow use	XML Web DataWindow use
When the HTMLGen.PageSize property is *not* assigned a value (row count per page changes)	When you want more industry-standard Web pages than HTML can provide and the ability to customize pages using an XHTML export template	When the HTMLGen.PageSize property is assigned a value (row count per page stays the same), and you want industry-standard Web pages and the ability to customize pages using an XHTML export template
Small amounts of data	Small amounts of data	Large amounts of paged data

Examples

In the following code, the DataWindow object htmlDwObj1 generates XHTML and maintains the page variables called “MyParam1” and “MyParam2”:

```
htmlDwObj1.GenerateXHTML("MyParam1", "MyParam2");
```

See also

Generate
GenerateXMLWeb

GenerateXMLWeb

Description

Generates XML content, XSLT layout, and CSS style sheets for a Web DataWindow, which is transformed to XHTML on the client side.

Applies to

PSDataWindowClass and PSWebDataWindowClass objects

Syntax

DWObject.GenerateXMLWeb({*page*[ ] variables})

Argument	Description
DWObject	A variable that contains a reference to an instance of PSDataWindowClass or PSWebDataWindowClass.
page	An array for passing page variables for objects of type PSDataWindowClass. The page variables must be defined as String datatypes.

Return value

Integer. 1 indicates success, and -1 indicates failure.

Usage

The GenerateXMLWeb function uses the resource base and publish paths for a DataWindow object to determine where it generates XML, XSLT, CSS, and JS files. If a resource base or a publish path is not specified for a DataWindow object, the GenerateXMLWeb function creates a TEMP directory on the server where the XML, XSLT, CSS, and JS files are stored.

At design time, you can override the resource base and publish paths by making Modify calls on the DataWindow object in the Source view before you call GenerateXMLWeb. The following example creates separate subdirectories for XML, XSLT, CSS, and JS files:

```
String resourceBase = request.getScheme() + "://" +
request.getServerName() + ":" + request.getServerPort()
+ request.getContextPath();

String publishPath = application.getRealPath("/");

dwGen.Modify("DataWindow.XMLGen.ResourceBase = '" +
resourceBase + "/xml'");

dwGen.Modify("DataWindow.XMLGen.PublishPath = '" +
publishPath + "xml'");

dwGen.Modify("DataWindow.XSLTGen.ResourceBase = '" +
resourceBase + "/xsl'");

dwGen.Modify("DataWindow.XSLTGen.PublishPath = '" +
publishPath + "xsl'");

dwGen.Modify("DataWindow.CSSGen.ResourceBase = '" +
resourceBase + "/css'");

dwGen.Modify("DataWindow.CSSGen.PublishPath = '" +
```

```
publishPath + "css'");

dwGen.Modify("DataWindow.JSGen.ResourceBase = '" +
resourceBase + "/js'");

dwGen.Modify("DataWindow.JSGen.PublishPath = '" +
publishPath + "js'");
```

At runtime, the client browser displays an XHTML page that it transforms from XML, XSLT, CSS, and JS files that it gets initially from the server. However, after the initial loading of the page, the client does not need to go back to the server to obtain structure (XSLT) and layout (CSS) information, as these remain in the browser's cache. This provides greater efficiency and scalability for your Web applications.

Examples

In the following example, the DataWindow object dwGen generates XML, XSLT, and CSS files for the content, structure, and style of the Web DataWindow:

```
dwGen.GenerateXMLWeb();
```

See also

Generate
GenerateXHTML

getCharacterEncoding

Description

Returns the charset encoding for the PSArgClass object. This method is for use with 4GL targets only.

Applies to

PSArgClass

Syntax

ArgObj.getCharacterEncoding ( )

Argument	Description
ArgObj	Object of the PSArgClass type for which you want to obtain the character set used to encode values

Return value

String. The returned string is the charset used by the PSArgClass object.

Usage

If you are adding arguments to a URL in a psPage.Redirect call, you might need to know the character set used by the PSArgClass argument.

You can set the character encoding for the PSArgClass object by calling the setCharacterEncoding method.

Examples

This example gets the value of the charset encoding of the PSArgClass and writes it to a text box:

```
PSArgClass myURLParam=null;
String charSet;
myURLParam = new PSArgClass();
charSet=myURLParam.getCharacterEncoding();
sle_1.value=charSet;
```

See also

addArg
setCharacterEncoding

GetCode

Description

Returns the code associated with the current error object.

Applies to

PSErrorClass objects

Syntax

errorobject.GetCode( )

Argument	Description
errorobject	A variable that contains a reference to an instance of PSErrorClass

Return value

String. Returns the error code.

Usage

At runtime, GetCode returns the error code stored in an error object.

Examples

The following example connects to a database and retrieves some data. In this example, “mycommand” is a variable of type PSCommandClass, “mycursor” is a variable of type PSCursorClass, “myquery” is a variable of type String, and “dberror” is a variable of type boolean. If an error occurs, a GetCode call retrieves the code for the error:

```
myconnect =
       psServer.GetConnection("SalesDBConnection");
if ( myconnect.GetError() == null )
       {
       myquery = "Select * from sales";
       mycursor = myconnect.CreateCursor ( myquery );
       if ( myconnect.GetError() == null )
          {
          // Process the retrieved data
          }
       else dberror = true;
```

```
        }
else dberror = true;
        if ( dberror == true )
            {
            errobj = myconnect.GetError();
            str = errobj.GetCode() + " " +
                errobj.GetMessage();
            psDocument.Write ( str );
            }
```

GetColumnCount

Description — Retrieves the number of columns in a cursor.

Applies to — PSCursorClass objects

Syntax — *cursorobject*.GetColumnCount( )

Argument	Description
cursorobject	A variable that contains a reference to an instance of PSCursorClass

Return value — Integer

Usage — At runtime, GetColumnCount calls the getColumnCount method on the ResultSetMetaData object.

Examples — The following code retrieves the number of columns in the “mycursor” object:

```
columncount = mycursor.GetColumnCount();
```

See also — CreateCursor

GetColumn<*DataType*>

Description

All of the GetColumn<*DataType*> methods have two syntaxes; one that takes a String argument for the name of the column in the cursor, and one that takes an int argument for the number of the column in the cursor.

Applies to

PSCursorClass objects

Syntax

cursorobject.GetColumnBoolean (String *strColName*)

cursorobject.GetColumnBoolean (int *iCol*)

cursorobject.GetColumnByte (String *strColName*)

cursorobject.GetColumnByte (int *iCol*)

cursorobject.GetColumnDouble (String *strColName*)

cursorobject.GetColumnDouble (int *iCol*)

cursorobject.GetColumnFloat (String *strColName*)

cursorobject.GetColumnFloat (int *iCol*)

cursorobject.GetColumnInt (String *strColName*)

cursorobject.GetColumnInt (int *iCol*)

cursorobject.GetColumnLong (String *strColName*)

cursorobject.GetColumnLong (int *iCol*)

cursorobject.GetColumnShort (String *strColName*)

cursorobject.GetColumnShort (int *iCol*)

cursorobject.GetColumnString (String *strColName*)

cursorobject.GetColumnString (int *iCol*)

Argument	Description
cursorobject	A variable that contains a reference to an instance of PSCursorClass
strColName	Name of the column for which you want to obtain the value
iCol	Number of the column for which you want to obtain the value

Return value

Corresponds to the datatype in the method name.

Usage

Use in conjunction with the GetColumnType method to obtain values in a cursor. You cannot use GetValue to obtain column values in a cursor.

See also

GetColumnType

GetColumnLength

Description — Retrieves the length of a column that you identify by column name or column number.

Applies to — PSCursorClass objects

Syntax — *cursorobject*.GetColumnLength (String *strColName*)

cursorobject.GetColumnLength (int *iCol*)

Argument	Description
cursorobject	A variable that contains a reference to an instance of PSCursorClass
strColName	Name of the column for which you want to obtain the length
iCol	Number of the column for which you want to obtain the length

Return value — Integer

Usage — Use this method to obtain the length of a column in the cursor. You need to construct an object of the PSCursorClass or return an object of the PSCursorClass from the Execute method on an object of type PSCommandClass.

Examples — This example returns the length of the second column in a cursor:

```
int li_col=0;
PSCursorClass myCursor = null;
...
myCursor = myCommand.Execute();
li_col = myCursor.GetColumnLength (2);
```

GetColumnName

Description — Returns the column name in a cursor.

Applies to — PSCursorClass objects

Syntax — *cursorobject*.GetColumnName(int *iCol*)

Argument	Description
cursorobject	A variable that contains a reference to an instance of PSCursorClass
iCol	Number of the column for which you want to obtain the name

Return value — String

Usage

Use in conjunction with GetColumn<*DataType*> methods to obtain values in a cursor. You cannot use GetValue to obtain column values in a cursor.

Examples

The following example retrieves the name of the fifth column in the object called "mycursor":

```
column_name = mycursor.GetColumnName(5);
```

See also

CreateCursor
GetColumn<DataType>
GetColumnType

GetColumnType

Description

Returns the designated column's SQL type.

Applies to

PSCursorClass objects

Syntax

cursorobject.GetColumnType (int *iCol*)

Argument	Description
cursorobject	A variable that contains a reference to an instance of PSCursorClass
iCol	Number of the column for which you want to obtain the SQL type

Return value

Integer

Usage

Returns a static member variable in java.sql.Types. For a list of SQL types, see the Sun Microsystems Web site at http://java.sun.com/j2se/1.3/docs/api/java/sql/Types.html.

Use this method in conjunction with GetColumn<*DataType*> methods to obtain values in a cursor. You cannot use GetValue to obtain column values in a cursor.

Examples

The following example retrieves the type of the fifth column in the mycursor object:

```
column_type = mycursor.GetColumnType(5);
```

See also

CreateCursor
GetColumn<DataType>
GetColumnName
GetColumnTypeName

GetColumnTypeName

Description — Retrieves the designated column's database-specific type name.

Applies to — PSCursorClass objects

Syntax — *cursorobject*.GetColumnTypeName (int *iCol*)

Argument	Description
cursorobject	A variable that contains a reference to an instance of PSCursorClass
iCol	Number of the column for which you want to obtain the (database-specific) datatype name

Return value — String

Usage — Use this method to obtain the type name used by the database (such as int, datetime, varchar, and so on). If the column type is a user-defined type, then a fully-qualified type name is returned.

Examples — The following example retrieves the database-specific type name of the fifth column in the object called “mycursor”:

```
column_typename = mycursor.GetColumnTypeName(5);
```

See also — CreateCursor
GetColumn<DataType>
GetColumnName
GetColumnType

GetConnection

Description — Gets a reference to a database connection defined in the DatabaseConnections dialog box.

Obsolete method
This method is obsolete. It was previously used with ASP and PowerDynamo Web targets.

Applies to — PSServerClass object

Syntax — psServer.GetConnection(name)

Return value — PSConnectionClass object

GetEnv

Description

Retrieves the value of a server environment variable.

Applies to

PSDocumentClass object

Syntax

psDocument.GetEnv(*envvar*)

Argument	Description
envvar	The name of a server environment variable

Return value

String. Returns the value of the specified server environment variable. Returns null if the server environment variable does not exist.

Usage

The server environment variables that can be specified in GetEnv vary depending on which application server you deploy to. Here are the most common server environment variables:

Server environment variable	Description
AUTH_TYPE	The authentication method that the server uses to validate users who try to access protected scripts.
CONTENT_LENGTH	The length of the data message sent to the server. Applies only to queries that have information attached (such as those that use the POST and PUT methods).
CONTENT_TYPE	The datatype for the message sent to the server. Applies only to queries that have information attached (such as those that use the POST and PUT methods).
GATEWAY_INTERFACE	The version of the CGI interface specification being used by the server.
HTTP_headername	The contents of the specified request header. The server interprets any underscores (_) in headername as dashes in the actual header. For example, if you specify HTTP_MY_HEADER, the server searches for a header sent as MY-HEADER.
PATH_INFO	Extra path information found in the URL. PATH_INFO is the part of the URL found after the script name but before the query string.
PATH_TRANSLATED	The value of PATH_INFO along with the virtual path converted to a physical directory name.
QUERY_STRING	The query information stored in the string following the question mark (?) in the HTTP request.
REMOTE_ADDR	The IP address of the client making the request.
REMOTE_HOST	The domain name of the client making the request.

Server environment variable	Description
REQUEST_METHOD	The method used to make the request. For HTTP, this is GET, HEAD, POST, PUT, and so on.
SCRIPT_NAME	A virtual path to the script being executed. SCRIPT_NAME is used to construct URLs that refer back to the same gateway program script.
SERVER_NAME	The domain name of the server.
SERVER_PORT	The TCP/IP port that received the request.
SERVER_PROTOCOL	The name and version of the protocol (for example, HTTP/1.0).
SERVER_SOFTWARE	The name and version of the Web server software (for example, NCSA/1.3).

For a complete list of the server environment variables supported on your server platform, see your server documentation.

At runtime, GetEnv calls the mapped method in the ENVPARAM class.

Examples

The following example retrieves the IP address of the remote host making the current request:

```
remaddr = psDocument.GetEnv("REMOTE_ADDR");
```

GetError

Description

Returns the first error object.

Applies to

PSConnectionClass objects

Syntax

connectionobject.GetError( )

Argument	Description
connectionobject	A variable that contains a reference to an instance of PSConnectionClass

Return value

PSErrorClass object. If multiple errors were generated by the last database operation, GetError returns the first error object. If there were no errors, it returns null.

Usage

At runtime, GetError returns an error object that stores database connection errors.

Examples

In this example, “myconnect” is a variable of type PSConnectionClass, “mycommand” is a variable of type PSCommandClass, “mycursor” is a variable of type PSCursorClass, “myquery” is a variable of type String, “dberror” is a variable of type boolean, and “errobj” is a variable of type PSErrorClass. After each database operation, a GetError call checks for errors. If an error occurs, the error code and message are displayed:

```
if ( myconnect.GetError() == null )
       {
       myquery = "Select * from sales";
       mycursor = myconnect.CreateCursor( myquery );
       if ( myconnect.GetError() == null )
           {
           // Process the retrieved data
           }
       else dberror = true;
       }
else dberror = true;
    if ( dberror == true )
           {
           errobj = myconnect.GetError();
           str = errobj.GetCode() + " " +
              errobj.GetMessage();
           psDocument.Write ( str );
           }
```

GetMessage

Description

Returns the message associated with the current error object.

Applies to

PSErrorClass objects

Syntax

errorobject.GetMessage( )

Argument	Description
errorobject	A variable that contains a reference to an instance of PSErrorClass

Return value

String

Usage

At runtime, GetMessage returns the most recent error message stored in the named error object.

Examples

The following example connects to a database and retrieves some data. In this example, “mycursor” is a variable of type PSCursorClass, “myquery” is a variable of type String, “dberror” is a variable of type boolean, and “errobj” is a variable of type PSErrorClass. If an error occurs, GetMessage is called to get the error message:

```
PSConnectionClass myconnect = new
      PSConnectionClass(pageContext,
      "com.sybase.jdbc2.jdbc.SybDriver",
      "jdbc:sybase:Tds:localhost:2638",
       "dba","sql", true);

if ( myconnect.GetError() == null )
       {
       myquery = "Select * from sales";
       mycursor = myconnect.CreateCursor( myquery );
       if ( myconnect.GetError() == null )
           {
           // Process the retrieved data
           }
       else dberror = true;
       }
else dberror = true;
    if ( dberror == true )
           {
           errobj = myconnect.GetError();
           str = errobj.GetCode() + " " +
               errobj.GetMessage();
           psDocument.Write ( str );
           }
```

GetNextError

Description

Returns the next error object, if one exists.

Applies to

PSErrorClass objects

Syntax

errorobject.GetNextError( )

Argument	Description
errorobject	A variable that contains a reference to an instance of PSErrorClass

Return value

PSErrorClass object. If multiple error objects exist, returns an instance of PSErrorClass for the next error object. If no more error objects exist, it returns null.

Usage

At runtime, GetNextError returns the next error stored in the named error object.

Examples

The following example connects to a database and retrieves some data. In this example, “mycursor” is a variable of type PSCursorClass, “myquery” is a variable of type String, “dberror” is a variable of type boolean, and “errobj” is a variable of type PSErrorClass. If an error occurs, a GetNextError call accesses the list of error objects. For each error object, it writes out the error code and message text:

```
PSConnectionClass myconnect = new
      PSConnectionClass(pageContext,
      "com.sybase.jdbc2.jdbc.SybDriver",
      "jdbc:sybase:Tds:localhost:2638",
       "dba","sql", true);
if ( myconnect.GetError() == null )
       {
       myquery = "Select * from sales";
       mycursor = myconnect.CreateCursor ( myquery );
       if ( myconnect.GetError() == null )
           {
           // Process the retrieved data
           }
       else dberror = true;
       }
else dberror = true;
if ( dberror == true )
       errobj = myconnect.GetError();
       {
       while ( errobj != null )
           {
```

```
        str = errobj.GetCode() + " " +
            errobj.GetMessage();
        psDocument.WriteLn ( str );
        errobj = errobj.GetNextError();
        }
    }
```

GetParam

Description: Retrieves a parameter passed to the current page.

Applies to: PSDocumentClass object

Syntax: psDocument.GetParam(*argvar*)

Argument	Description
argvar	The name of a parameter passed to the current page

Return value: String. Returns the value of the specified parameter.

Usage: At runtime, GetParam calls the getParameter method on the request object.

Examples: The following code retrieves the value of the "EmpID" parameter:

```
empid = psDocument.GetParam("EmpID");
```

GetParameterString

Description: Returns the URL encoded string for the value of a parameter that you add to a PSArgClass object. This method is for use with 4GL targets only.

Applies to: PSArgClass

Syntax: *ArgObj*.GetParameterString ( )

Argument	Description
ArgObj	Object of the PSArgClass type containing the string you want to convert to URL encoding

Return value: String. The returned string is URL encoded.

Usage

The GetParameterString method is called automatically by the psPage.Redirect method. You can also call it directly to return a URL encoded value for a parameter string.

Examples

This example puts the URL-encoded value of the PSArgClass object into a text box. The value defined for the parameter is added to the PSArgClass object that is included in the psPage.Redirect call:

```
PSArgClass myParam=null;
String paramURL;
myParam = new PSArgClass();
myParam.addArg("param1", "my parameter value");
paramURL=myParam.GetParameterString();
sle_1.value=paramURL;
```

The result entered in the sle_1 text box is:

```
param1=my+parameter+value
```

See also

addArg
Redirect

GetPrecision

Description

Returns the number of decimal digits in a designated column in a cursor. This method is for use with JSP targets only.

Applies to

PSCursorClass objects

Syntax

cursorobject.GetPrecision (int *iCol*)

Argument	Description
cursorobject	A variable that contains a reference to an instance of PSCursorClass
iCol	Number of the column for which you want to obtain the precision

Return value

Integer

Examples

The following code retrieves the precision of the fifth column in the "mycursor" object:

```
li_precision = mycursor.GetPrecision(5);
```

See also

GetScale

GetResultSet

Description: Returns the result set in a cursor.

Applies to: PSCursorClass objects

Syntax: *cursorobject*.GetResultSet ( )

Argument	Description
cursorobject	A variable that contains a reference to an instance of PSCursorClass

Return value: java.sql.ResultSet

Examples: This example gets the java.sql.ResultSet from the cursor class through the GetResultSet method. The result set is manipulated using common java.sql.ResultSet methods:

```
<%
   try {
    PSConnectionClass dbconn = new
         PSConnectionClass(pageContext,
         "com.sybase.jdbc2.jdbc.SybDriver",
         "jdbc:sybase:Tds:localhost:2638",
         "dba","sql", false);
    PSCommandClass sqlcmd =
         dbconn.CreateCommand("select * from product");
    PSCursorClass mycursor = sqlcmd.Execute();
    java.sql.ResultSet rs = mycursor.GetResultSet();
    java.sql.ResultSetMetaData md = rs.getMetaData();
    psDocument.Write("<TABLE border=1>");
    psDocument.Write("<TR>");
    for (int col=1; col<= md.getColumnCount(); col++){
          psDocument.Write("<TD><B>"+
          md.getColumnName(col)+"</B></TD>");
         }
    psDocument.Write("</TR>");
    while(rs.next()) {
         psDocument.Write("<TR>");
         for(int i=1; i<= md.getColumnCount(); i++) {
             psDocument.Write("<TD>"+rs.getString(i) +
                "</TD>");
             }
         psDocument.Write("</TR>");
         }
    psDocument.Write("</TABLE>");
    rs.close();
    }
```

```
   catch(Exception e) {
    e.printStackTrace();
    }
%>
```

The following example uses the getResultSet method of the com.sybase.CORBA.jdbc11.SQL class to convert data from a tabular result set obtained by the "n_resultset" component running on EAServer. The connection to the database is handled by the component.

```
<%@ page import="com.sybase.CORBA.jdbc11.SQL" %>
...
<%
TabularResults.ResultSet trs=
      n_resultset.of_getresultset();
java.sql.ResultSet rs = SQL.getResultSet(trs );

while (rs.next())
  {psDocument.Write(Integer.toString(rs.getInt
      ("dept_id")));
  psDocument.Write("  ");
  psDocument.Write(rs.getString("dept_name"));
  psDocument.Write("  ");
  psDocument.WriteLn(Integer.toString(rs.getInt(
    "dept_head_id")));
}
%>
```

See also GetResultSetMetaData

GetResultSetMetaData

Description: Returns the metadata result set in a cursor.

Applies to: PSCursorClass objects

Syntax: *cursorobject*.GetResultSetMetaData ( )

Argument	Description
cursorobject	A variable that contains a reference to an instance of PSCursorClass

Return value: java.sql.ResultSetMetaData

Examples: The following example loops through the return value of the GetResultSetMetaData method to list all the columns and datatypes in the "employee" table of an ASA database accessed through JDBC:

```
try
   {
   PSConnectionClass dbconn = new
      PSConnectionClass(pageContext,
      "com.sybase.jdbc2.jdbc.SybDriver",
      "jdbc:sybase:Tds:localhost:2638",
       "dba","sql", true);
   PSCommandClass sqlcmd = dbconn.CreateCommand
       ("select * from employee");
   PSCursorClass rs = sqlcmd.Execute();
   java.sql.ResultSetMetaData meta =
       rs.GetResultSetMetaData();

   for( int col=1; col<=meta.getColumnCount(); col++ )
       psDocument.WriteLn( "Column: <B>" +
          meta.getColumnName(col) + "</B>\t, Type: <B>"
          + meta.getColumnTypeName(col)+"</B>" );
   }
catch( Exception e )
   {
   psDocument.WriteLn( " Error : " + e.toString() );
   }
```

See also: GetResultSet

GetRowCount

Description — Retrieves the number of rows in a cursor.

Applies to — PSCursorClass objects

Syntax — *cursorobject*.GetRowCount( )

Argument	Description
cursorobject	A variable that contains a reference to an instance of PSCursorClass

Return value — Integer

Usage — At runtime, GetRowCount calculates the number of rows in the ResultSet object used by the cursor instance.

Examples — The following example gets the row count for a cursor that retrieves rows from the customer table. In this example, "myquery" is a variable of type String, "myconnect" is a variable of type PSConnectionClass, "mycursor" is a variable of type PSCursorClass, and "rowcount" is a variable of type int:

```
myquery = "SELECT customer.cust_id, customer.cust_name
       FROM DBA.customer";
mycursor = myconnect.CreateCursor(myquery);
rowcount = mycursor.GetRowCount();
```

GetScale

Description — Returns the number of digits to the right of the decimal point for a designated column in a cursor. This method is for use in JSP targets only.

Applies to — PSCursorClass objects

Syntax — *cursorobject*.GetScale (int *iCol*)

Argument	Description
cursorobject	A variable that contains a reference to an instance of PSCursorClass
iCol	Number of the column for which you want to obtain the scale

Return value — Integer

Examples

The following code retrieves the scale of the fifth column in the “mycursor” object:

```
li_scale = mycursor.GetScale(5);
```

See also

CreateCursor
GetPrecision

GetValue

Retrieves the value of a column in a cursor or retrieves the value of a session variable.

To get the value of	Use
A column in a cursor (obsolete)	Syntax 1
A session variable	Syntax 2

Syntax 1 For PSCursorClass objects

Description

Retrieves the value of a column in a cursor.

Obsolete syntax
This method syntax is obsolete. It was previously used with ASP and PowerDynamo Web targets. For JSP targets, use the GetColumn<DataType> methods instead, where <DataType> is the datatype of the value you want to retrieve.

Applies to

PSCursorClass objects

Syntax

cursorobject.GetValue(*field*)

Return value

The value of the specified column

Syntax 2 For the PSSessionClass object

Description

Retrieves the value of a session variable.

Applies to

PSSessionClass object

Syntax psSession.GetValue(*propname*)

Argument	Description
propname	The name of a session variable

Return value String. Returns the value of the specified session variable. If the property does not exist, GetValue returns null.

Usage At runtime, GetValue accesses a user-defined property of the session object.

Examples The following example retrieves the values of the *UserID* and *Password* session variables:

```
userid = psSession.GetValue("UserID");
password = psSession.GetValue("Password");
```

IsTrace

Description Indicates whether tracing is enabled for a 4GL JSP target.

Applies to psPage object

Syntax psPage.IsTrace ( )

Return value boolean

Usage This method can be used to check before calling the Trace method multiple times.

Examples This example checks to see if tracing is on using the IsTrace method. If tracing is not on, it turns tracing on, prints out a trace statement, and turns tracing off:

```
boolean bTraceOn;
bTraceOn = psPage.IsTrace();
If (!bTraceOn)  {
    psPage.SetTrace(true); // turn on trace
    psPage.Trace("print out some trace information");
    psPage.SetTrace(false); // turn off trace
}
```

See also SetTrace
Trace

MapPath

Description — Maps a relative or virtual path to a physical path on the server.

Obsolete method
This method is obsolete. It was previously used with ASP Web targets that are no longer supported.

Applies to — PSServerClass object

Syntax — psServer.MapPath(*path*)

Return value — Returns a physical path on the server.

Move

Description — Moves to an absolute row in a cursor.

Applies to — PSCursorClass objects

Syntax — *cursorobject*.Move(*rownumber*)

Argument	Description
cursorobject	A variable that contains a reference to an instance of PSCursorClass.
rownumber	An absolute row number in the cursor. The array of rows starts with index zero.

Return value — Boolean. Returns true if the end of the cursor has been reached, and false if it has not.

Usage — At runtime, Move calls the absolute method on the ResultSet object to move the cursor to the entered *rownumber*. The Move method can work only if the JDBC driver you use supports the result set types TYPE_SCROLL_INSENSITIVE and TYPE_SCROLL_SENSITIVE.

Examples — The following example moves to row 10 of the “mycursor” object. In this example, “myquery” is a variable of type String, “mycursor” is a variable of type PSCursorClass, “myconnect” as a variable of type PSConnectionClass, and “eof” is a variable of type boolean. Because the row index is zero-based, the Move function specifies 9 as the row number:

```
myquery = "SELECT customer.cust_id, customer.cust_name
       FROM DBA.customer";
```

```
mycursor = myconnect.CreateCursor(myquery);
eof = mycursor.Move(9);
if ( eof == true )
       {
       psDocument.Write ("That row does not exist");
       }
```

MoveFirst

Description — Moves to the first row in a cursor.

Applies to — PSCursorClass objects

Syntax — *cursorobject*.MoveFirst( )

Argument	Description
cursorobject	A variable that contains a reference to an instance of PSCursorClass

Return value — Boolean. Returns true if the end of the cursor has been reached, and false if it has not.

Usage — At runtime, MoveFirst calls the first method of the ResultSet object.

Examples — The following example moves to the first row in the "mycursor" object:

```
eof = mycursor.MoveFirst();
if (eof == true )
 {
  psDocument.Write("End of cursor has been reached.");
 }
```

See also — CreateCursor
Move
MoveLast
MoveNext
MovePrevious

MoveLast

Description: Moves to the last row in a cursor.

Applies to: PSCursorClass objects

Syntax: *cursorobject*.MoveLast( )

Argument	Description
cursorobject	A variable that contains a reference to an instance of PSCursorClass

Return value: Boolean. Returns true if the end of the cursor has been reached, and false if it has not.

Usage: At runtime, MoveLast calls the last method on the ResultSet object.

Examples: The following example moves to the last row in the "mycursor" object:

```
eof = mycursor.MoveLast();
if (eof == true )
 {
 psDocument.Write("End of cursor has been reached.");
 }
```

See also:
CreateCursor
Move
MoveFirst
MoveNext
MovePrevious

MoveNext

Description — Moves to the next row in a cursor.

Applies to — PSCursorClass objects

Syntax — *cursorobject*.MoveNext( )

Argument	Description
cursorobject	A variable that contains a reference to an instance of PSCursorClass

Return value — Boolean. Returns true if the end of the cursor has been reached, and false if it has not.

Usage — At runtime, MoveNext calls the next method on the ResultSet object.

Examples — The following example uses MoveNext to process all of the rows in a cursor. Each time the cursor position changes, it writes the column values for the current row to the HTML page:

```
myquery = "SELECT products.prod_id, products.prod_name
       FROM DBA.products";
mycursor = myconnect.CreateCursor(myquery);
if ( myconnect.GetError() == null )
       {
       for (i=0; (!mycursor.EOF()); i++)
           {
           psDocument.Write(mycursor.GetValue(0) + " " +
               mycursor.GetValue(1));
           psDocument.Write("<BR>");
           mycursor.MoveNext();
           }
       }
else {
       dberror = true;
       }
    if ( dberror == true )
       {
       errobj = myconnect.GetError();
       str = errobj.GetCode() + " " +
        errobj.GetMessage();
       psDocument.Write ( str );
       }
```

See also — CreateCursor
Move
MoveFirst

MoveLast
MovePrevious

MovePrevious

Description — Moves to the previous row in a cursor.

Applies to — PSCursorClass objects

Syntax — *cursorobject*.MovePrevious( )

Argument	Description
cursorobject	A variable that contains a reference to an instance of PSCursorClass

Return value — Boolean. Returns true if the end of the cursor has been reached, and false if it has not.

Usage — At runtime, MovePrevious calls the previous method on the ResultSet object.

Examples — The following moves to the previous row in the "mycursor"object:

```
eof = mycursor.MovePrevious();
if (eof == true )
    {
psDocument.Write("End of cursor has been reached.");
    }
```

See also — CreateCursor
Move
MoveFirst
MoveLast
MoveNext

ObjectModelType

Description	Identifies the application server you are running.
Applies to	PSServerClass object
Syntax	psServer.ObjectModelType( )
Return value	String
Usage	At runtime, ObjectModelType returns the string "JSPObject". Because ASP and PowerDynamo Web target object models are no longer supported, this method has limited usefulness.

ObjectModelVersion

Description	Returns the version of the Web Target object model you are using.
Applies to	PSServerClass object
Syntax	psServer.ObjectModelVersion( )
Return value	String
Usage	At runtime, ObjectModelVersion returns the string value for the PowerBuilder Web Target object model. ObjectModelVersion returns "11.0" for applications created using PowerBuilder 11.
Examples	The following example performs different logic depending on the version of the Web Target object model you are using:

```
version = psServer.ObjectModelVersion();
if (version == "10.0") {
// Perform 10.x-specific logic
}
else {
// Perform alternative logic
}
```

Path

Description	Returns the path portion of the URL for the current document. The path does not include the file name.
Applies to	PSDocumentClass object
Syntax	psDocument.Path( )
Return value	String.
Usage	At runtime, Path extracts the URL path for the current document from the PATH_INFO server environment variable.
Examples	The following code returns the path portion of the current URL into a string variable called *mypath*. If the URL for the current document is *http://MyServer/Files/myscript.jsp*, the value of *mypath* is set to "/Files":

```
mypath = psDocument.Path( );
```

Redirect

Redirects the client's browser to another page.

To redirect	Use
From a 4GL Web page	Syntax 1
From a Web page that is not 4GL Web-enabled	Syntax 2

Syntax 1 For 4GL Web pages

Description

Redirects the client's browser to another page. Use this method to navigate programmatically to a new page. This method is for 4GL targets only.

Applies to

psPage object

Syntax

psPage.Redirect (string *destination*, PSArgClass *argument*)

Argument	Description
destination	This must be a URL.
argument	If you do not use null for this value, you must construct an object with PSArgClass and call the addArg method on that object to add parameter names and values.

Return value

None

Usage

You define the parameters that you pass in the PSArgClass object. The Redirect method calls the GetParameterString method on the PSArgClass object to return a URL-encoded string.

If a redirect is done before HTML generation starts, HTML generation will not occur but will still trigger the RequestFinish event. If called in or after the BeforeGenerate event, generation will complete, but the generated page will not be sent to the client browser.

If Redirect is called more than once, earlier calls to this method are overwritten. Only the last call will have any effect.

Examples

This example uses the Redirect method with parameters, where "param1" is a parameter defined on the Parameters tab of the Page Properties dialog box:

```
PSArgClass myParam=null;
myParam = new PSArgClass();
myParam.addArg("param1", param1);
psPage.Redirect("page_2.jsp", myParam);
```

Syntax 2 For pages not 4GL Web enabled

Description: Redirects the current request to another URL.

Applies to: PSDocumentClass object

Syntax: psDocument.Redirect(*URL*)

Argument	Description
URL	The URL to which the current request is being directed

Return value: None.

Usage: At runtime, Redirect calls the sendRedirect method on the response object.

Examples: The following example redirects the current request to the Sybase Web site:

```
psDocument.Redirect("http://www.sybase.com");
```

ReportError

Description: Indicates that a server-side error has occurred. This method is for use with 4GL targets only.

Applies to: psPage object

Syntax: **psPage.ReportError** (string *location*, string *cause*, string *message*)

Argument	Description
location	The *objectName.methodName* on which the error is detected
cause	Cause for the processing error
message	The system error message

Return value: None

Usage: The ReportError method simplifies error processing for 4GL Web pages. It causes the ServerError event to be triggered and, depending on the return value of ServerError, adds the error to the error log. Because most events are triggered before generation occurs, the standard way of reporting errors (by doing a psDocument.Write) cannot be used, nor can you centralize error processing using psDocument.Write.

The arguments for the ReportError method are passed to the ServerError event. If the default generation is used, an HTML BR tag is appended to each message, and the message can contain valid HTML to perform formatting. If you display the message in an alert box, do not use HTML tags in the message.

When the server detects an error, it also calls ReportError, passing in the error location as *objectName.methodName*, the cause of the error, and any applicable system message. If you are assigning the method arguments directly, you should make sure the location argument is meaningful in determining where the error occurred.

Only call before HTML generation
If ReportError is called after the start of generation, the errors are not added to the list; the list is fixed when generation starts. To report errors after generation starts, use psDocument.Write.

Examples

This example passes variables for arguments and uses HTML formatting tags:

```
psPage.ReportError(myLocation, myCause,
"<B><FONT color=green>"+myMessage+"</FONT></B>")
```

See also

ServerError
WriteErrorsToDocument

setCharacterEncoding

Description

Sets the charset encoding for the PSArgClass object. This method is for use with 4GL targets only.

Applies to

PSArgClass

Syntax

ArgObj.setCharacterEncoding (String *enc*)

Argument	Description
ArgObj	Object of the PSArgClass type for which you want to change the character set
enc	Character set encoding you want to use for adding arguments to a PSArgClass object

Return value

None.

Usage

If you are adding arguments to a URL in a psPage.Redirect call, you might need to change the character set used by the PSArgClass argument.

You can get the character encoding used by the PSArgClass object by calling the getCharacterEncoding method.

Examples

This example sets the value of the charset encoding used by PSArgClass to simplified Chinese:

```
PSArgClass myURLParam=null;
myURLParam = new PSArgClass();
myURLParam.setCharacterEncoding("gb2312");
```

See also

addArg
getCharacterEncoding

SetColumnLink

Description

Establishes a link on a column that is passed from the database to the Web DataWindow control. This link lets the Web DataWindow DTC pass data to another page.

Applies to

PSDataWindowClass object

Syntax

PSDataWindowClassObject.SetColumnLink(*columnName*, *link*, *linkArgs*, *linkTarget*)

Argument	Description
columnName	A string that specifies the name of the column that you want to link to a target page.
link	A string that specifies the URL target of a link (HTML A element) from a data item in the column.
linkArgs	A string that specifies the arguments passed with the *link* argument. This string is appended to the *link* argument when the HTML is generated. This argument has the form: `argname='exp'` where "argname" is a page parameter that gets passed with the URL and "exp" is a DataWindow expression that gets evaluated. The value of the expression is converted using URL encoding.
linkTarget	A string that specifies the name of a target frame or window for the link specified in the *link* argument. The target is included in the HTML element using the HTML TARGET attribute. You can use *linkTarget* to link from a master to a detail page by specifying a different window or frame for the detail page.

Return value

None.

Usage

At runtime SetColumnLink sets up the link on the passed column. It allows you to make master and detail links easily from server scripts.

Examples

In the following example, the column called “emp_id” links to the file *empdetail.htm*, passing the “emp_id” as an argument:

```
htmlDwObj1.SetColumnLink("emp_id",
       "empdetail.stm","emp_id='emp_id'");
```

SetSQL

Description

Sets the SQL statement for a command. This method is for use with ASP targets only.

Applies to

PSCommandClass objects

Syntax

commandobject.SetSQL(*SQLstring*)

Argument	Description
commandobject	A variable that contains a reference to an instance of PSCommandClass
SQLstring	A string that specifies the SQL statement

Return value

Boolean. Returns true if it succeeds and false if it fails.

Usage

At runtime, SetSQL uses services provided by the Web Target object model rather than native services of the application server.

Examples

The following example sets the SQL statement for a PSCommandClass object and then executes the SQL:

```
mycommand.SetSQL("SELECT * FROM products");
mycommand.Execute();
```

SetTrace

Description

Turns tracing for event processing on or off. Turning tracing on also enables you to include your own messages by calling the Trace method. This method is for use with 4GL targets only.

Applies to

psPage object

Syntax

psPage.SetTrace (boolean *bTraceOn*)

Argument	Description
bTraceOn	Generates a trace message if true

Return value

None

Usage

Tracing code for server-side event processing makes it easier to track down a problem. By programmatically setting SetTrace to true, or by selecting the Enable Trace check box on the Errors page of the Page Properties dialog box, you can see the details of server-side event processing at the top of your Web page.

To include your own messages in the trace, call the psPage.Trace method in event scripts. To offset your custom trace message, surround it with TraceIndent and TraceOutdent calls.

Tracing cannot be used in the AfterGenerate and RequestFinish events, because the trace would already have been generated at the top of the page. If there are any embedded new lines or carriage returns in the trace text, they will be turned into \n or \r as required. This preserves the indenting of the tracing.

Examples

This code fragment turns tracing on from a server-side event:

```
psPage.SetTrace (true);
```

See also

IsTrace
Trace
TraceIndent
TraceOutdent

SetValue

Description — Sets the value of a session variable.

Applies to — PSSessionClass object

Syntax — psSession.SetValue(*propname*, *value*)

Argument	Description
propname	The name of a session variable
value	The new value for the session variable

Return value — String. Returns the new value for the session variable.

Usage — At runtime, SetValue assigns a value to a user-defined property of the session object.

Examples — The following example sets the value of the *UserID* and *Password* session variables:

```
psSession.SetValue("UserID", userid);
psSession.SetValue("Password", password);
```

SetWeight

Description — Identifies the type of functionality included on your JSP page. As you include more functionality on your page, the size of the control increases. The largest (heaviest) but most feature-rich objects would support both client-side scripting and client-side formatting.

Applies to — PSDataWindowClass object

Syntax — PSDataWindowClassObject.SetWeight(*bAllowForm*, *bClientValidation*, *bClientEvents*, *bClientScriptable*, *bClientFormatting*)

Argument	Description
bAllowForm	Boolean indicating whether or not the object supports data input: • **true** (default) The object supports data input. • **false** The object does not support data input. The object will provide only navigation. *bAllowForm* must be set to true to set the *bClientValidation* and *bClientFormatting* arguments to true.

Argument	Description
bClientValidation	Boolean indicating whether or not the client validates the the syntax of the data entered by the user. Client-side validation can determine if the data is in a valid format for the database. The *bAllowForm* argument must be set to true to use client-side validation: • **true** (default) The client validates the syntax of the entries that users make. • **false** The client does not support syntax validation for entries that users make.
bClientEvents	Boolean indicating whether or not the object supports invoking client-side events: • **true** (default) Invoking client-side events is supported. • **false** Invoking client-side events is not supported.
bClientScriptable	Boolean indicating whether or not you can add scripts to manipulate the Web DataWindow control on the client. The scripts that you add would run on the client system. • **true** Client-side scripting is supported. • **false** (default) Client-side scripting is not supported.
bClientFormatting	Boolean indicating whether or not display formats are applied to newly entered data. The *bAllowForm* argument must be set to true to apply formatting: • **true** Newly entered data will be formatted. • **false** (default) Newly entered data will not be formatted.

Return value

1 indicates success, -1 indicates failure.

Usage

At runtime SetWeight determines the weight (or size) of the control by the functionality included for a Web target page.

Examples

In the following code fragment, the Web DataWindow object named "htmlDwObj1" supports validation of the syntax for entries that users make in the object:

```
htmlDwObj1.SetWeight(true,true);
```

Site

Description — Returns the domain name for the current document.

Applies to — PSDocumentClass object

Syntax — psDocument.Site( )

Return value — String

Usage — At runtime, Site retrieves the value of the SERVER_NAME server environment variable.

Examples — The following code fragment returns the domain name of the current document into a string variable called ”mydomain”:

```
mydomain = psDocument.Site( );
```

TestCompError

Description — Tests whether an EAServer component method that was just called had an error. This method is for use with 4GL targets only.

Applies to — psPage object

Syntax — **psPage.TestCompError** (string *location*)

Argument	Description
location	The *objectName.methodName* on which the error is detected

Return value — Boolean. The method returns true if an error occurred on an EAServer component method call. In that case, the ReportError method passes the location and the system message generated by the call. If no error occurred, the method returns false.

Usage — All EAServer component method calls done by the Web Target object model invoke this method. Add TestCompError calls after your own calls to components to check that they do not produce any errors.

You must make sure to generate stubs for all your EAServer components and compile them.

Examples — This call to the EAServer component method nvo_math.division returns a divide-by-zero error from the server when its second argument is set to zero. Calling TestCompError will cause the error to display on your Web page:

```
nvo_math.division(m1, m2);
```

```
psPage.TestCompError("call to nvo_math.division");
```

The following result displays on the Web page when the divisor is set to zero:

```
call to nvo_math.division:Component Call
Failed:Exception thrown: CTS.PBUserException: Divide by
zero;1;3;nvo_math;nvo_math;division; in method division
of class new_math/_st_nvo_math.
```

See also ReportError

Trace

Description Adds the message you specify to the internal trace buffer. This method is for 4GL Web pages only.

Applies to psPage object

Syntax **psPage.Trace** (string *message*)

Argument	Description
message	Message to be displayed in the internal trace buffer

Return value None

Usage The Trace method allows you to insert a custom message in the trace buffer for server-side event processing. You must turn tracing on before your message (and the rest of the trace buffer) can be displayed. You can do this by selecting the Enable Trace check box on the Errors page of the Page Properties dialog box or by calling psPage.SetTrace(true) in code that is parsed before your Trace call.

The trace message must be in plain text. It will be indented by the indent amount of the trace buffer and will appear on its own line, with blank spaces preserved. Embedding new lines in the message will disrupt the formatting of the trace. You can increase the indent level of your trace messages by surrounding your Trace method call with (multiple) calls to TraceIndent and TraceOutdent.

Examples This example adds a message to the trace buffer. The message is set off from the other event-processing trace messages by an additional indent space:

```
psPage.TraceIndent( );
psPage.Trace(MyVar + " is the value of MyVar at this
       time");
psPage.TraceOutdent( );
```

See also

IsTrace
SetTrace
TraceIndent
TraceOutdent

TraceIndent

Description

Increases the indent level of trace messages. This method is for 4GL Web pages only.

Applies to

psPage object

Syntax

psPage.TraceIndent ( )

Return value

None

Usage

Each call to the TraceIndent method increases the indent level of subsequent messages in the trace buffer. You can reduce the indent level of trace messages by calling TraceOutdent.

Examples

This example adds a message to the trace buffer. The message is set off from other event-processing trace messages by the additional indent level coded in this script:

```
psPage.TraceIndent( );
psPage.Trace(MyVar + " is the value of MyVar at this
       time");
psPage.TraceOutdent( );
```

See also

IsTrace
SetTrace
Trace
TraceOutdent

TraceOutdent

Description	Decreases the indent level of trace messages. This method is for 4GL Web pages only.
Applies to	psPage object
Syntax	**psPage.TraceOutdent** ( )
Return value	None
Usage	Each call to the TraceOutdent method decreases the indent level of subsequent messages in the trace buffer. You can increase the indent level of trace messages by calling TraceIndent.
Examples	This example adds a message to the trace buffer. It is set off from other event-processing trace messages by the additional indent level coded in this script:

```
psPage.TraceIndent( );
psPage.Trace(MyVar + " is the value of MyVar at this
       time");
psPage.TraceOutdent( );
```

See also	IsTrace SetTrace Trace TraceIndent

Type

Description	Identifies the Web server you are running.
Applies to	PSServerClass object
Syntax	psServer.Type( )
Return value	String
Usage	At runtime, Type gets the name of the Web server software from the SERVER_SOFTWARE server environment variable.

Examples

The following example tests to see which Web server is running and performs platform-specific logic that varies depending on the outcome of the test:

```
servertype = psServer.Type( );
if (servertype == "Apache Tomcat") {
// Perform platform-specific logic
}

if (servertype == "Jaguar Server") {
// Perform platform-specific logic
}
```

URLEncode

Description

Applies URL encoding rules to a string.

Applies to

PSServerClass object

Syntax

psServer.URLEncode(*string*)

Argument	Description
string	The string to which you want to apply URL encoding

Return value

String

Usage

At runtime, URLEncode calls the encode method of java.net.URLEncoder.

Examples

The following example applies URL encoding to a URL query string that contains a percent sign. This string needs to be encoded because the percent sign is itself used to indicate URL-encoded characters:

```
value1 = psServer.URLEncode("33%");
value2 = psServer.URLEncode("50%");
value3 = psServer.URLEncode("100%");

text = "<P>Increase salary by:</P>";
text += "<A HREF='" + "salary.jsp?increase=" + value1 +
       "'> 33% </A> <BR>";
text += "<A HREF='" + "salary.jsp?increase=" + value2 +
       "'> 50% </A> <BR>";
text += "<A HREF='" + "salary.jsp?increase=" + value3 +
       "'> 100% </A>";

psDocument.Write(text);
```

This code sends the following HTML to the browser:

```
<P>Increase salary by:</P>
<A HREF="salary.jsp?increase=33%25")%>"> 33% </A> <BR>
<A HREF="salary.jsp?increase=50%25")%>"> 50% </A> <BR>
<A HREF="salary.jsp?increase=100%25")%>"> 100% </A>
```

In the HTML output shown above, the number 25 is the ANSI code for the percent character. The percent sign indicates that the value that follows (in this case, 25) is a URL-encoded character.

Version

Description	Returns the version of the Web server you are running.
Applies to	PSServerClass object
Syntax	psServer.Version( )
Return value	String
Usage	At runtime, Version gets the version of the Web server software from the SERVER_SOFTWARE server environment variable.
Examples	The following example tests to see which application server is running and performs platform-specific logic that varies depending on the outcome of the test:

```
server = psServer.Type();
version = psServer.Version();
if (server == "Apache Tomcat") {
       if (version == "5.0.0") {
       // Perform 5.0.0-specific logic
       }

       if (version == "4.1.13") {
       //Perform 4.1.13-specific logic
       }
}

else {
// Perform EAServer-specific logic
}
```

Write

Description: Writes an output string to a document.

Applies to: PSDocumentClass object

Syntax: psDocument.Write(*string*)

Argument	Description
string	The output string

Return value: None

Usage: At runtime, Write calls the print method on javax.servlet.jsp.JspWriter.

Examples: The following example writes the string “Hello World!” to the current document:

```
psDocument.Write("Hello World!");
```

WriteErrorsToDocument

Description: Writes the current errors at the current place in the page. This method is for 4GL Web pages only.

Applies to: psPage object

Syntax: **psPage.WriteErrorsToDocument** ( )

Return value: None

Usage: This method must only be called in a server-side script tag, between HTML BODY tags. It allows you to place messages from the error buffer at a precise location on the page.

ReportError must first be called to populate the error buffer. ReportError is called either programmatically or automatically (when the server detects an error). You can then turn off the error display by setting the psPage.showErrorsOnPage property to false.

Examples

This example tests whether the user ID is the same as the password entered on a logon page. If the ID and password are not the same, a ReportError method can be called (with an "incorrect password" message as an argument) in the RequestStart or FirstTime event to write an error message to the error buffer. The WriteErrorsToDocument method can then cause the error message to display at the place in the page where this server-side script is called:

```
<P>
<% user = psDocument.GetParam("user");
password= psDocument.GetParam("password");

if (user == password ) {
  psSession.SetValue("user", user);
  psDocument.Redirect("Home.htm");
  }
else {
  psPage.WriteErrorsToDocument ();
} %>
</P>
```

See also

ReportError

WriteLn

Description

Writes an output string to the document that ends with a line break.

Applies to

PSDocumentClass object

Syntax

psDocument.WriteLn(*string*)

Argument	Description
string	The output string

Return value

None

Usage

Calling this method adds a line break in the HTML source, not in the final HTML output. For a line break in the HTML output, you still must add a
 element to the HTML source.

At runtime, WriteLn calls the print method on javax.servlet.jsp.JspWriter.

Examples

The following example uses WriteLn to write the string "Hello World!" to the current document and adds a line break in the HTML output as well as in the HTML source:

```
psDocument.WriteLn("<P>Hello World! <BR>");
```

CHAPTER 5

Custom Tag Reference

About this chapter

This chapter describes the custom tags in the Web DataWindow tag library that is deployed by default with the Web Target server-side object model.

Contents

About the Web DataWindow custom tag library

You can use the Web DataWindow custom tag library to specify the parameters and values required by a Web DataWindow on a JSP page. The tag library is defined in the file *DataWindow110.tld*. To use the tag library, place the *DataWindow110.tld* file in a *WEB-INF/tlds* directory in your Web applications Source directory. The tag classes are included in the *jspobject.jar* file that is deployed with all PowerBuilder JSP Web applications.

The tag library contains two tags, DataWindow and DWColumnLink. The DWColumnLink tag is an inner tag; it can be used only inside the DataWindow tag.

Attributes have three subelements: *name*, *required*, and *rtexprvalue*. The *rtexprvalue* element is optional and indicates whether the attribute's value can be dynamically calculated at runtime.

DataWindow

Description

Sets parameters for a Web DataWindow on a JSP page.

Attributes

All DataWindow tag attributes are required unless noted in the Description column. The value of the *rtexprvalue* subelements is true for all attributes.

Attributes of DataWindow tag	Java type	Description
action	String	See the *action* argument for the SetAction DataWindow method in the *DataWindow Reference*.
allowForm	boolean	See the *bAllowForm* argument for the PSDataWindowClass.SetWeight method.
argument	String	See the *argument* argument for the RetrieveEx DataWindow method in the *DataWindow Reference*.
clientEvents	boolean	See the *bClientEvents* argument for the PSDataWindowClass.SetWeight method.
clientFormatting	boolean	See the *bClientFormatting* argument for the PSDataWindowClass.SetWeight method.
clientScriptable	boolean	See the *bclientScriptable* argument for the PSDataWindowClass.SetWeight method.
clientValidation	boolean	See the *bClientValidation* argument for the PSDataWindowClass.SetWeight method.
context	String	See the *context* argument for the SetAction DataWindow method in the *DataWindow Reference*.
database	String	See the *database* constructor for PSConnectionParmsClass.
dbms	String	See the *dbms* constructor for PSConnectionParmsClass.
dbparm	String	See the *dbparm* constructor for PSConnectionParmsClass.
dwHTMLObjectName	String	See the *objectname* argument for the SetHTMLObjectName DataWindow method in the *DataWindow Reference*.
dwName	String	See the *dwName* property for PSDataWindowSourceClass.
fourGLWeb	boolean	You must set this to true in a 4GL page.
id	String	Optional identifier.
jaglogid	String	(Optional) See the *userId* constructor for PSJaguarConnection.

Attributes of DataWindow tag	Java type	Description
jaglogpass	String	(Optional) See the *password* constructor for PSJaguarConnection.
jagservername	String	See the *serverName* constructor for PSJaguarConnection.
libName	String	See the *sourceFileName* property for PSDataWindowSourceClass.
lock	String	See the *lock* constructor for PSConnectionParmsClass.
logid	String	See the *user* constructor for PSConnectionParmsClass.
logpass	String	See the *password* constructor for PSConnectionParmsClass.
pageSize	String	(Optional) See the *pagesize* argument for the SetPageSize DataWindow method in the *DataWindow Reference*.
selfLink	String	The URL for the current page.
selfLinkArg	String	Page parameters to be passed to the server. The syntax is: *argname*='*exp*'{ | *argname* = '*exp*' } ... where *argname* is a page parameter and *exp* is a DataWindow expression whose value is a string. See HTMLGen.property in the *DataWindow Reference* for more information.
servername	String	See the *serverName* constructor for PSConnectionParmsClass.

DWColumnLink

Description

Establishes a link on a column that is passed from the database to the Web DataWindow control. This link lets the Web DataWindow DTC pass data to another page.

Attributes

All DWColumnLink tag attributes are required. The value of the *rtexprvalue* subelements is unspecified for all attributes.

Attributes of DWColumnLink tag	Java type	Description
sColumnName	String	The name of the column that you want to link to a target page.
sColLink	String	The URL target of a link from a data item in the column.
sColLinkArgs	String	The arguments passed with the *link* argument.
sColLinkTarget	String	The name of a target frame or window for the link specified in the Link argument. The target is included in the HTML element using the HTML TARGET attribute. You can use sColLinkTarget to link from a master to a detail page by specifying a different window or frame for the detail page.

Example using the Web DataWindow custom tag library

This example shows two JSP pages that use the DataWindow tag. The first, *Departments.jsp*, uses a nested DWColumnLink tag to pass data to the *Employees.jsp* page. The link is from the dept_id column of the d_departments DataWindow that uses the department table in the EAS Demo database. In the DataWindow painter, you must set the tab order for this column to 0 or the Protect property to 1 in order for the link to work.

The deployment descriptor for the application must include a taglib element that associates the short name "DW110" with the *DataWindow110.tld* file in the Web application's */WEB-INF/tlds* directory:

```
<taglib>
  <taglib-uri>/DW110</taglib-uri>
  <taglib-location>/WEB-INF/tlds/DataWindow110.tld
  </taglib-location>
</taglib>
```

The deployment descriptor for the application is the file *web.xml*, which resides in the Web application's *WEB-INF* directory. For more information, see the section on editing a JSP deployment configuration in the *Working with JSP Targets* book.

Departments.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0
Transitional//EN">

<%-- Import tag class--%>
<%@ taglib prefix="webdw" uri="/DW110"  %>
<HTML>
<HEAD>
<META HTTP-EQUIV="PowerSiteData" NAME="SERVERLANGUAGE"
CONTENT="JavaScript">
<TITLE></TITLE>
<META http-equiv="Content-Type" content="text/html">
<META content="MSHTML 5.50.4522.1800" name="GENERATOR">
</HEAD>
<BODY PSPARAMS="">

<%-- Use DataWindow custom tag--%>
<webdw:DataWindow argument=""
   selfLinkArg=""
   logpass=""
   jaglogpass=""
   dbms="ODBC"
```

```
   servername=""
   clientScriptable="true"
   clientFormatting="true"
   action=""
   selfLink="dwpage2.jsp"
   jaglogid="jagadmin"
   dwHtmlObjectName="dwTest"
   logid="" lock=""
   clientEvents="true"
   clientValidation="true"
   libName="f:\\Mywork\\Pbjsp\\d_departments.srd"
   database=""
   dbparm="ConnectString='DSN=EAS Demo DB V10;
      UID=dba;PWD=sql',ConnectOption=
      'SQL_DRIVER_CONNECT,SQL_DRIVER_NOPROMPT'"
   jagservername="myEASserver:9000"
   dwName=""
   context=""
   allowForm="true"
   >
<webdw:DWColumnLink
   sColLink="Employees.jsp"
   sColLinkArgs="dept_id='dept_id'"
   sColLinkTarget=""
   sColumnName="dept_id">
</webdw:DWColumnLink>
</webdw:DataWindow>
</BODY>
</HTML>
```

Employees.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0
Transitional//EN">
<%@ taglib prefix="webdw" uri="/DW110"  %>
<HTML>
<HEAD><TITLE>
DataWindowJSP Example
</TITLE></HEAD>
<BODY>
<H2>
Basic JSP Datawindow: Employee List Report
</H2>
<%! String strDept; %>
<%  strDept = psDocument.GetParam("dept_id");%>

<webdw:DataWindow
libName="d:\\Mywork\\Pbjsp\\d_employees.srd"
   dwName=""
```

```
    allowForm="true"
    clientValidation="true"
    clientEvents="true"
    clientScriptable="true"
    clientFormatting="true"
    dbms="ODBC"
    dbparm="ConnectString='DSN=EAS Demo DB V10;
       UID=dba;PWD=sql',ConnectOption=
       'SQL_DRIVER_CONNECT,SQL_DRIVER_NOPROMPT'"
    lock=""
    logid=""
    logpass=""
    database=""
    servername=""
    jagservername="myEASserver:9000"
    jaglogid="jagadmin"
    jaglogpass=""
    selfLink="Employees.jsp"
    selfLinkArg=""
    action=""
    context=""
    argument="<%=strDept%>"
    dwHtmlObjectName="dwTest"
    pageSize="10"
 >

 </webdw:DataWindow>
 </BODY></HTML>
```

JSP deployment controller adds server script

When you deploy the JSP target from PowerBuilder, the JSP deployment controller adds the following server script to the top of each JSP page:

```
<%@ page import="com.sybase.powerbuilder.jspobject.*" %>
<%
   // global instance for the page
   PSDocumentClass psDocument = new PSDocumentClass
      (request, response, out, application);
   PSSessionClass psSession = new
      PSSessionClass(session);
   PSServerClass psServer = new
      PSServerClass(psDocument);
%>
```

Index

M

O

P

R

S

T

U

V

W