This publication pertains to Sybase software and to any subsequent release until otherwise indicated in new editions or technical notes. Information in this document is subject to change without notice. The software described herein is furnished under a license agreement, and it may be used or copied only in accordance with the terms of that agreement.

To order additional documents, U.S. and Canadian customers should call Customer Fulfillment at (800) 685-8225, fax (617) 229-9845.

Customers in other countries with a U.S. license agreement may contact Customer Fulfillment via the above fax number. All other international customers should contact their Sybase subsidiary or local distributor. Upgrades are provided only at regularly scheduled software release dates. No part of this publication may be reproduced, transmitted, or translated in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without the prior written permission of Sybase, Inc.


Unicode and the Unicode Logo are registered trademarks of Unicode, Inc.

All other company and product names used herein may be trademarks or registered trademarks of their respective companies.

Use, duplication, or disclosure by the government is subject to the restrictions set forth in subparagraph (c)(1)(ii) of DFARS 52.227-7013 for the DOD and as set forth in FAR 52.227-19(a)-(d) for civilian agencies.
Contents

About This Book .......................................................................................................................... vii

CHAPTER 1 Web Target Classes and Objects ................................................................. 1
  PSArcClass ........................................................................................................ 2
  PSButtonClass .................................................................................................. 2
  PSCheckBoxClass ............................................................................................. 3
  PSCCommandClass ............................................................................................ 4
  PSCConnectionClass .......................................................................................... 5
  PSCConnectionParmsClass .................................................................................. 7
  PSCursorClass ..................................................................................................... 9
  PSDataWindowClass .......................................................................................... 10
  PSDataWindowSourceClass ............................................................................... 14
  PSDocumentClass .............................................................................................. 15
  PSDropDownListClass ........................................................................................ 16
  PSErrorClass ........................................................................................................ 17
  PSImageClass ...................................................................................................... 18
  PSJaguarConnection .......................................................................................... 18
  PSLinkClass .......................................................................................................... 20
  PSNamedConnectionParmsClass ........................................................................ 21
  psPage .................................................................................................................. 22
  PSPasswordClass ................................................................................................. 24
  PSRadioGroupClass ............................................................................................ 25
  PSServerClass ..................................................................................................... 26
  PSSessionClass .................................................................................................... 27
  PSSStaticTextClass ............................................................................................ 27
  PSTTextAreaClass ............................................................................................... 28
  PSTextClass ........................................................................................................... 29
  PSWebDataWindowClass ..................................................................................... 30

CHAPTER 2 4GL Web Target Events ............................................................................... 37
  AfterAction .......................................................................................................... 38
  AfterBinding ......................................................................................................... 39
  AfterGenerate ......................................................................................................... 40
### Contents

<table>
<thead>
<tr>
<th>Event</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>BeforeAction</td>
<td>41</td>
</tr>
<tr>
<td>BeforeBinding</td>
<td>42</td>
</tr>
<tr>
<td>BeforeGenerate</td>
<td>43</td>
</tr>
<tr>
<td>FirstTime</td>
<td>44</td>
</tr>
<tr>
<td>ItemChanged</td>
<td>45</td>
</tr>
<tr>
<td>RequestFinish</td>
<td>46</td>
</tr>
<tr>
<td>RequestStart</td>
<td>46</td>
</tr>
<tr>
<td>ServerAction</td>
<td>47</td>
</tr>
<tr>
<td>ServerError</td>
<td>48</td>
</tr>
<tr>
<td>Validate</td>
<td>50</td>
</tr>
<tr>
<td>ValidationError</td>
<td>52</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>CHAPTER 3</strong></td>
<td></td>
</tr>
<tr>
<td>Web DataWindow Events</td>
<td>55</td>
</tr>
<tr>
<td>Using server-side events with the Web DataWindow DTC</td>
<td>55</td>
</tr>
<tr>
<td>AfterAction</td>
<td>56</td>
</tr>
<tr>
<td>AfterRetrieve</td>
<td>56</td>
</tr>
<tr>
<td>AfterUpdate</td>
<td>57</td>
</tr>
<tr>
<td>BeforeAction</td>
<td>57</td>
</tr>
<tr>
<td>BeforeRetrieve</td>
<td>58</td>
</tr>
<tr>
<td>BeforeUpdate</td>
<td>58</td>
</tr>
<tr>
<td>OnDBError</td>
<td>59</td>
</tr>
<tr>
<td>Validate</td>
<td>60</td>
</tr>
<tr>
<td>ValidationError</td>
<td>61</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>CHAPTER 4</strong></td>
<td></td>
</tr>
<tr>
<td>Web Target Methods</td>
<td>63</td>
</tr>
<tr>
<td>Abandon</td>
<td>65</td>
</tr>
<tr>
<td>addArg</td>
<td>66</td>
</tr>
<tr>
<td>Alert</td>
<td>67</td>
</tr>
<tr>
<td>ClearError</td>
<td>68</td>
</tr>
<tr>
<td>CreateCommand</td>
<td>69</td>
</tr>
<tr>
<td>CreateConnection</td>
<td>70</td>
</tr>
<tr>
<td>CreateCursor</td>
<td>72</td>
</tr>
<tr>
<td>EOF</td>
<td>74</td>
</tr>
<tr>
<td>Execute</td>
<td>75</td>
</tr>
<tr>
<td>File</td>
<td>76</td>
</tr>
<tr>
<td>FillRetrievalArgs</td>
<td>77</td>
</tr>
<tr>
<td>Generate</td>
<td>78</td>
</tr>
<tr>
<td>GenerateXHTML</td>
<td>79</td>
</tr>
<tr>
<td>GenerateXMLWeb</td>
<td>80</td>
</tr>
<tr>
<td>getCharacterEncoding</td>
<td>82</td>
</tr>
<tr>
<td>getCode</td>
<td>83</td>
</tr>
<tr>
<td>GetColumnCount</td>
<td>84</td>
</tr>
<tr>
<td>GetColumn&lt;DataType&gt;</td>
<td>85</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>GetColumnLength</td>
<td>86</td>
</tr>
<tr>
<td>GetColumnName</td>
<td>87</td>
</tr>
<tr>
<td>GetColumnType</td>
<td>88</td>
</tr>
<tr>
<td>GetColumnTypeName</td>
<td>89</td>
</tr>
<tr>
<td>GetConnection</td>
<td>90</td>
</tr>
<tr>
<td>GetEnv</td>
<td>91</td>
</tr>
<tr>
<td>GetError</td>
<td>93</td>
</tr>
<tr>
<td>GetMessage</td>
<td>95</td>
</tr>
<tr>
<td>getNextError</td>
<td>96</td>
</tr>
<tr>
<td>GetParam</td>
<td>97</td>
</tr>
<tr>
<td>GetParameterString</td>
<td>98</td>
</tr>
<tr>
<td>GetPrecision</td>
<td>99</td>
</tr>
<tr>
<td>GetResultSet</td>
<td>100</td>
</tr>
<tr>
<td>GetResultSetMetaData</td>
<td>102</td>
</tr>
<tr>
<td>GetRowCount</td>
<td>103</td>
</tr>
<tr>
<td>GetScale</td>
<td>104</td>
</tr>
<tr>
<td>GetValue</td>
<td>105</td>
</tr>
<tr>
<td>IsTrace</td>
<td>107</td>
</tr>
<tr>
<td>MapPath</td>
<td>108</td>
</tr>
<tr>
<td>Move</td>
<td>109</td>
</tr>
<tr>
<td>MoveFirst</td>
<td>110</td>
</tr>
<tr>
<td>MoveLast</td>
<td>111</td>
</tr>
<tr>
<td>MoveNext</td>
<td>112</td>
</tr>
<tr>
<td>MovePrevious</td>
<td>113</td>
</tr>
<tr>
<td>ObjectModelType</td>
<td>114</td>
</tr>
<tr>
<td>ObjectModelVersion</td>
<td>115</td>
</tr>
<tr>
<td>Path</td>
<td>116</td>
</tr>
<tr>
<td>Redirect</td>
<td>117</td>
</tr>
<tr>
<td>ReportError</td>
<td>119</td>
</tr>
<tr>
<td>setCharacterEncoding</td>
<td>120</td>
</tr>
<tr>
<td>SetColumnLink</td>
<td>121</td>
</tr>
<tr>
<td>SetSQL</td>
<td>122</td>
</tr>
<tr>
<td>SetTrace</td>
<td>123</td>
</tr>
<tr>
<td>SetValue</td>
<td>124</td>
</tr>
<tr>
<td>SetWeight</td>
<td>125</td>
</tr>
<tr>
<td>Site</td>
<td>126</td>
</tr>
<tr>
<td>TestCompError</td>
<td>127</td>
</tr>
<tr>
<td>Trace</td>
<td>128</td>
</tr>
<tr>
<td>TraceIndent</td>
<td>129</td>
</tr>
<tr>
<td>TraceOutdent</td>
<td>130</td>
</tr>
<tr>
<td>Type</td>
<td>131</td>
</tr>
<tr>
<td>URLEncode</td>
<td>132</td>
</tr>
<tr>
<td>Version</td>
<td>133</td>
</tr>
<tr>
<td>Write</td>
<td>134</td>
</tr>
</tbody>
</table>
About This Book

Audience
This book is for programmers who will be using PowerBuilder® to build applications for the Web.

How to use this book
This book describes classes that make up the Web Target object model and its 4GL extensions. This includes syntax, usage notes, and examples for the methods on these classes, as well as descriptions of server-side events from which you can call these methods.

PowerBuilder supports Java Server Pages (JSP) and Active Server Pages (ASP) as Web targets. The same development environment is used for creating JSP and HTML pages. Application of classes and methods to specific types of Web targets is noted.

Related documents
- Working with Web and JSP Targets
- DataWindow Reference

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

Web 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 Web site (ASP) and 4GL and non-4GL JSP targets.

Contents

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSArgClass</td>
<td>2</td>
</tr>
<tr>
<td>PSButtonClass</td>
<td>2</td>
</tr>
<tr>
<td>PSCheckBoxClass</td>
<td>3</td>
</tr>
<tr>
<td>PSCommandClass</td>
<td>4</td>
</tr>
<tr>
<td>PSCursorClass</td>
<td>5</td>
</tr>
<tr>
<td>PSConnectionParmsClass</td>
<td>7</td>
</tr>
<tr>
<td>PSCursorClass</td>
<td>9</td>
</tr>
<tr>
<td>PSDataWindowClass</td>
<td>10</td>
</tr>
<tr>
<td>PSDataWindowSourceClass</td>
<td>14</td>
</tr>
<tr>
<td>PSDocumentClass</td>
<td>15</td>
</tr>
<tr>
<td>PSDropDownListClass</td>
<td>16</td>
</tr>
<tr>
<td>PSErrorClass</td>
<td>17</td>
</tr>
<tr>
<td>PSImageClass</td>
<td>18</td>
</tr>
<tr>
<td>PSJaguarConnection</td>
<td>18</td>
</tr>
<tr>
<td>PSLinkClass</td>
<td>20</td>
</tr>
<tr>
<td>PSNamedConnectionParmsClass</td>
<td>21</td>
</tr>
<tr>
<td>psPage</td>
<td>22</td>
</tr>
<tr>
<td>PSPasswordClass</td>
<td>24</td>
</tr>
<tr>
<td>PSRadioGroupClass</td>
<td>25</td>
</tr>
<tr>
<td>PSServerClass</td>
<td>26</td>
</tr>
<tr>
<td>PSSessionClass</td>
<td>27</td>
</tr>
<tr>
<td>PSTextAreaClass</td>
<td>28</td>
</tr>
<tr>
<td>PSTextClass</td>
<td>29</td>
</tr>
<tr>
<td>PSTextClass</td>
<td>29</td>
</tr>
<tr>
<td>PSWebDataWindowClass</td>
<td>30</td>
</tr>
</tbody>
</table>
PSArgClass

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

Methods

<table>
<thead>
<tr>
<th>PSArgClass method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>addArg</td>
<td>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.</td>
</tr>
<tr>
<td>getCharacterEncoding</td>
<td>Returns the name of charset used by the PSArgClass object.</td>
</tr>
<tr>
<td>GetParameterString</td>
<td>Returns the URL encoded value for the string to be passed as a parameter.</td>
</tr>
<tr>
<td>setCharacterEncoding</td>
<td>Sets the name of the charset used by the PSArgClass object.</td>
</tr>
</tbody>
</table>

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.
CHAPTER 1    Web Target Classes and Objects

Properties

<table>
<thead>
<tr>
<th>PSButtonClass property</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>String</td>
<td>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.</td>
</tr>
<tr>
<td>enabled</td>
<td>boolean</td>
<td>Whether or not the control allows focus. This property works only in browsers that support the DISABLED attribute.</td>
</tr>
<tr>
<td>value</td>
<td>String</td>
<td>The label for the button control.</td>
</tr>
<tr>
<td>visible</td>
<td>boolean</td>
<td>Whether or not the client control is generated. If not visible, there is no access to the client control.</td>
</tr>
</tbody>
</table>

Events

<table>
<thead>
<tr>
<th>PSButtonClass event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ServerAction</td>
<td>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.</td>
</tr>
</tbody>
</table>

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.
PSCommandClass

Properties

<table>
<thead>
<tr>
<th>PSCheckBoxClass property</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>String</td>
<td>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.</td>
</tr>
<tr>
<td>enabled</td>
<td>boolean</td>
<td>Whether or not the control can be edited. This property works only in browsers that support the DISABLED attribute.</td>
</tr>
<tr>
<td>value</td>
<td>boolean</td>
<td>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.</td>
</tr>
<tr>
<td>visible</td>
<td>boolean</td>
<td>Whether or not the client control is generated. If not visible, there is no access to the client control.</td>
</tr>
</tbody>
</table>

Events

<table>
<thead>
<tr>
<th>PSCheckBoxClass event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ItemChanged</td>
<td>This event is triggered when the value of the control has changed and passed validation.</td>
</tr>
</tbody>
</table>

PSCommandClass

Provides a mechanism for storing and re-executing a SQL statement.

For JSP targets you must assign a variable of the PSCommandClass type before you can create an instance of the object or call methods on it. To create an object of type PSCommandClass in ASP targets, you can designate an untyped variable to reference an instance of the object that is returned by the CreateCommand method on a PSConnectionClass object.

Syntax

JSP targets

PSCommandClass ( strSql, conn)
CHAPTER 1  Web Target Classes and Objects

**ASP targets**

PSCommandClass ( _ADOCom_ )

### Constructors

<table>
<thead>
<tr>
<th>PSCommandClass constructor</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>strSql</td>
<td>String</td>
<td>SQL statement you want to execute</td>
</tr>
<tr>
<td>conn</td>
<td>PSConnectionClass</td>
<td>Object you use to connect to the database where you want to execute the SQL statement</td>
</tr>
<tr>
<td>ADOCom</td>
<td>ADODB.command object (returned by the PSConnectionClass CreateCommand method)</td>
<td>Object you use to connect to the database where you want to execute the SQL statement</td>
</tr>
</tbody>
</table>

### Methods

<table>
<thead>
<tr>
<th>PSCommandClass method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>Executes a SQL command</td>
</tr>
<tr>
<td>SetSQL</td>
<td>Sets the SQL statement for a command (ASP targets)</td>
</tr>
</tbody>
</table>

**PSConnectionClass**

Allows you to connect to a database, obtain or clear database errors, and create database cursors.

For JSP targets you must assign a variable of the PSConnectionClass type before you can create an instance of the object or call methods on it. To create an object of type PSConnectionClass in ASP targets, you can designate an untyped variable to reference an instance of the object that is returned by the CreateConnection method on the psServer object.

**Syntax**

**Syntax with user name and password for JSP targets**

PSConnectionClass ( pageContext, _Driver, URL, user, password, (bTrace) )
PSConnectionClass

Syntax with database properties for JSP targets

PSConnectionClass ( pageContext, Driver, URL, Properties, 
(bTrace) )

Syntax for ASP targets

PSConnectionClass ( name )

Constructors

<table>
<thead>
<tr>
<th>PSConnectionClass constructor</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pageContext</td>
<td>PageContext (javax.servlet.jsp class)</td>
<td>The implicit pageContext object available to JSP targets.</td>
</tr>
<tr>
<td>Driver</td>
<td>String</td>
<td>The name of the JDBC driver used to connect to the database.</td>
</tr>
<tr>
<td>URL</td>
<td>String</td>
<td>The location of the database to which you want to connect. The database URL is obtained from the database JDBC driver documentation.</td>
</tr>
<tr>
<td>user</td>
<td>String</td>
<td>The user name that the object uses to connect to the specified database.</td>
</tr>
<tr>
<td>password</td>
<td>String</td>
<td>The password that the object uses to connect to the specified database.</td>
</tr>
<tr>
<td>Properties</td>
<td>String</td>
<td>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.</td>
</tr>
<tr>
<td>bTrace (Optional)</td>
<td>boolean</td>
<td>Allows tracing if set to true. The default is false.</td>
</tr>
<tr>
<td>name</td>
<td>String</td>
<td>Name of the connection object.</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>PSConnectionClass method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClearError</td>
<td>Clears the list of error objects</td>
</tr>
<tr>
<td>CreateCommand</td>
<td>Creates a named object that represents a SQL statement</td>
</tr>
<tr>
<td>CreateCursor</td>
<td>Creates a database cursor</td>
</tr>
<tr>
<td>GetError</td>
<td>Returns the first error object</td>
</tr>
</tbody>
</table>
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 for JSP targets, you need to be familiar with the connection parameters for your database system before using this object. The PowerBuilder book Connecting to Your Database provides information about making database connections.

For ASP targets, if you want to set up a database connection to a database that has a named connection, use the PSNamedConnectionParmsClass object.

Syntax

**Syntax with single argument**

PSConnectionParmsClass(connectString)

**Syntax with two arguments (JSP targets only)**

PSConnectionParmsClass(DBProfile, Prop)

**Syntax with multiple arguments**

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

Constructors

<table>
<thead>
<tr>
<th>PSConnectionParmsClass constructor</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>connectString</td>
<td>String</td>
<td>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.</td>
</tr>
<tr>
<td>DBProfile</td>
<td>String</td>
<td>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.</td>
</tr>
<tr>
<td>Prop</td>
<td>Boolean</td>
<td>This value is always “true”. It is used to distinguish the constructor syntax with the DBProfile argument from the syntax with the connectString argument.</td>
</tr>
<tr>
<td>username</td>
<td>String</td>
<td>The user name that the object uses to connect to the specified database.</td>
</tr>
</tbody>
</table>
The following example for an ASP target creates a new object named "connParm". The new object defines a connection to the database using the connect string "mydb" and the user name and password "guest". The connection is made through ODBC.

```powerbuilder
Var connParm = new PSConnectionParmsClass(mydb, guest, guest, ODBC)
```

**Optional arguments in constructor for ASP targets**

Although the last three arguments in the constructor for ASP targets are optional, if you enter a value for the `database` or `serverName` arguments, the optional arguments that can precede the value you enter are no longer optional and must be included in the constructor.

For JSP targets, you must assign a variable as an object of `PSConnectionParmsClass` type before you can instantiate it, and you must use semicolons to terminate each line of code:

```powerbuilder
PSConnectionParmsClass dbConn = new
PSConnectionParmsClass("ConnectString='DSN=EAS Demo DB V4;UID=dba;PWD=sql';ConnectOption='SQL_DRIVER_CONNECT, SQL_DRIVER_NOPROMPT'"");
```
CHAPTER 1  Web Target Classes and Objects

PSCursorClass

Provides access to a SQL result set.

For JSP targets you must assign a variable of the PSCursorClass type before you can create an instance of the object or call methods on it. To create an object of type PSCursorClass in ASP targets, you can designate an untyped variable to reference an instance of the object that is returned by the CreateCursor method on a PSConectionClass object.

**Syntax**

```java
PSCursorClass( ResSet )
```

**Constructors**

<table>
<thead>
<tr>
<th>PSCursorClass constructor</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResSet</td>
<td>ResultSet</td>
<td>Result set object returned by the SQL query</td>
</tr>
</tbody>
</table>

**Methods**

<table>
<thead>
<tr>
<th>PSCursorClass method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EOF</td>
<td>Determines whether the end of a cursor has been reached</td>
</tr>
<tr>
<td>GetColumnCount</td>
<td>Retrieves the number of columns in a cursor</td>
</tr>
<tr>
<td>GetColumn&lt;DataType&gt;</td>
<td>Retrieves the value of a column in a cursor (JSP targets)</td>
</tr>
<tr>
<td>GetColumnLength</td>
<td>Retrieves the length of a column in a cursor (JSP targets)</td>
</tr>
<tr>
<td>GetColumnName</td>
<td>Retrieves the name of a column in a cursor (JSP targets)</td>
</tr>
<tr>
<td>GetColumnType</td>
<td>Retrieves the SQL type of a column in a cursor (JSP targets)</td>
</tr>
<tr>
<td>GetColumnTypeName</td>
<td>Retrieves the database-specific type of a column in a cursor (JSP targets)</td>
</tr>
<tr>
<td>GetPrecision</td>
<td>Retrieves the number of decimal digits of a column in a cursor (JSP targets)</td>
</tr>
<tr>
<td>GetResultSet</td>
<td>Retrieves the result set for a cursor (JSP targets)</td>
</tr>
<tr>
<td>GetResultSetMetaData</td>
<td>Retrieves the metadata result set (JSP targets)</td>
</tr>
<tr>
<td>GetRowCount</td>
<td>Retrieves the number of rows in a cursor</td>
</tr>
<tr>
<td>GetScale</td>
<td>Retrieves the number of digits to right of the decimal point for a column in a cursor (JSP targets)</td>
</tr>
<tr>
<td>GetValue</td>
<td>Retrieves the value of a column in a cursor (ASP targets)</td>
</tr>
</tbody>
</table>
PSDataWindowClass

Creates a new object for a Web DataWindow control. This object lets you add a Web DataWindow object (that you create in DataWindow Designer, 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

**ASP targets**

```psdatawindowclass({objectName}, {ServerSideStateManagement}, {jaguarConnection}, {sourceLocation}, {dbConnection}, {lPageSize})
```

**JSP targets**

```psdatawindowclass(pageContext, request, objectName, ServerSideStateManagement, jaguarConnection, sourceLocation, dbConnection, {lPageSize})
```

**Optional arguments in constructor for ASP targets**

If you use a String value for any of the last three arguments in the constructor for ASP targets, the optional arguments that can precede the value you enter are no longer optional and must be included in the constructor.

**Constructors**

<table>
<thead>
<tr>
<th>PSDataWindowClass constructor</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pageContext</td>
<td>PageContext</td>
<td>Implicit object created by the JSP server to handle page requests.</td>
</tr>
<tr>
<td>request</td>
<td>HttpServletRequest</td>
<td>Object created by servlet container for HTTP requests.</td>
</tr>
</tbody>
</table>
### PSDataWindowClass Constructor

| **objectName** | **String** | The name of the client-side control. An entry for `objectName` is optional for ASP targets. If you do not specify a name, htmlDW is used as the default object. |
| **ServerSideStateManagement** | **boolean** | Specifies where the database state is managed. (An entry for `ServerSideStateManagement` is optional for ASP targets):  
- **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 entry for `jaguarConnection` is optional for ASP targets. If this property is null (default), the object uses an ActiveX server component. |
| **sourceLocation** | **String** | The location of the DataWindow object. An entry for `sourceLocation` is optional for ASP targets. If this property is null (default), the server component must encapsulate the identity of the source. |
| **dbConnection** | **String** | The database connection properties. An entry for `dbConnection` is optional for ASP targets. If this property is null (default), the server component must encapsulate the database connection properties. |
### PSDataWindowClass

**PSDataWindowClass constructor**

<table>
<thead>
<tr>
<th>Name (Optional)</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
</table>
| lPageSize       | 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**

<table>
<thead>
<tr>
<th>Name</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component</td>
<td>Object</td>
<td>Represents a reference to a server component for a Web DataWindow control. A server component is either an ActiveX or EAServer component that interacts with a page server that supports ActiveX or Java. For information about the server component, see the <em>DataWindows Programmer's Guide</em>.</td>
</tr>
<tr>
<td>RetrievalArgs</td>
<td>String</td>
<td>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.</td>
</tr>
</tbody>
</table>

### Methods

**PSDataWindowClass method**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FillRetrievalArgs</td>
<td>Fills in the array that stores the retrieval arguments.</td>
</tr>
</tbody>
</table>
Examples

The following example shows how to define a new Web DataWindow object named htmlDwObj for an ASP target. The Web DataWindow object uses a client-side control named htmlDw1, and a previously defined EAServer connection object named jagParm:

```javascript
Var htmlDwObj = new PSDataWindowClass(htmlDw1, false, jagParm)
```

For JSP targets, you must assign a variable as an object of the PSDataWindowClass type before you can instantiate it, and you must use semicolons to terminate each line of code:

```javascript
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 V4;UID=dba;PWD=sql',ConnectOption='SQL_DRIVER_CONNECT, SQL_DRIVER_NOPROMPT'");
PSDataWindowClass webDW = new PSDataWindowClass(pageContext, request, "webDW", false, jagConn, dwSource, dbConn, 10);
```

### PSDataWindowClass method Description

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generate</td>
<td>Generates the DataWindow as HTML.</td>
</tr>
<tr>
<td>GenerateXHTML</td>
<td>Generates the DataWindow as XHTML.</td>
</tr>
<tr>
<td>GenerateXMLWeb</td>
<td>Generates the DataWindow as XML.</td>
</tr>
<tr>
<td>SetColumnLink</td>
<td>Establishes a link for a column that is passed to the Web DataWindow control.</td>
</tr>
<tr>
<td>SetWeight</td>
<td>Identifies the type of functionality included on your HTML page. (As you include more functionality on your page, the size of the control increases.)</td>
</tr>
</tbody>
</table>
PSDataWindowSourceClass

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

Syntax

PSDataWindowSourceClass(sourceFileName, dwName, stringSourceURL)

Constructors

<table>
<thead>
<tr>
<th>PSDataWindowSourceClass constructor</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sourceFileName</td>
<td>String</td>
<td>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.</td>
</tr>
<tr>
<td>dwName</td>
<td>String</td>
<td>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.</td>
</tr>
<tr>
<td>stringSourceURL</td>
<td>String</td>
<td>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.</td>
</tr>
</tbody>
</table>

Examples

The following JavaScript example (ASP target) creates a new source parameter object named dwParm using the my.pbl library and the DataWindow control dataWin1.

Var dwParm = PSDataWindowSourceClass(my.pbl, dataWin1)
For JSP targets, you must assign a variable as an object of the PSDataWindowSourceClass type before you can instantiate it, and you must use semicolons to terminate each line of code:

```java
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 DBV4;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**

<table>
<thead>
<tr>
<th>PSConnectionClass method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>File</td>
<td>Returns the file name for the document.</td>
</tr>
<tr>
<td>GetEnv</td>
<td>Retrieves the value of a server environment variable.</td>
</tr>
<tr>
<td>GetParam</td>
<td>Retrieves a parameter passed to the current page.</td>
</tr>
<tr>
<td>Path</td>
<td>Returns the path portion of the URL for the document.</td>
</tr>
</tbody>
</table>
**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**

<table>
<thead>
<tr>
<th>PSDropDownListClass property</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>String</td>
<td>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.</td>
</tr>
<tr>
<td>enabled</td>
<td>boolean</td>
<td>Whether or not the control can be edited. This property works only in browsers that support the DISABLED attribute.</td>
</tr>
<tr>
<td>value</td>
<td>String</td>
<td>The label for the drop-down list control.</td>
</tr>
<tr>
<td>visible</td>
<td>boolean</td>
<td>Whether or not the client control is generated. If not visible, there is no access to the client control.</td>
</tr>
</tbody>
</table>
Events

<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ItemChanged</td>
<td>This event is triggered when the value of the control has changed and passed validation.</td>
</tr>
</tbody>
</table>

PSErrorClass

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

For JSP targets you must assign a variable of the PSErrorClass type before you can construct an instance of the object or call methods on it. To create an object of type PSErrorClass in ASP targets, you can designate an untyped variable to reference an instance of the object that is returned by the GetError method on a PSConnectionClass object.

Syntax

```java
PSErrorClass (code, info)
```

Constructors

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>code</td>
<td>int</td>
<td>The error code returned from the connection object</td>
</tr>
<tr>
<td>info</td>
<td>String</td>
<td>The error message returned from the connection object</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getCode</td>
<td>Returns the code associated with the current error object</td>
</tr>
<tr>
<td>getMessage</td>
<td>Returns the message associated with the current error object</td>
</tr>
<tr>
<td>getNextError</td>
<td>Returns the next error object, if one exists</td>
</tr>
</tbody>
</table>
**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**

<table>
<thead>
<tr>
<th>Property</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enabled</td>
<td>boolean</td>
<td>Whether or not the control allows focus. This property works only in browsers that support the DISABLED attribute.</td>
</tr>
<tr>
<td>value</td>
<td>String</td>
<td>The label for the Image Button control.</td>
</tr>
<tr>
<td>visible</td>
<td>boolean</td>
<td>Whether or not the client control is generated. If not visible, there is no access to the client control.</td>
</tr>
</tbody>
</table>

**Events**

<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ServerAction</td>
<td>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.</td>
</tr>
</tbody>
</table>

**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 ActiveX or Java.
Syntax specifying server name and properties

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

Syntax specifying an EAServer profile (JSP targets only)

PSJaguarConnection(profileName, componentName, bOneTrip)

Constructors

<table>
<thead>
<tr>
<th>PSJaguarConnection constructor</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>serverName</td>
<td>String</td>
<td>The name of the server that runs the component for your Web DataWindow control. The syntax for this entry is serverName:port.</td>
</tr>
<tr>
<td>profileName</td>
<td>String</td>
<td>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.</td>
</tr>
<tr>
<td>userId</td>
<td>String</td>
<td>The username that the object uses to connect to the specified EAServer. The default is Jaguar.</td>
</tr>
<tr>
<td>password</td>
<td>String</td>
<td>The password that the object uses to connect to the specified EAServer. The default is guest.</td>
</tr>
<tr>
<td>componentName</td>
<td>String</td>
<td>The name of the Web DataWindow server component on EAServer that uses this connection. The default is DataWindow/nv_html_data_window.</td>
</tr>
</tbody>
</table>
The following example for an ASP target defines a connection to the EAServer named "Jaguar1" using the port 9000:

```javascript
Var jagParm = new PSJaguarConnection(Jaguar1:9000)
```

For JSP targets, you must assign a variable as an object of the PSJaguarConnection type before you can instantiate it, and you must use semicolons to terminate each line of code:

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

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

<table>
<thead>
<tr>
<th>PSLinkClass property</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>String</td>
<td>The destination URL</td>
</tr>
</tbody>
</table>

PSNamedConnectionParmsClass

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

Use the PSConnectionParmsClass object to set up a database connection to a database that does not have a named connection.

You cannot use PSNamedConnectionParmsClass objects with JSP targets.

Syntax

**ASP targets**

```
PSNamedConnectionParmsClass(connectionName)
```

Constructors

<table>
<thead>
<tr>
<th>PSNamedConnectionParmsClass constructor</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>connectionName</td>
<td>String</td>
<td>The name of a named database connection</td>
</tr>
</tbody>
</table>

Platform dependency

How the connection constructor is stored depends on the deployment platform.

Examples

The following example creates the new object, ConnParmNamed, that defines a connection to a database named "mydb":

```
Var connParmNamed = new
    PSNamedConnectionParmsClass(mydb)
```
**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**

<table>
<thead>
<tr>
<th><code>psPage</code> property</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>errors[ ]</td>
<td>Vector</td>
<td>The Vector datatype is a collection of PageError objects. PageError is a value class with 3 string attributes: location, cause, and message.</td>
</tr>
<tr>
<td>showErrorsOnPage</td>
<td>boolean</td>
<td>Specifies whether errors contained in the errors[ ] array are displayed on the page when the page is generated.</td>
</tr>
<tr>
<td>showErrorsAtTop</td>
<td>boolean</td>
<td>Specifies whether errors are displayed at the top or the bottom of the page.</td>
</tr>
<tr>
<td>showErrorsInAlert</td>
<td>boolean</td>
<td>Specifies whether errors are displayed in a client-side alert box after the page has completed loading.</td>
</tr>
<tr>
<td>pageName</td>
<td>String</td>
<td>The current name of the page.</td>
</tr>
<tr>
<td>firstTime</td>
<td>boolean</td>
<td>Indicates whether this is the first time the page was called.</td>
</tr>
<tr>
<td>hadValidationErr</td>
<td>boolean</td>
<td>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.</td>
</tr>
<tr>
<td>didRedirect</td>
<td>boolean</td>
<td>Indicates whether <code>psPage.Redirect</code> has been called. If it has, generation will not occur.</td>
</tr>
<tr>
<td>doTrace</td>
<td>boolean</td>
<td>Indicates whether tracing is enabled. This can be used in the code to check before calling the Trace method multiple times. This property is obsolete. For JSP targets, use the SetTrace and IsTrace methods.</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>psPage event</th>
<th>Occurs</th>
<th>Order of occurrence</th>
</tr>
</thead>
<tbody>
<tr>
<td>RequestStart</td>
<td>At the beginning of a request</td>
<td>1</td>
</tr>
<tr>
<td>FirstTime</td>
<td>The first time a page is loaded</td>
<td>2</td>
</tr>
<tr>
<td>BeforeBinding</td>
<td>Just before binding starts</td>
<td>3</td>
</tr>
<tr>
<td>Validate</td>
<td>To allow whole page validation</td>
<td>4</td>
</tr>
<tr>
<td>Validation Error</td>
<td>If any validate on the page fails</td>
<td>5</td>
</tr>
<tr>
<td>AfterBinding</td>
<td>Just after binding completes</td>
<td>6</td>
</tr>
<tr>
<td>BeforeAction</td>
<td>Just before the action is performed</td>
<td>7</td>
</tr>
<tr>
<td>AfterAction</td>
<td>Just after the action is performed</td>
<td>8</td>
</tr>
<tr>
<td>BeforeGenerate</td>
<td>Just before the page is generated</td>
<td>9</td>
</tr>
<tr>
<td>AfterGenerate</td>
<td>After all generation is complete</td>
<td>10</td>
</tr>
<tr>
<td>RequestFinish</td>
<td>After all generation is complete</td>
<td>11</td>
</tr>
<tr>
<td>ServerError</td>
<td>When ReportError() is called</td>
<td>When invoked</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>psPage method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alert</td>
<td>Causes client-side alert box to display when the page is finished loading</td>
</tr>
<tr>
<td>IsTrace</td>
<td>Indicates whether tracing is enabled</td>
</tr>
<tr>
<td>Redirect</td>
<td>Redirects the client's browser to another page</td>
</tr>
<tr>
<td>ReportError</td>
<td>Indicates whether a server-side error has occurred</td>
</tr>
</tbody>
</table>
**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

<table>
<thead>
<tr>
<th>PSPasswordClass property</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>String</td>
<td>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.</td>
</tr>
</tbody>
</table>
Events

<table>
<thead>
<tr>
<th>PSPasswordClass Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ItemChanged</td>
<td>This event is triggered when the value of the control has changed and passed validation.</td>
</tr>
<tr>
<td>Validate</td>
<td>This event occurs when the client changes a value of a text control. The event is passed the new value.</td>
</tr>
<tr>
<td>ValidationError</td>
<td>This event is triggered when the Validate event fails. It is passed the user-entered value from the Validate event.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>PSRadioGroupClass Property</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enabled</td>
<td>boolean</td>
<td>Whether the control can be edited. This property works only in browsers that support the DISABLED attribute.</td>
</tr>
<tr>
<td>value</td>
<td>String</td>
<td>The value of the selected radio button in the group.</td>
</tr>
<tr>
<td>visible</td>
<td>boolean</td>
<td>Whether the client control is generated. If not visible, there is no access to the client control.</td>
</tr>
</tbody>
</table>
PSServerClass

Events

<table>
<thead>
<tr>
<th>PSRadioGroup Class event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ItemChanged</td>
<td>This event is triggered when the value of a control has changed and passed validation.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>PSServerClass method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreateConnection</td>
<td>Creates a new database connection</td>
</tr>
<tr>
<td>GetConnection</td>
<td>Gets a reference to a connection defined in the Global.asa file (ASP targets)</td>
</tr>
<tr>
<td>MapPath</td>
<td>Maps a relative or virtual path to a physical path on the server (ASP targets)</td>
</tr>
<tr>
<td>ObjectModelType</td>
<td>Identifies the application server</td>
</tr>
<tr>
<td>ObjectModelVersion</td>
<td>Returns the version of the Web Target object model you are using</td>
</tr>
<tr>
<td>Type</td>
<td>Identifies the Web server</td>
</tr>
<tr>
<td>URLEncode</td>
<td>Applies URL encoding rules to a string</td>
</tr>
<tr>
<td>Version</td>
<td>Returns the version of the Web server</td>
</tr>
</tbody>
</table>
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**

<table>
<thead>
<tr>
<th>PSServerClass method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abandon</td>
<td>Causes a session object to be discarded</td>
</tr>
<tr>
<td>GetValue</td>
<td>Retrieves the value of a session variable</td>
</tr>
<tr>
<td>SetValue</td>
<td>Sets the value of a session variable</td>
</tr>
</tbody>
</table>

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**

<table>
<thead>
<tr>
<th>PSStaticTextClass property</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>String</td>
<td>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.</td>
</tr>
<tr>
<td>value</td>
<td>String</td>
<td>The text to be displayed (can be HTML).</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>PSTextAreaClass property</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>String</td>
<td>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.</td>
</tr>
<tr>
<td>enabled</td>
<td>boolean</td>
<td>Whether or not text in the control can be edited. This property works only in browsers that support the DISABLED attribute.</td>
</tr>
<tr>
<td>value</td>
<td>String</td>
<td>The text in the edit control.</td>
</tr>
</tbody>
</table>

Events

<table>
<thead>
<tr>
<th>PSStaticTextClass event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ServerAction</td>
<td>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.</td>
</tr>
</tbody>
</table>
### PSTextAreaClass

<table>
<thead>
<tr>
<th>Property</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>visible</td>
<td>boolean</td>
<td>Whether or not the client control is generated. If not visible, there is no access to the client control.</td>
</tr>
</tbody>
</table>

### Events

<table>
<thead>
<tr>
<th>PSTextAreaClass event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ItemChanged</td>
<td>This event is triggered when the value of the control has changed and passed validation.</td>
</tr>
<tr>
<td>Validate</td>
<td>This event occurs when the client changes a value of a text control. The event is passed the new value.</td>
</tr>
<tr>
<td>ValidationError</td>
<td>This event is triggered when the Validate event fails. It is passed the user-entered value from the Validate event.</td>
</tr>
</tbody>
</table>

### 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.
**PSWebDataWindowClass**

**Properties**

<table>
<thead>
<tr>
<th>PSTextClass property</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>String</td>
<td>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.</td>
</tr>
<tr>
<td>enabled</td>
<td>boolean</td>
<td>Whether or not the text in the control can be edited. This property works only in browsers that support the DISABLED attribute.</td>
</tr>
<tr>
<td>value</td>
<td>String</td>
<td>The text in the edit control.</td>
</tr>
<tr>
<td>visible</td>
<td>boolean</td>
<td>Whether or not the client control is generated. If not visible, there is no access to the client control.</td>
</tr>
</tbody>
</table>

**Events**

<table>
<thead>
<tr>
<th>PSTextClass event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ItemChanged</td>
<td>This event is triggered when the value of the control has changed and passed validation.</td>
</tr>
<tr>
<td>Validate</td>
<td>This event occurs when the client changes a value of a text control. The event is passed the new value.</td>
</tr>
<tr>
<td>ValidationFailure</td>
<td>This event is triggered when the Validate event fails. It is passed the user-entered value from the Validate event.</td>
</tr>
</tbody>
</table>

**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**

```powerbuild
PSWebDataWindowClass(objectName, ServerSideStateManagement, jaguarConnection, sourceLocation, dbConnection, [PageSize])
```
Constructors

<table>
<thead>
<tr>
<th>PSWebDataWindowClass constructor</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>objectName</strong></td>
<td>String</td>
<td>The name of the client-side control. By default, the name is set to dw_1.</td>
</tr>
<tr>
<td><strong>ServerSideStateManagement</strong></td>
<td>boolean</td>
<td>Specifies where the database state is managed:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- 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).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- false (default) The client manages the database state.</td>
</tr>
<tr>
<td><strong>jaguarConnection</strong></td>
<td>String</td>
<td>The connection information needed to connect to EAServer. An EAServer profile must be defined.</td>
</tr>
<tr>
<td><strong>sourceLocation</strong></td>
<td>String</td>
<td>The location of the DataWindow object. If this property is null (default), the server component must encapsulate the identity of the source.</td>
</tr>
<tr>
<td><strong>dbConnection</strong></td>
<td>String</td>
<td>The database connection properties. If this property is null (default), the server component must encapsulate the database connection properties.</td>
</tr>
<tr>
<td><strong>lPageSize (optional)</strong></td>
<td>String</td>
<td>A positive integer specifies the number of rows that will be passed to, and therefore contained in, the Web DataWindow control.</td>
</tr>
</tbody>
</table>

Events

<table>
<thead>
<tr>
<th>PSWebDataWindow Class event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AfterAction</strong></td>
<td>Triggered just after the call to SetAction on the server component</td>
</tr>
<tr>
<td><strong>AfterRetrieve</strong></td>
<td>Triggered just after the call to Retrieve on the server component</td>
</tr>
<tr>
<td><strong>AfterUpdate</strong></td>
<td>Triggered just after the call to Update on the server component</td>
</tr>
</tbody>
</table>
### PSWebDataWindowClass

<table>
<thead>
<tr>
<th>PSWebDataWindowClass event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BeforeAction</td>
<td>Triggered just before SetAction is called on the server component</td>
</tr>
<tr>
<td>BeforeRetrieve</td>
<td>Triggered just before Retrieve is called on the server component</td>
</tr>
<tr>
<td>BeforeUpdate</td>
<td>Triggered just before Update is called on the server component</td>
</tr>
<tr>
<td>OnDBError</td>
<td>Triggered if a database error occurs during processing</td>
</tr>
<tr>
<td>Validate</td>
<td>Triggered immediately after the context is restored in the server component</td>
</tr>
<tr>
<td>Validation Error</td>
<td>Triggered if the webdw.Validate event fails</td>
</tr>
</tbody>
</table>

### Methods

See the DataWindow Reference for more information about these methods:

<table>
<thead>
<tr>
<th>PSWebDataWindowClass method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClearValues</td>
<td>Deletes all items from a value list or code table associated with a DataWindow column</td>
</tr>
<tr>
<td>Create</td>
<td>Creates a DataWindow object using DataWindow source code and puts that object in the specified DataWindow control</td>
</tr>
<tr>
<td>DeletedCount</td>
<td>Reports the number of rows that have been marked for deletion in the database</td>
</tr>
<tr>
<td>DeleteRow</td>
<td>Deletes a row from the DataWindow control</td>
</tr>
<tr>
<td>Describe</td>
<td>Reports the values of properties of a DataWindow object and controls within the DataWindow object</td>
</tr>
<tr>
<td>Filter</td>
<td>Displays rows in a DataWindow that pass the current filter criteria</td>
</tr>
<tr>
<td>FilteredCount</td>
<td>Reports the number of rows that are not displayed in the DataWindow because of the current filter criteria</td>
</tr>
<tr>
<td>Find</td>
<td>Finds the next row in a DataWindow in which data meets a specified condition</td>
</tr>
<tr>
<td>FindGroupChange</td>
<td>Searches for the next break for the specified group</td>
</tr>
<tr>
<td>Generate</td>
<td>Generates the DataWindow as HTML</td>
</tr>
<tr>
<td>GenerateXHTML</td>
<td>Generates the DataWindow as XHTML</td>
</tr>
<tr>
<td>GenerateXMLWeb</td>
<td>Generates the DataWindow as XML</td>
</tr>
<tr>
<td><strong>PSWebDataWindowClass method</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>---------------------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>GetColumn</td>
<td>Obtains the number of the current column</td>
</tr>
<tr>
<td>GetColumnName</td>
<td>Obtains the name of the current column</td>
</tr>
<tr>
<td>GetFormat</td>
<td>Obtains the display format assigned to a column in a DataWindow control</td>
</tr>
<tr>
<td>GetItemDate</td>
<td>Gets data of type Date from the specified buffer of a DataWindow control</td>
</tr>
<tr>
<td>GetItemDateTime</td>
<td>Gets data of type DateTime from the specified buffer of a DataWindow control</td>
</tr>
<tr>
<td>GetItemFormattedString</td>
<td>Gets and formats data of type String from the specified buffer of a DataWindow control or DataStore object.</td>
</tr>
<tr>
<td>GetItemNumber</td>
<td>Gets numeric data from the specified buffer of a DataWindow control</td>
</tr>
<tr>
<td>GetItemStatus</td>
<td>Reports the modification status of a row or a column within a row</td>
</tr>
<tr>
<td>GetItemString</td>
<td>Gets data of type String from the specified buffer of a DataWindow control</td>
</tr>
<tr>
<td>GetItemTime</td>
<td>Gets data of type Time from the specified buffer of a DataWindow control</td>
</tr>
<tr>
<td>GetItemUnformattedString</td>
<td>Gets unformatted data of type String from the specified buffer of a DataWindow control or DataStore object.</td>
</tr>
<tr>
<td>GetRow</td>
<td>Reports the number of the current row in a DataWindow control</td>
</tr>
<tr>
<td>GetValidate</td>
<td>Obtains the validation rule for a column in a DataWindow</td>
</tr>
<tr>
<td>GetValue</td>
<td>Obtains the value of an item in a value list or code table associated with a column in a DataWindow</td>
</tr>
<tr>
<td>GroupCalc</td>
<td>Recalculates the breaks in the grouping levels in a DataWindow</td>
</tr>
<tr>
<td>ImportString</td>
<td>Inserts data into a DataWindow control from tab-delimited data in a string</td>
</tr>
<tr>
<td>InsertRow</td>
<td>Inserts a row in a DataWindow</td>
</tr>
<tr>
<td>ModifiedCount</td>
<td>Reports the number of rows that have been modified but not updated in a DataWindow</td>
</tr>
<tr>
<td>Modify</td>
<td>Modifies a DataWindow object by applying specifications (given as a list of instructions) that change the DataWindow object's definition</td>
</tr>
<tr>
<td>PSWebDataWindowClass method</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>ReselectRow</td>
<td>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</td>
</tr>
<tr>
<td>Reset</td>
<td>Clears all the data from a DataWindow control</td>
</tr>
<tr>
<td>ResetUpdate</td>
<td>Clears the update flags in the primary and filter buffers and empties the delete buffer of a DataWindow</td>
</tr>
<tr>
<td>Retrieve</td>
<td>Retrieves rows from the database for a DataWindow control</td>
</tr>
<tr>
<td>RowCount</td>
<td>Obtains the number of rows that are currently available in a DataWindow control</td>
</tr>
<tr>
<td>RowsDiscard</td>
<td>Discards a range of rows in a DataWindow control</td>
</tr>
<tr>
<td>SaveAs</td>
<td>Saves the contents of a DataWindow in the format you specify</td>
</tr>
<tr>
<td>SetColumn</td>
<td>Sets the current column in a DataWindow control</td>
</tr>
<tr>
<td>SetColumnLink</td>
<td>Specifies information used for constructing hyperlinks for data in a column in generated HTML</td>
</tr>
<tr>
<td>SetDetailHeight</td>
<td>Sets the height of each row in the specified range to the specified value</td>
</tr>
<tr>
<td>SetDWOBJECT</td>
<td>Specifies the DataWindow library and object that the Web DataWindow server component will use for generating HTML</td>
</tr>
<tr>
<td>SetFilter</td>
<td>Specifies filter criteria for a DataWindow control</td>
</tr>
<tr>
<td>SetFormat</td>
<td>Specifies a display format for a column in a DataWindow control</td>
</tr>
<tr>
<td>SetItem</td>
<td>Sets the value of a row and column in a DataWindow control to the specified value</td>
</tr>
<tr>
<td>SetItemDate</td>
<td>Sets the value of a row and column in a DataWindow control to the specified value</td>
</tr>
<tr>
<td>SetItemDateTime</td>
<td>Sets the value of a row and column in a DataWindow control to the specified value</td>
</tr>
<tr>
<td>SetItemNumber</td>
<td>Sets the value of a row and column in a DataWindow control to the specified value</td>
</tr>
<tr>
<td>SetItemStatus</td>
<td>Changes the modification status of a row or a column within a row</td>
</tr>
<tr>
<td>SetItemString</td>
<td>Sets the value of a row and column in a DataWindow control to the specified value</td>
</tr>
<tr>
<td><strong>PSWebDataWindowClass method</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>---------------------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>SetItemTime</td>
<td>Sets the value of a row and column in a DataWindow control to the specified value</td>
</tr>
<tr>
<td>SetPosition</td>
<td>Moves a control within the DataWindow to another band or changes the front-to-back order of controls within a band</td>
</tr>
<tr>
<td>SetRow</td>
<td>Sets the current row in a DataWindow control</td>
</tr>
<tr>
<td>SetServerServiceClasses</td>
<td>Tells the server component to trigger custom events defined in user objects for data validation</td>
</tr>
<tr>
<td>SetSort</td>
<td>Specifies sort criteria for a DataWindow control</td>
</tr>
<tr>
<td>SetSQLSelect</td>
<td>Specifies the SQL SELECT statement for a DataWindow control</td>
</tr>
<tr>
<td>SetValidate</td>
<td>Sets the input validation rule for a column in a DataWindow control</td>
</tr>
<tr>
<td>SetValue</td>
<td>Sets the value of an item in a value list or code table for a column in a DataWindow control</td>
</tr>
<tr>
<td>SetWeight</td>
<td>Specifies the types of JavaScript code that will be included in the generated HTML</td>
</tr>
<tr>
<td>Sort</td>
<td>Sorts the rows in a DataWindow control using the DataWindow’s current sort criteria</td>
</tr>
<tr>
<td>Update</td>
<td>Updates the database with the changes made in a DataWindow control</td>
</tr>
</tbody>
</table>
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. 4GL events are not available for ASP targets.

Contents

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>AfterAction</td>
<td>38</td>
</tr>
<tr>
<td>AfterBinding</td>
<td>39</td>
</tr>
<tr>
<td>AfterGenerate</td>
<td>40</td>
</tr>
<tr>
<td>BeforeAction</td>
<td>41</td>
</tr>
<tr>
<td>BeforeBinding</td>
<td>42</td>
</tr>
<tr>
<td>BeforeGenerate</td>
<td>43</td>
</tr>
<tr>
<td>FirstTime</td>
<td>44</td>
</tr>
<tr>
<td>ItemChanged</td>
<td>45</td>
</tr>
<tr>
<td>RequestFinish</td>
<td>46</td>
</tr>
<tr>
<td>RequestStart</td>
<td>46</td>
</tr>
<tr>
<td>ServerAction</td>
<td>47</td>
</tr>
<tr>
<td>ServerError</td>
<td>48</td>
</tr>
<tr>
<td>Validate</td>
<td>50</td>
</tr>
<tr>
<td>Validation Error</td>
<td>52</td>
</tr>
</tbody>
</table>
### AfterAction

<table>
<thead>
<tr>
<th>Description</th>
<th>Occurs after all actions have been performed but before generation of the HTML page.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Applies to</td>
<td>psPage object</td>
</tr>
<tr>
<td>Arguments</td>
<td>None</td>
</tr>
<tr>
<td>Return codes</td>
<td>Boolean</td>
</tr>
<tr>
<td>Usage</td>
<td>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.</td>
</tr>
</tbody>
</table>

**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:

```powershell
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.

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

**See also**
BeforeAction
BeforeBinding
# AfterGenerate

<table>
<thead>
<tr>
<th>Description</th>
<th>Occurs after all generation has taken place.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Applies to</td>
<td>psPage object</td>
</tr>
<tr>
<td>Arguments</td>
<td>None</td>
</tr>
<tr>
<td>Return codes</td>
<td>None</td>
</tr>
<tr>
<td>Usage</td>
<td>This event does not occur if a Redirect method is called. If this event occurs, it is followed by the RequestFinish event.</td>
</tr>
<tr>
<td>Examples</td>
<td>This example turns off tracing for all events after page generation.</td>
</tr>
<tr>
<td></td>
<td>psPage.SetTrace (false);</td>
</tr>
<tr>
<td>See also</td>
<td>RequestFinish</td>
</tr>
</tbody>
</table>
**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 for a JSP target displays a client-side alert box message if the `Redirect` method is not called for another `psPage` event:

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

**See also**
AfterBinding
BeforeAction for PSWebDataWindowClass objects
### BeforeBinding

<table>
<thead>
<tr>
<th>Description</th>
<th>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.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Applies to</td>
<td>psPage object</td>
</tr>
<tr>
<td>Arguments</td>
<td>None</td>
</tr>
<tr>
<td>Return codes</td>
<td>None</td>
</tr>
<tr>
<td>Usage</td>
<td>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.</td>
</tr>
</tbody>
</table>

**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:

```powershell
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 for a JSP target displays a client-side alert box message if the Redirect method is not called for another psPage event:

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

See also
AfterGenerate
FirstTime

**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:

```java
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:

```java
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:

```javascript
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, PDDropDownListClass, 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:

```javascript
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

46
PowerBuilder
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, PSGlobalClass, 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:

```javascript
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**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>location</code></td>
<td>String for <code>objectName.methodName</code> passed to <code>ReportError</code></td>
</tr>
<tr>
<td><code>cause</code></td>
<td>String for the error cause</td>
</tr>
<tr>
<td><code>message</code></td>
<td>String for the system error message</td>
</tr>
</tbody>
</table>

**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:

<table>
<thead>
<tr>
<th>Cause</th>
<th>Object (method)</th>
<th>Meaning</th>
<th>Message string</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component call failed</td>
<td>Various (TestCompError)</td>
<td>An EAServer method call failed</td>
<td>Returned message</td>
</tr>
<tr>
<td>Could not create</td>
<td>PSfaguarObjectClass (CreateComponent)</td>
<td><code>java.createComponent</code> failed for declarative EAServer object</td>
<td>Returned message</td>
</tr>
<tr>
<td>context failed</td>
<td>PSWebDataWindow Class(BindToInput)</td>
<td>Restoring the context through <code>SetAction</code> call failed</td>
<td>Returned code</td>
</tr>
<tr>
<td>Create failed</td>
<td>PSWebDataWindow Class(loadDWOObject)</td>
<td>The source for a DataWindow definition failed to compile</td>
<td>Compile error</td>
</tr>
<tr>
<td>Source not found</td>
<td>PSWebDataWindow Class(loadDWOObject)</td>
<td>The source for a DataWindow definition could not be found</td>
<td>Specified URL</td>
</tr>
<tr>
<td><code>SetDWOObject</code> failed</td>
<td>PSWebDataWindow Class(loadDWOObject)</td>
<td>The call to <code>SetDWOObject</code> failed</td>
<td>Error return code, specified PBL, and DW name</td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>Object</th>
<th>See</th>
</tr>
</thead>
<tbody>
<tr>
<td>psPage</td>
<td>Syntax 1</td>
</tr>
<tr>
<td>PSPasswordClass, PSTextAreaClass, PSTextClass</td>
<td>Syntax 2</td>
</tr>
</tbody>
</table>

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:

```java
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:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sNewValue</td>
<td>String variable for the new value the user enters</td>
</tr>
</tbody>
</table>
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 EASe\textregistered{}r component to do the validation). Simple validation should preferably be written in client-side JavaScript.

Examples

This example calls the validate method on an EASe\textregistered{}r component called “n\_creditcard”. The method (and event) returns true if the credit card is valid.

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

See also

`ValidationError`
Validation Error

The ValidationError event has different arguments for different objects:

<table>
<thead>
<tr>
<th>Object</th>
<th>See</th>
</tr>
</thead>
<tbody>
<tr>
<td>psPage</td>
<td>Syntax 1</td>
</tr>
<tr>
<td>PSPasswordClass, PSTextAreaClass, PSTextClass</td>
<td>Syntax 2</td>
</tr>
</tbody>
</table>

For use with PSWebDataWindowClass objects, see Validation Error 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 Validation Error 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:

```powershell
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

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sNewValue</td>
<td>String variable for the user-entered value that fails the control.Validate test</td>
</tr>
</tbody>
</table>

Return codes

None

Usage

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

See also

Validate
ValidationError
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. 4GL events are not available for ASP targets.

Contents

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Using server-side events with the Web DataWindow DTC</td>
<td>55</td>
</tr>
<tr>
<td>AfterAction</td>
<td>56</td>
</tr>
<tr>
<td>AfterRetrieve</td>
<td>56</td>
</tr>
<tr>
<td>AfterUpdate</td>
<td>57</td>
</tr>
<tr>
<td>BeforeAction</td>
<td>57</td>
</tr>
<tr>
<td>BeforeRetrieve</td>
<td>58</td>
</tr>
<tr>
<td>BeforeUpdate</td>
<td>58</td>
</tr>
<tr>
<td>OnDBError</td>
<td>59</td>
</tr>
<tr>
<td>Validate</td>
<td>60</td>
</tr>
<tr>
<td>Validation Error</td>
<td>61</td>
</tr>
</tbody>
</table>

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 Web and 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

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>numRows</td>
<td>Number of rows retrieved by the server component</td>
</tr>
</tbody>
</table>

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
CHAPTER 3   Web DataWindow Events

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:

```powershell
    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**

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

**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:

```java
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

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

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

See also
- Validate for other Web Target object model objects
- `ValidationError`
**Validation Error**

**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.

```javascript
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
`Validation Error` for other Web Target object model objects
ValidationError
CHAPTER 4

Web Target Methods

About this chapter

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

Contents

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abandon</td>
<td>65</td>
</tr>
<tr>
<td>addArg</td>
<td>66</td>
</tr>
<tr>
<td>Alert</td>
<td>67</td>
</tr>
<tr>
<td>ClearError</td>
<td>68</td>
</tr>
<tr>
<td>CreateCommand</td>
<td>69</td>
</tr>
<tr>
<td>CreateConnection</td>
<td>70</td>
</tr>
<tr>
<td>CreateCursor</td>
<td>72</td>
</tr>
<tr>
<td>EOF</td>
<td>74</td>
</tr>
<tr>
<td>Execute</td>
<td>75</td>
</tr>
<tr>
<td>File</td>
<td>76</td>
</tr>
<tr>
<td>FillRetrievalArgs</td>
<td>77</td>
</tr>
<tr>
<td>Generate</td>
<td>78</td>
</tr>
<tr>
<td>GenerateXHTML</td>
<td>79</td>
</tr>
<tr>
<td>GenerateXMLWeb</td>
<td>80</td>
</tr>
<tr>
<td>getCharacterEncoding</td>
<td>82</td>
</tr>
<tr>
<td>getCode</td>
<td>83</td>
</tr>
<tr>
<td>GetColumnCount</td>
<td>84</td>
</tr>
<tr>
<td>GetColumn&lt;DataType&gt;</td>
<td>85</td>
</tr>
<tr>
<td>GetColumnLength</td>
<td>86</td>
</tr>
<tr>
<td>GetColumnName</td>
<td>87</td>
</tr>
<tr>
<td>GetColumnType</td>
<td>88</td>
</tr>
<tr>
<td>GetColumnTypeName</td>
<td>89</td>
</tr>
<tr>
<td>GetConnection</td>
<td>90</td>
</tr>
<tr>
<td>GetEnv</td>
<td>91</td>
</tr>
<tr>
<td>GetError</td>
<td>93</td>
</tr>
<tr>
<td>GetMessage</td>
<td>95</td>
</tr>
<tr>
<td>getNextError</td>
<td>96</td>
</tr>
<tr>
<td>GetParam</td>
<td>97</td>
</tr>
<tr>
<td>Topic</td>
<td>Page</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>GetParameterString</td>
<td>98</td>
</tr>
<tr>
<td>GetPrecision</td>
<td>99</td>
</tr>
<tr>
<td>GetResultSet</td>
<td>100</td>
</tr>
<tr>
<td>GetResultSetMetaData</td>
<td>102</td>
</tr>
<tr>
<td>GetRowCount</td>
<td>103</td>
</tr>
<tr>
<td>GetScale</td>
<td>104</td>
</tr>
<tr>
<td>GetValue</td>
<td>105</td>
</tr>
<tr>
<td>IsTrace</td>
<td>107</td>
</tr>
<tr>
<td>MapPath</td>
<td>108</td>
</tr>
<tr>
<td>Move</td>
<td>109</td>
</tr>
<tr>
<td>MoveFirst</td>
<td>110</td>
</tr>
<tr>
<td>MoveLast</td>
<td>111</td>
</tr>
<tr>
<td>MoveNext</td>
<td>112</td>
</tr>
<tr>
<td>MovePrevious</td>
<td>113</td>
</tr>
<tr>
<td>ObjectModelType</td>
<td>114</td>
</tr>
<tr>
<td>ObjectModelVersion</td>
<td>115</td>
</tr>
<tr>
<td>Path</td>
<td>116</td>
</tr>
<tr>
<td>Redirect</td>
<td>117</td>
</tr>
<tr>
<td>ReportError</td>
<td>119</td>
</tr>
<tr>
<td>setCharacterEncoding</td>
<td>120</td>
</tr>
<tr>
<td>SetColumnLink</td>
<td>121</td>
</tr>
<tr>
<td>SetSQL</td>
<td>122</td>
</tr>
<tr>
<td>SetTrace</td>
<td>123</td>
</tr>
<tr>
<td>SetValue</td>
<td>124</td>
</tr>
<tr>
<td>SetWeight</td>
<td>125</td>
</tr>
<tr>
<td>Site</td>
<td>126</td>
</tr>
<tr>
<td>TestCompError</td>
<td>127</td>
</tr>
<tr>
<td>Trace</td>
<td>128</td>
</tr>
<tr>
<td>TraceIndent</td>
<td>129</td>
</tr>
<tr>
<td>TraceOutdent</td>
<td>130</td>
</tr>
<tr>
<td>Type</td>
<td>131</td>
</tr>
<tr>
<td>URLEncode</td>
<td>132</td>
</tr>
<tr>
<td>Version</td>
<td>133</td>
</tr>
<tr>
<td>Write</td>
<td>134</td>
</tr>
<tr>
<td>WriteErrorsToDocument</td>
<td>135</td>
</tr>
<tr>
<td>WriteLn</td>
<td>136</td>
</tr>
</tbody>
</table>
Abandon

Description
Causes a session object to be discarded.

Applies to
PSSessionClass object

Syntax
psSession.Abandon();

Return value
None

Usage
At runtime, Abandon has the following behavior:

<table>
<thead>
<tr>
<th>Application server</th>
<th>Runtime behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASP</td>
<td>Calls the Abandon method of the Session object</td>
</tr>
<tr>
<td>JSP</td>
<td>Destroys the current session object immediately</td>
</tr>
</tbody>
</table>

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 JSP 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 )
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ArgObj</td>
<td>Object of the PSArgClass type to which you add name-value pairs</td>
</tr>
<tr>
<td>argName</td>
<td>Name of the argument you want to add</td>
</tr>
<tr>
<td>Value</td>
<td>Value of the argument you want to add</td>
</tr>
</tbody>
</table>

**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 JSP 4GL targets only.

Applies to
psPage object

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

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>message</td>
<td>Message to be displayed in the client-side alert box</td>
</tr>
<tr>
<td>appendToCurrent</td>
<td>(optional) Indicates whether the passed message should replace any current message (the default) or be appended to the current message</td>
</tr>
</tbody>
</table>

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:

```java
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 has the following behavior:

<table>
<thead>
<tr>
<th>Application server</th>
<th>Runtime behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASP</td>
<td>Calls the Clear method of the Errors collection to remove all of the errors in the connection object's error object list</td>
</tr>
<tr>
<td>JSP</td>
<td>Destroys the current error object by setting it to null</td>
</tr>
</tbody>
</table>

**Examples**
The following example clears the list of errors for the myconnect connection object:

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

For a JSP target, you must declare “myconnect” as a variable of type PSConnectionClass before instantiating the connection object (in a psServer.CreateConnection call) and calling its ClearError method.

**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 has the following behavior:

<table>
<thead>
<tr>
<th>Application server</th>
<th>Runtime behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASP</td>
<td>Calls the CreateObject method of the Server object to create an ADODB.Command object</td>
</tr>
<tr>
<td>JSP</td>
<td>Returns a command object that can be used to execute the SQL statement that you pass as an argument</td>
</tr>
</tbody>
</table>

Examples
The following example creates a SQL command object and executes the SQL statement associated with the command:

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

For a JSP target, you must declare “mycommand” as a variable of type PSCommandClass before instantiating it and calling any methods on it.
CreateConnection

Description
Creates a new database connection.

Applies to
PSServerClass object

Syntax

**ASP targets**
psServer.CreateConnection( connectionstring, user, password )

**JSP targets ( syntax with user name and password )**
psServer.CreateConnection ( pageContext, Driver, URL, user, password, {bTrace} )

**JSP targets ( syntax with database properties )**
psServer.CreateConnection ( pageContext, Driver, URL, Properties, {bTrace} )

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>connectionstring</td>
<td>A string that specifies connection parameters (for example, the name of the data source to which you want to connect).</td>
</tr>
<tr>
<td>user</td>
<td>The user name for the connection.</td>
</tr>
<tr>
<td>password</td>
<td>The password for the specified user name.</td>
</tr>
<tr>
<td>pageContext</td>
<td>The implicit pageContext object available to JSP targets.</td>
</tr>
<tr>
<td>Driver</td>
<td>The connection mechanism used to connect to the database.</td>
</tr>
<tr>
<td>URL</td>
<td>The location of the database to which you want to connect. The database URL is obtained from the database JDBC driver documentation.</td>
</tr>
<tr>
<td>Properties</td>
<td>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.</td>
</tr>
<tr>
<td>bTrace (Optional)</td>
<td>Allows tracing if set to true. The default is false.</td>
</tr>
</tbody>
</table>

Return value
PSConnectionClass object

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

At runtime, CreateConnection has the following behavior:

<table>
<thead>
<tr>
<th>Application server</th>
<th>Runtime behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASP</td>
<td>Creates an ADODB.Connection object</td>
</tr>
<tr>
<td>JSP</td>
<td>Creates a JDBC connection object</td>
</tr>
</tbody>
</table>
Examples

The following example creates a new connection for an ASP target and stores the object reference in the `myconnect` variable. The connection specifies “SalesDB” as the data source. The user ID is “DBA” and the password is “SQL”:

```csharp
myconnect = psServer.CreateConnection("DSN=SalesDB", "DBA", "SQL");
```

The following example creates a new connection for a JSP target and turns on tracing:

```csharp
PSConnectionClass myConnect;
    "jdbc:sybase:Tds:localhost:2638", "dba", "sql", true);
```
CreateCursor

Description
Creates a database cursor.

Applies to
PSConnectionClass objects

Syntax
**ASP and JSP targets**

\[
\text{connectionobject.CreateCursor( sql )}
\]

**JSP targets only**

\[
\text{connectionobject.CreateCursor( sql, resultSetType, resultSetConcurrency )}
\]

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>connectionobject</code></td>
<td>A variable that contains a reference to an instance of PSConnectionClass.</td>
</tr>
<tr>
<td><code>sql</code></td>
<td>A string that specifies the SQL statement.</td>
</tr>
<tr>
<td><code>resultSetType</code></td>
<td>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.</td>
</tr>
<tr>
<td><code>resultSetConcurrency</code></td>
<td>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.</td>
</tr>
</tbody>
</table>

Return value
PSCursorClass object

Usage
At runtime, CreateCursor has the following behavior:

<table>
<thead>
<tr>
<th>Application server</th>
<th>Runtime behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASP</td>
<td>Calls the Execute method of the Connection object</td>
</tr>
<tr>
<td>JSP</td>
<td>Creates a cursor object and executes the SQL statement</td>
</tr>
</tbody>
</table>

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:

```java
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++)
    {
```
CHAPTER 4    Web Target Methods

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 );
}

To use the same example in a JSP target, you must declare “mycursor” as a variable of type PSCursorClass before instantiating it and calling any methods on it. You must also declare “myquery” as a variable of type String, “myconnect” as a variable of type PSConnectionClass, “dberror” as a variable of type boolean, and “errobj” as a variable of type PSErrorClass.
**EOF**

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

**Applies to**
PSCursorClass objects

**Syntax**
`cursorobject.EOF()`

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cursorobject</code></td>
<td>A variable that contains a reference to an instance of PSCursorClass</td>
</tr>
</tbody>
</table>

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

**Usage**
At runtime, EOF has the following behavior:

<table>
<thead>
<tr>
<th>Application server</th>
<th>Runtime behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASP</td>
<td>Accesses the EOF property of the Recordset object</td>
</tr>
<tr>
<td>JSP</td>
<td>Calls the <code>isAfterLast</code> method on the ResultSet object to determine if the end of the cursor has been reached</td>
</tr>
</tbody>
</table>

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

```java
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;
```

To use the same example in a JSP target, you must declare “mycursor” as a variable of type PSCursorClass before instantiating it and calling any methods on it. You must also declare “myquery” as a variable of type String, “myconnect” as a variable of type PSConnectionClass, and “dberror” as a variable of type boolean.
CHAPTER 4  Web Target Methods

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 has the following behavior:

<table>
<thead>
<tr>
<th>Application server</th>
<th>Runtime behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASP</td>
<td>Creates a new RecordSet object using the SQL statement stored with the PSCommandClass object</td>
</tr>
<tr>
<td>JSP</td>
<td>Creates a new ResultSet object using the SQL statement stored with the PSCommandClass object</td>
</tr>
</tbody>
</table>

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

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

To use the same example in a JSP target, you must declare “mycommand” as a variable of type PSCommandClass before instantiating it and calling any methods on it.
### File

<table>
<thead>
<tr>
<th>Description</th>
<th>Returns the file name and extension for the current document.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Applies to</td>
<td>PSDocumentClass object</td>
</tr>
<tr>
<td>Syntax</td>
<td>psDocument.File( )</td>
</tr>
<tr>
<td>Return value</td>
<td>String</td>
</tr>
<tr>
<td>Usage</td>
<td>At runtime, File extracts the file name and extension from the PATH_INFO server environment variable.</td>
</tr>
<tr>
<td>Examples</td>
<td>The following example returns the file name of the current document into a variable called myfile. If the URL for the current document were <a href="http://www.sybase.com/index.htm">http://www.sybase.com/index.htm</a>, the value of myfile would be index.htm after the call to File:</td>
</tr>
<tr>
<td></td>
<td>myfile = psDocument.File( );</td>
</tr>
</tbody>
</table>

```
### FillRetrievalArgs

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

**Applies to**
PSDataWindowClass object

**Syntax**
```java
DWOBJ ect.FillRetrievalArgs(variable[ ] arguments)
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>DWOBJ ect</code></td>
<td>A variable that contains a reference to an instance of PSDataWindowClass.</td>
</tr>
<tr>
<td><code>variable</code></td>
<td>An array for variables to use as retrieval arguments. For JSP targets, the variables must be defined as String datatypes.</td>
</tr>
</tbody>
</table>

**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.

The behavior is the same for ASP and JSP targets.

**Examples**
In the following example, the retrieval argument array is filled with “MyParam1” and “MyParam2” for the DataWindow object named “htmlDwObj1”:

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

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

```
DWOObject.Generate(page[ ] variables)
```

Argument | Description
--- | ---
DWOObject | A variable that contains a reference to an instance of PSDataWindowClass.
page | An array for passing page variables. For JSP targets, the page variables must be defined as String datatypes.

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.

The behavior is the same for ASP and JSP targets.

Examples
In the following example, the DataWindow object htmlDwObj1 generates HTML and maintains the page variables called “MyParam1” and “MyParam2”:

```powershell
htmlDwObj1.Generate("MyParam1", "MyParam2");
```
CHAPTER 4  Web Target Methods

GenerateXHTML

Description  Generates the inline content for the XHTML Web DataWindow. This method is for use with JSP targets only.

Applies to  PSDataWindowClass and PSWebDataWindowClass objects

Syntax  

```
  DWObject.GenerateXHTML({
    page: [ ] variables
  })
```

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:

<table>
<thead>
<tr>
<th>HTML Web DataWindow use</th>
<th>XHTML Web DataWindow use</th>
<th>XML Web DataWindow use</th>
</tr>
</thead>
<tbody>
<tr>
<td>When the HTMLGen.PageSize property is not assigned a value (row count per page changes)</td>
<td>When you want more industry-standard Web pages than HTML can provide and the ability to customize pages using an XHTML export template</td>
<td>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</td>
</tr>
<tr>
<td>Small amounts of data</td>
<td>Small amounts of data</td>
<td>Large amounts of paged data</td>
</tr>
</tbody>
</table>

Examples  In the following example, 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. This method is for use with JSP targets only.

### Applies to
PSDataWindowClass and PSWebDataWindowClass objects

### Syntax

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

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DWObject</td>
<td>A variable that contains a reference to an instance of PSDataWindowClass or PSWebDataWindowClass.</td>
</tr>
<tr>
<td>page</td>
<td>An array for passing page variables for objects of type PSDataWindowClass. The page variables must be defined as String datatypes.</td>
</tr>
</tbody>
</table>

### 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:

```java
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'"");
```
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:

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

See also

Generate
GenerateXHTML
getCharacterEncoding

**Description**
Returns the charset encoding for the PSArgClass object. This method is for use with JSP 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();
sl_e_1.value = charSet;
```

**See also**

addArg
setCharacterEncoding
GetCode

Description
Returns the code associated with the current error object.

Applies to
PSErrorClass objects

Syntax
errorobjectUserCode()

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 has the following behavior:

<table>
<thead>
<tr>
<th>Application server</th>
<th>Runtime behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASP</td>
<td>Returns the error code, which was obtained by accessing the Number property of the Error object</td>
</tr>
<tr>
<td>JSP</td>
<td>Returns the error code stored in an error object</td>
</tr>
</tbody>
</table>

Examples
The following example connects to a database and retrieves some data. If an error occurs, it uses GetCode to retrieve the code for the error:

```java
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 = errobjUserCode() + " " +
        errobj.GetMessage();
  psDocument.Write ( str );
}
```
To use the same example in a JSP target, you must declare “mycommand” as a variable of type PSCCommandClass before instantiating it and calling any methods on it. You must also declare “mycursor” as a variable of type PSCursorClass, “myquery” as a variable of type String, and “dberror” as a variable of type boolean.

**GetColumnCount**

Description: Retrieves the number of columns in a cursor.

Applies to: PSCursorClass objects

Syntax: `cursorobject.GetColumnCount()`

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cursorobject</code></td>
<td>A variable that contains a reference to an instance of PSCursorClass</td>
</tr>
</tbody>
</table>

Return value: Number. Returns an int for JSP Web targets.

Usage: At runtime, GetColumnCount has the following behavior:

<table>
<thead>
<tr>
<th>Application server</th>
<th>Runtime behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASP</td>
<td>Gets the value of the Count property of the Fields collection</td>
</tr>
<tr>
<td>JSP</td>
<td>Calls the getColumnCount method on the ResultSetMetaData object</td>
</tr>
</tbody>
</table>

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

```java
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. These methods can be used in JSP Web targets only.

Applies to
PSCursorClass objects

Syntax

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cursorobject.GetColumnBoolean ( String strColName )</td>
<td>A variable that contains a reference to an instance of PSCursorClass</td>
</tr>
<tr>
<td>cursorobject.GetColumnBoolean ( int iCol )</td>
<td>Name of the column for which you want to obtain the value</td>
</tr>
<tr>
<td>cursorobject.GetColumnByte ( String strColName )</td>
<td>Number of the column for which you want to obtain the value</td>
</tr>
<tr>
<td>cursorobject.GetColumnByte ( int iCol )</td>
<td></td>
</tr>
<tr>
<td>cursorobject.GetColumnDouble ( String strColName )</td>
<td></td>
</tr>
<tr>
<td>cursorobject.GetColumnDouble ( int iCol )</td>
<td></td>
</tr>
<tr>
<td>cursorobject.GetColumnFloat ( String strColName )</td>
<td></td>
</tr>
<tr>
<td>cursorobject.GetColumnFloat ( int iCol )</td>
<td></td>
</tr>
<tr>
<td>cursorobject.GetColumnInt ( String strColName )</td>
<td></td>
</tr>
<tr>
<td>cursorobject.GetColumnInt ( int iCol )</td>
<td></td>
</tr>
<tr>
<td>cursorobject.GetColumnLong ( String strColName )</td>
<td></td>
</tr>
<tr>
<td>cursorobject.GetColumnLong ( int iCol )</td>
<td></td>
</tr>
<tr>
<td>cursorobject.GetColumnShort ( String strColName )</td>
<td></td>
</tr>
<tr>
<td>cursorobject.GetColumnShort ( int iCol )</td>
<td></td>
</tr>
<tr>
<td>cursorobject.GetColumnString ( String strColName )</td>
<td></td>
</tr>
<tr>
<td>cursorobject.GetColumnString ( int iCol )</td>
<td></td>
</tr>
</tbody>
</table>

Return value
Corresponds to the datatype in the method name.

Usage
Use in conjunction with the GetColumnType method to obtain values in a cursor. For JSP targets, 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. This method is for use in JSP targets only.

Applies to
PSCursorClass objects

Syntax

```csharp
cursorobject.GetColumnLength ( String strColName )
cursorobject.GetColumnLength ( int iCol )
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cursorobject</code></td>
<td>A variable that contains a reference to an instance of PSCursorClass</td>
</tr>
<tr>
<td><code>strColName</code></td>
<td>Name of the column for which you want to obtain the length</td>
</tr>
<tr>
<td><code>iCol</code></td>
<td>Number of the column for which you want to obtain the length</td>
</tr>
</tbody>
</table>

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 PSCursorClass.

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

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

Description: Returns the column name in a cursor for a JSP Web target.

Applies to: PSCursorClass objects

Syntax: `cursorobject.GetColumnName( int iCol )`

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cursorobject</code></td>
<td>A variable that contains a reference to an instance of PSCursorClass</td>
</tr>
<tr>
<td><code>iCol</code></td>
<td>Number of the column for which you want to obtain the name</td>
</tr>
</tbody>
</table>

Return value: String

Usage: Use in conjunction with `GetColumn<DataType>` methods to obtain values in a cursor. For JSP targets, 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 for a JSP Web target.

Applies to
PSCursorClass objects

Syntax
cursorobject.GetColumnType ( int iCol )

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cursorobject</td>
<td>A variable that contains a reference to an instance of PSCursorClass</td>
</tr>
<tr>
<td>iCol</td>
<td>Number of the column for which you want to obtain the SQL type</td>
</tr>
</tbody>
</table>

Return value
int

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. In JSP targets 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 for a JSP Web target.

Applies to
PSCursorClass objects

Syntax
`cursorobject.GetColumnTypeName ( int iCol )`

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cursorobject</code></td>
<td>A variable that contains a reference to an instance of PSCursorClass</td>
</tr>
<tr>
<td><code>iCol</code></td>
<td>Number of the column for which you want to obtain the (database-specific) datatype name</td>
</tr>
</tbody>
</table>

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. This method is not implemented for JSP targets.

Applies to
PSServerClass object

Syntax
psServer.GetConnection( name )

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name of the connection</td>
</tr>
</tbody>
</table>

Return value
PSConnectionClass object

Usage
In a server script that uses the Web Target object model, you can use a database connection that has been predefined in the Database Profiles painter in PowerBuilder. In the Web Target user interface, you can include connections for deployment to your Web server by selecting them on the Database tab of the Web target properties dialog box. To use the same connections for a Web DataWindow, you must make sure that the connections are also defined for the server hosting your Web DataWindow generator component.

In an ASP target at runtime, GetConnection opens a connection defined in the Global.asa file.

Examples
The following example gets a reference to the connection called “SalesDBConnection” and stores the object reference in the myconnect variable. If GetConnection is successful, it uses the connection object to create a cursor:

```plaintext
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;
```
**GetEnv**

**Description**
Retrieves the value of a server environment variable.

**Applies to**
PSDocumentClass object

**Syntax**
psDocument.GetEnv( envvar )

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>envvar</td>
<td>The name of a server environment variable</td>
</tr>
</tbody>
</table>

**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:

<table>
<thead>
<tr>
<th>Server environment variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUTH_TYPE</td>
<td>The authentication method that the server uses to validate users who try to access protected scripts.</td>
</tr>
<tr>
<td>CONTENT_LENGTH</td>
<td>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).</td>
</tr>
<tr>
<td>CONTENT_TYPE</td>
<td>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).</td>
</tr>
<tr>
<td>GATEWAY_INTERFACE</td>
<td>The version of the CGI interface specification being used by the server.</td>
</tr>
<tr>
<td>HTTP_headername</td>
<td>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.</td>
</tr>
<tr>
<td>PATH_INFO</td>
<td>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.</td>
</tr>
<tr>
<td>PATH_TRANSLATED</td>
<td>The value of PATH_INFO along with the virtual path converted to a physical directory name.</td>
</tr>
<tr>
<td>QUERY_STRING</td>
<td>The query information stored in the string following the question mark (?) in the HTTP request.</td>
</tr>
<tr>
<td>REMOTE_ADDR</td>
<td>The IP address of the client making the request.</td>
</tr>
<tr>
<td>REMOTE_HOST</td>
<td>The domain name of the client making the request.</td>
</tr>
</tbody>
</table>
For a complete list of the server environment variables supported on your server platform, see your server documentation.

At runtime, GetEnv has the following behavior:

<table>
<thead>
<tr>
<th>Application server</th>
<th>Runtime behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASP</td>
<td>Accesses the ServerVariables collection of the Request object</td>
</tr>
<tr>
<td>JSP</td>
<td>Calls the mapped method in the ENVPARAM class</td>
</tr>
</tbody>
</table>

Examples

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

```powershell
remaddr = psDocument.GetEnv("REMOTE_ADDR");
```
**GetError**

**Description**
Returns the first error object.

**Applies to**
PSConnectionClass objects

**Syntax**
`connectionobject.GetError( )`

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>connectionobject</code></td>
<td>A variable that contains a reference to an instance of PSConnectionClass</td>
</tr>
</tbody>
</table>

**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 has the following behavior:

<table>
<thead>
<tr>
<th>Application server</th>
<th>Runtime behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASP</td>
<td>If the Count property of the Errors collection indicates that any errors were caused by the last database operation, creates a new instance of PSErrorClass and returns the object instance</td>
</tr>
<tr>
<td>JSP</td>
<td>Returns an error object that stores database connection errors</td>
</tr>
</tbody>
</table>

**Examples**
The following example for an ASP target connects to a database and retrieves some data. After each database operation, it uses GetError to check for errors. If an error occurs, it displays the error code and message:

```csharp
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 );
}
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` has the following behavior:

<table>
<thead>
<tr>
<th>Application server</th>
<th>Runtime behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASP</td>
<td>Returns the most recent error message, which is obtained by accessing the Description property of the Error object</td>
</tr>
<tr>
<td>JSP</td>
<td>Returns the most recent error message stored in the named error object</td>
</tr>
</tbody>
</table>

Examples
The following example for an ASP target connects to a database and retrieves some data. If an error occurs, it uses `GetMessage` to get the error message:

```csharp
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 );
}
```
GetNextError

Description Returns the next error object, if one exists.

Applies to PSErrorClass objects

Syntax `errorobject.GetNextError()`

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>errorobject</td>
<td>A variable that contains a reference to an instance of PSErrorClass</td>
</tr>
</tbody>
</table>

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 has the following behavior:

<table>
<thead>
<tr>
<th>Application server</th>
<th>Runtime behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASP</td>
<td>Returns the next error object</td>
</tr>
<tr>
<td>JSP</td>
<td>Returns the next error stored in the named error object</td>
</tr>
</tbody>
</table>

Examples The following example for an ASP target connects to a database and retrieves some data. If an error occurs, it uses GetNextError to access the list of error objects. For each error object, it writes out the error code and message text:

```cpp
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();
{
    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 )

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>argvar</td>
<td>The name of a parameter passed to the current page</td>
</tr>
</tbody>
</table>

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

Usage
At runtime, GetParam has the following behavior:

<table>
<thead>
<tr>
<th>Application server</th>
<th>Runtime behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASP</td>
<td>Accesses either the QueryString or the Form collection of the Request object:</td>
</tr>
<tr>
<td></td>
<td>• When the METHOD attribute of a &lt;FORM&gt; is GET (or when data is passed directly in the HREF attribute of an &lt;A&gt; element), GetParam accesses the specified parameter in the QueryString collection.</td>
</tr>
<tr>
<td></td>
<td>• When the METHOD attribute of a &lt;FORM&gt; is POST, GetParam accesses the specified parameter in the Form collection.</td>
</tr>
<tr>
<td>JSP</td>
<td>Calls the getParameter method on the request object.</td>
</tr>
</tbody>
</table>

Examples
The following example retrieves the value of the “EmpID” parameter:

```javascript
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 JSP 4GL targets only.

Applies to
PSArgClass

Syntax
ArgObj.GetParameterString ( )

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)`

Return value
int

Examples
The following example 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. This method is for use in JSP targets only.

Applies to
PSCursorClass objects

Syntax
`cursorobject.GetResultSet()`

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cursorobject</code></td>
<td>A variable that contains a reference to an instance of PSCursorClass</td>
</tr>
</tbody>
</table>

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>\n" + 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("&nbsp&nbsp");
    psDocument.Write(rs.getString("dept_name"));
    psDocument.Write("&nbsp&nbsp");
    psDocument.WriteLine(Integer.toString(rs.getInt(
    "dept_head_id")));
}

See also GetResultSetMetaData
GetResultSetMetaData

GetResultSetMetaData

Description
Returns the metadata result set in a cursor. This method is for use in JSP targets only.

Applies to
PSCursorClass objects

Syntax

```
cursorobject.GetResultSetMetaData ( )
```

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);
    PSCursorClass 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) + ")	, 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
\textit{cursorobject}\textunderscore GetRowCount( )

Return value
Number in ASP targets, int in JSP targets

Usage
At runtime, GetRowCount has the following behavior:

<table>
<thead>
<tr>
<th>Application server</th>
<th>Runtime behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASP</td>
<td>Gets the value of the RecordCount property of the RecordSet object</td>
</tr>
<tr>
<td>JSP</td>
<td>Calculates the number of rows in the ResultSet object used by the cursor instance</td>
</tr>
</tbody>
</table>

Examples
The following example gets the row count for a cursor that retrieves rows from the customer table:

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

To use the same example in a JSP target, you must declare “mycursor” as a variable of type PSCursorClass before instantiating it and calling any methods on it. You must also declare “myquery” as a variable of type String, “myconnect” as a variable of type PSConnectionClass, and “rowcount” as a variable of type int.
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)`

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cursorobject</code></td>
<td>A variable that contains a reference to an instance of PSCursorClass</td>
</tr>
<tr>
<td><code>iCol</code></td>
<td>Number of the column for which you want to obtain the scale</td>
</tr>
</tbody>
</table>

Return value
`int`

Examples
The following example 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.

<table>
<thead>
<tr>
<th>To get the value of</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>A column in a cursor</td>
<td>Syntax 1</td>
</tr>
<tr>
<td>A session variable</td>
<td>Syntax 2</td>
</tr>
</tbody>
</table>

Syntax 1  

For PSCursorClass objects

Description
Retrieves the value of a column in a cursor.

Applies to
PSCursorClass objects

Syntax

cursorobject.GetValue( field )

Argument | Description
-----------|------------------
cursorobject | A variable that contains a reference to an instance of PSCursorClass.
field | The name or number of a column in the cursor. If you specify a number, you need to keep in mind that the array of column numbers starts with zero.

For cursors with aggregate functions
If your cursor’s SQL statement performs a SELECT with an aggregate function, you must specify a column number (not a column name) for field.

Return value
The value of the specified column

Usage
At runtime, GetValue has the following behavior:

<table>
<thead>
<tr>
<th>Application server</th>
<th>Runtime behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASP</td>
<td>Accesses the named item in the Fields collection of the RecordSet object</td>
</tr>
<tr>
<td>JSP</td>
<td>Not supported</td>
</tr>
</tbody>
</table>

For JSP targets, use the GetColumn<DataType> methods instead, where <DataType> is the datatype of the value you want to retrieve.

Examples
The following example retrieves the values of the “prod_id” and “prod_name” columns:

    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 );
}

Syntax 2  For the PSSessionClass object

Description  Retrieves the value of a session variable.

Applies to  PSSessionClass object

Syntax  psSession.GetValue( propname )

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>propname</td>
<td>The name of a session variable</td>
</tr>
</tbody>
</table>

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:

```powershell
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:

```java
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. This method is available to ASP targets only.

**Applies to**
PServerClass object

**Syntax**
```
psServer.MapPath(path)
```

**Argument** | **Description**
--- | ---
`path` | The relative or virtual path to map to a physical directory. If the path argument starts with either a forward or backward slash, MapPath returns the physical path that corresponds to this virtual path. If the path argument does not start with a slash, MapPath returns a physical path relative to the directory where the current server page is located.

**Return value**
Returns a physical path on the server.

**Usage**
At runtime, MapPath calls the MapPath method of the Server object.

**Examples**
The following example specifies a relative path as an argument to MapPath. If the current server page is located in `C:\WEBSERVER\SCRIPTS`, this call to MapPath returns `C:\WEBSERVER\SCRIPTS\SERVER\MYFILE.ASP`:
```
file = psServer.MapPath("scripts\myfile.asp");
```

The following example specifies a full virtual path as an argument to MapPath. Assuming that the server’s home directory is `C:\WEBSERVER`, this call to MapPath returns `C:\WEBSERVER\GRAPHICS\MYFILE.GIF`:
```
file = psServer.MapPath("/graphics/myfile.gif");
```
Move

Description
Moves to an absolute row in a cursor.

Applies to
PSCursorClass objects

Syntax
 cursorobject.Move( rownumber )

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cursorobject</td>
<td>A variable that contains a reference to an instance of PSCursorClass.</td>
</tr>
<tr>
<td>rownumber</td>
<td>An absolute row number in the cursor. The array of rows starts with index zero.</td>
</tr>
</tbody>
</table>

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

Usage
At runtime, Move has the following behavior:

<table>
<thead>
<tr>
<th>Application server</th>
<th>Runtime behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASP</td>
<td>Calls the Move method of the RecordSet object. Because ASP does not support an absolute move operation, the Web Targets Move converts the row you specify into a relative number before calling the ASP Move function.</td>
</tr>
<tr>
<td>JSP</td>
<td>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.</td>
</tr>
</tbody>
</table>

Examples
The following example moves to row 10 of the “mycursor” object. 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");
    }

To use the same example in a JSP target, you must declare “myquery” as a variable of type String, “mycursor” as a variable of type PSCursorClass, and “myconnect” as a variable of type PSConnectionClass.
MoveFirst

Description: Moves to the first row in a cursor.

Applies to: PSCursorClass objects

Syntax: `cursorobject.MoveFirst()`

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cursorobject</code></td>
<td>A variable that contains a reference to an instance of PSCursorClass</td>
</tr>
</tbody>
</table>

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

Usage: At runtime, MoveFirst has the following behavior:

<table>
<thead>
<tr>
<th>Application server</th>
<th>Runtime behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASP</td>
<td>Calls the <code>MoveFirst</code> method of the <code>RecordSet</code> object</td>
</tr>
<tr>
<td>JSP</td>
<td>Calls the first method of the <code>ResultSet</code> object</td>
</tr>
</tbody>
</table>

Examples: The following example moves to the first row in the “mycursor” object:

```powershell
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()`

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cursorobject</code></td>
<td>A variable that contains a reference to an instance of PSCursorClass</td>
</tr>
</tbody>
</table>

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

Usage: At runtime, MoveLast has the following behavior:

<table>
<thead>
<tr>
<th>Application server</th>
<th>Runtime behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASP</td>
<td>Calls the MoveLast method of the RecordSet object</td>
</tr>
<tr>
<td>JSP</td>
<td>Calls the last method on the ResultSet object</td>
</tr>
</tbody>
</table>

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

```java
    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( )
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cursorobject</td>
<td>A variable that contains a reference to an instance of PSCursorClass</td>
</tr>
</tbody>
</table>

Return value  Boolean. Returns true if the end of the cursor has been reached, and false if it has not.
Usage  At runtime, MoveNext has the following behavior:

<table>
<thead>
<tr>
<th>Application server</th>
<th>Runtime behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASP</td>
<td>Calls the MoveNext method of the RecordSet object</td>
</tr>
<tr>
<td>JSP</td>
<td>Calls the next method on the ResultSet object</td>
</tr>
</tbody>
</table>

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( )

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cursorobject</td>
<td>A variable that contains a reference to an instance of PSCursorClass</td>
</tr>
</tbody>
</table>

Return value
Boolean. Returns true if the end of the cursor has been reached, and false if it has not.
Usage
At runtime, MovePrevious has the following behavior:

<table>
<thead>
<tr>
<th>Application server</th>
<th>Runtime behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASP</td>
<td>Calls the MovePrevious method of the RecordSet object</td>
</tr>
<tr>
<td>JSP</td>
<td>Calls the previous method on the ResultSet object</td>
</tr>
</tbody>
</table>

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 has the following behavior:

<table>
<thead>
<tr>
<th>Application server</th>
<th>Runtime behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASP</td>
<td>Returns the string “ASP”</td>
</tr>
<tr>
<td>JSP</td>
<td>Returns the string “JSPObject”</td>
</tr>
</tbody>
</table>

**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:

```powershell
serverType = psServer.ObjectModelType();
if (serverType == "ASP")
{
    // Perform ASP-specific logic
}
else if (serverType == "JSPObject")
{
    // Perform JSP-specific logic
}
```
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 has the following behavior:

<table>
<thead>
<tr>
<th>Application server</th>
<th>Runtime behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASP</td>
<td>Returns the string “1.0”</td>
</tr>
<tr>
<td>JSP</td>
<td>Returns the string “10.0” for the PowerBuilder 10.0 Web Target object model.</td>
</tr>
</tbody>
</table>

Examples: The following example performs different logic depending on the version of the Web Target object model you are using:

```java
version = psServer.ObjectModelVersion();
if (version == "1.0") {
    // Perform 1.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 example returns the path portion of the current URL into a variable called mypath. If the URL for the current document is http://MyServer/Files/myscript.asp, the value of mypath is set to “/Files”:

```
myPath = psDocument.Path( );
```
Redirect

Redirects the client’s browser to another page.

<table>
<thead>
<tr>
<th>To redirect</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>From a 4GL Web page</td>
<td>Syntax 1</td>
</tr>
<tr>
<td>From a Web page that is not 4GL Web-enabled</td>
<td>Syntax 2</td>
</tr>
</tbody>
</table>

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 JSP targets only.

Applies to
psPage object

Syntax
psPage.Redirect ( string destination, PSArgClass argument )

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>destination</td>
<td>This must be a URL.</td>
</tr>
<tr>
<td>argument</td>
<td>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.</td>
</tr>
</tbody>
</table>

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:

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

Syntax 2 For pages not 4GL Web enabled

Description
Redirects the current request to another URL.

Applies to
PSDocumentClass object

Syntax
psDocument.Redirect( URL )

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>URL</td>
<td>The URL to which the current request is being directed</td>
</tr>
</tbody>
</table>

Return value
None.

Usage
At runtime, Redirect has the following behavior:

<table>
<thead>
<tr>
<th>Application server</th>
<th>Runtime behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASP</td>
<td>Sets the value of the redirect property of the document object</td>
</tr>
<tr>
<td>JSP</td>
<td>Calls thesendRedirect method on the response object</td>
</tr>
</tbody>
</table>

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

```powershell
```
ReportError

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

Applies to
psPage object

Syntax
```java
psPage.ReportError(string location, string cause, string 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:

```java
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 JSP 4GL targets only.

Applies to
PSArgClass

Syntax
ArgObj.setCharacterEncoding ( String enc )

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ArgObj</td>
<td>Object of the PSArgClass type for which you want to change the character set</td>
</tr>
<tr>
<td>enc</td>
<td>Character set encoding you want to use for adding arguments to a PSArgClass object</td>
</tr>
</tbody>
</table>

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:

```java
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)

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>columnName</td>
<td>A string that specifies the name of the column that you want to link to a target page.</td>
</tr>
<tr>
<td>link</td>
<td>A string that specifies the URL target of a link (HTML A element) from a data item in the column.</td>
</tr>
<tr>
<td>linkArgs</td>
<td>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.</td>
</tr>
<tr>
<td>linkTarget</td>
<td>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.</td>
</tr>
</tbody>
</table>

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. The behavior is the same for ASP and JSP Web targets.

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

```javascript
htmlDwObj1.SetColumnLink("emp_id",
    "empdetail.htm","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 )
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>commandobject</td>
<td>A variable that contains a reference to an instance of PSCommandClass</td>
</tr>
<tr>
<td>SQLstring</td>
<td>A string that specifies the SQL statement</td>
</tr>
</tbody>
</table>

**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();
```
CHAPTER 4  Web Target Methods

**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 JSP 4GL targets only.

**Applies to**
psPage object

**Syntax**
```java
psPage.SetTrace ( boolean bTraceOn )
```

**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 example turns tracing on from a server-side event:

```java
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 has the following behavior:

<table>
<thead>
<tr>
<th>Application server</th>
<th>Runtime behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASP</td>
<td>Assigns a value to a user-defined property of the Session object</td>
</tr>
<tr>
<td>JSP</td>
<td>Assigns a value to a user-defined property of the session object</td>
</tr>
</tbody>
</table>

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 HTML or 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)

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bAllowForm</td>
<td>Boolean indicating whether or not the object supports data input:</td>
</tr>
<tr>
<td></td>
<td>• true (default) The object supports data input.</td>
</tr>
<tr>
<td></td>
<td>• false The object does not support data input. The object will provide only navigation.</td>
</tr>
<tr>
<td></td>
<td>bAllowForm must be set to true to set the bClientValidation and bClientFormatting arguments to true.</td>
</tr>
<tr>
<td>bClientValidation</td>
<td>Boolean indicating whether or not the client validates 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.</td>
</tr>
<tr>
<td></td>
<td>The bAllowForm argument must be set to true to use client-side validation:</td>
</tr>
<tr>
<td></td>
<td>• true (default) The client validates the syntax of the entries that users make.</td>
</tr>
<tr>
<td></td>
<td>• false The client does not support syntax validation for entries that users make.</td>
</tr>
<tr>
<td>bClientEvents</td>
<td>Boolean indicating whether or not the object supports invoking client-side events:</td>
</tr>
<tr>
<td></td>
<td>• true (default) Invoking client-side events is supported.</td>
</tr>
<tr>
<td></td>
<td>• false Invoking client-side events is not supported.</td>
</tr>
<tr>
<td>bClientScriptable</td>
<td>Boolean indicating whether or not you can add scripts to manipulate the Web DataWindow control on the client.</td>
</tr>
<tr>
<td></td>
<td>The scripts that you add would run on the client system.</td>
</tr>
<tr>
<td></td>
<td>• true Client-side scripting is supported.</td>
</tr>
<tr>
<td></td>
<td>• false (default) Client-side scripting is not supported.</td>
</tr>
</tbody>
</table>
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 example returns the domain name of the current document into a variable called “mydomain”:

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

---

**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 example returns the domain name of the current document into a variable called ”mydomain”:

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

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

Applies to
psPage object

Syntax
```java
psPage.TestCompError ( string location )
```

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:

```java
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 JSP Web pages only.

Applies to
psPage object

Syntax
```plaintext
psPage.Trace( string message )
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>message</td>
<td>Message to be displayed in the internal trace buffer</td>
</tr>
</tbody>
</table>

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:

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

See also
IsTrace
SetTrace
TraceIndent
TraceOutdent
**TraceIndent**

<table>
<thead>
<tr>
<th>Description</th>
<th>Increases the indent level of trace messages. This method is for 4GL JSP Web pages only.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Applies to</td>
<td>psPage object</td>
</tr>
<tr>
<td>Syntax</td>
<td><code>psPage.TraceIndent()</code></td>
</tr>
<tr>
<td>Return value</td>
<td>None</td>
</tr>
<tr>
<td>Usage</td>
<td>Each call to the <code>TraceIndent</code> method increases the indent level of subsequent messages in the trace buffer. You can reduce the indent level of trace messages by calling <code>TraceOutdent</code>.</td>
</tr>
<tr>
<td>Examples</td>
<td>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:</td>
</tr>
<tr>
<td></td>
<td><code>psPage.TraceIndent();</code></td>
</tr>
<tr>
<td></td>
<td><code>psPage.Trace(MyVar + &quot; is the value of MyVar at this time&quot;);</code></td>
</tr>
<tr>
<td></td>
<td><code>psPage.TraceOutdent();</code></td>
</tr>
<tr>
<td>See also</td>
<td>IsTrace</td>
</tr>
<tr>
<td></td>
<td>SetTrace</td>
</tr>
<tr>
<td></td>
<td>Trace</td>
</tr>
<tr>
<td></td>
<td>TraceOutdent</td>
</tr>
</tbody>
</table>
TraceOutdent

Description
Decreases the indent level of trace messages. This method is for 4GL JSP 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

<table>
<thead>
<tr>
<th>Description</th>
<th>Identifies the Web server you are running.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Applies to</td>
<td>PSServerClass object</td>
</tr>
<tr>
<td>Syntax</td>
<td>psServer.Type()</td>
</tr>
<tr>
<td>Return value</td>
<td>String</td>
</tr>
</tbody>
</table>

#### 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:

```java
serverType = psServer.Type( );
if (serverType == "Microsoft-PWS") {
  // Perform platform-specific logic
}
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 has the following behavior:

<table>
<thead>
<tr>
<th>Application server</th>
<th>Runtime behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASP</td>
<td>Calls the URLEncode method of the Server object</td>
</tr>
<tr>
<td>JSP</td>
<td>Calls the encode method of java.net.URLEncoder</td>
</tr>
</tbody>
</table>

**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:

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

text = "\n\n\n\n"<P>Increase salary by:</P>\n"<A HREF="salary.asp?increase=" + value1 + "+33\%"></A> <BR>"<A HREF="salary.asp?increase=" + value2 + "+50\%"></A> <BR>"<A HREF="salary.asp?increase=" + value3 + "+100\%"></A>\n\n\npsDocument.Write(text);
```

This code sends the following HTML to the browser:

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

```java
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 )

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>The output string</td>
</tr>
</tbody>
</table>

Return value
None

Usage
At runtime, Write has the following behavior:

<table>
<thead>
<tr>
<th>Application server</th>
<th>Runtime behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASP</td>
<td>Calls the Write method of the Response object</td>
</tr>
<tr>
<td>JSP</td>
<td>Calls the print method on javax.servlet.jsp.JspWriter</td>
</tr>
</tbody>
</table>

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

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

Description
Writes the current errors at the current place in the page. This method is for
4GL JSP 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**

```plaintext
psDocument.WriteLn( string )
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>The output string</td>
</tr>
</tbody>
</table>

**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 `<br>` element to the HTML source.

At runtime, `WriteLn` has the following behavior:

<table>
<thead>
<tr>
<th>Application server</th>
<th>Runtime behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASP</td>
<td>Calls the <code>Write</code> method of the Response object</td>
</tr>
<tr>
<td>JSP</td>
<td>Calls the <code>print</code> method on <code>javax.servlet.jsp.JspWriter</code></td>
</tr>
</tbody>
</table>

**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:

```plaintext
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 for JSP targets.

Contents

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>About the Web DataWindow custom tag library</td>
<td>137</td>
</tr>
<tr>
<td>Example using the Web DataWindow custom tag library</td>
<td>141</td>
</tr>
</tbody>
</table>

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 `DataWindow100.tld`. To use the tag library, place the `DataWindow100.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.
About the Web DataWindow custom tag library

**DataWindow**

**Description**

Sets parameters for a Web DataWindow on a JSP page.

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

**Attributes**

<table>
<thead>
<tr>
<th>Attributes of DataWindow tag</th>
<th>Java type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>action</td>
<td>String</td>
<td>See the <code>action</code> argument for the <code>SetAction</code> method in the DataWindow Reference.</td>
</tr>
<tr>
<td>allowForm</td>
<td>boolean</td>
<td>See the <code>allowForm</code> argument for the <code>PSDataWindowClass.SetWeight</code> method.</td>
</tr>
<tr>
<td>argument</td>
<td>String</td>
<td>See the <code>argument</code> argument for the <code>RetrieveEx</code> method in the DataWindow Reference.</td>
</tr>
<tr>
<td>clientEvents</td>
<td>boolean</td>
<td>See the <code>clientEvents</code> argument for the <code>PSDataWindowClass.SetWeight</code> method.</td>
</tr>
<tr>
<td>clientFormatting</td>
<td>boolean</td>
<td>See the <code>clientFormatting</code> argument for the <code>PSDataWindowClass.SetWeight</code> method.</td>
</tr>
<tr>
<td>clientScriptable</td>
<td>boolean</td>
<td>See the <code>clientScriptable</code> argument for the <code>PSDataWindowClass.SetWeight</code> method.</td>
</tr>
<tr>
<td>clientValidation</td>
<td>boolean</td>
<td>See the <code>clientValidation</code> argument for the <code>PSDataWindowClass.SetWeight</code> method.</td>
</tr>
<tr>
<td>context</td>
<td>String</td>
<td>See the <code>context</code> argument for the <code>SetAction</code> method in the DataWindow Reference.</td>
</tr>
<tr>
<td>database</td>
<td>String</td>
<td>See the <code>database</code> constructor for <code>PSConnectionParmsClass</code>.</td>
</tr>
<tr>
<td>dbms</td>
<td>String</td>
<td>See the <code>dbms</code> constructor for <code>PSConnectionParmsClass</code>.</td>
</tr>
<tr>
<td>dbparm</td>
<td>String</td>
<td>See the <code>dbparm</code> constructor for <code>PSConnectionParmsClass</code>.</td>
</tr>
<tr>
<td>dwHTMLObjectName</td>
<td>String</td>
<td>See the <code>objectname</code> argument for the <code>SetHTMLObjectName</code> method in the DataWindow Reference.</td>
</tr>
<tr>
<td>dwName</td>
<td>String</td>
<td>See the <code>dwName</code> property for <code>PSDataWindowSourceClass</code>.</td>
</tr>
<tr>
<td>fourGLWeb</td>
<td>boolean</td>
<td>You must set this to <code>true</code> in a 4GL page.</td>
</tr>
<tr>
<td>id</td>
<td>String</td>
<td>Optional identifier.</td>
</tr>
<tr>
<td>jaglogid</td>
<td>String</td>
<td>(Optional) See the <code>userld</code> constructor for <code>PSJaguarConnection</code>.</td>
</tr>
</tbody>
</table>
### Attributes of DataWindow tag

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Java Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>jaglogpass</td>
<td>String</td>
<td>(Optional) See the <code>password</code> constructor for PSJaguarConnection.</td>
</tr>
<tr>
<td>jagservename</td>
<td>String</td>
<td>See the <code>serverName</code> constructor for PSJaguarConnection.</td>
</tr>
<tr>
<td>libName</td>
<td>String</td>
<td>See the <code>sourceFileName</code> property for PSDataWindowSourceClass.</td>
</tr>
<tr>
<td>lock</td>
<td>String</td>
<td>See the <code>lock</code> constructor for PSConnectionParmsClass.</td>
</tr>
<tr>
<td>logid</td>
<td>String</td>
<td>See the <code>user</code> constructor for PSConnectionParmsClass.</td>
</tr>
<tr>
<td>logpass</td>
<td>String</td>
<td>See the <code>password</code> constructor for PSConnectionParmsClass.</td>
</tr>
<tr>
<td>pageSize</td>
<td>String</td>
<td>(Optional) See the <code>pagesize</code> argument for the SetPageSize DataWindow method in the DataWindow Reference.</td>
</tr>
<tr>
<td>selfLink</td>
<td>String</td>
<td>The URL for the current page.</td>
</tr>
<tr>
<td>selfLinkArg</td>
<td>String</td>
<td>Page parameters to be passed to the server. The syntax is:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[ argname = 'exp'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>where <code>argname</code> is a page parameter and <code>exp</code> is a DataWindow expression whose value is a string. See HTMLGen.property in the DataWindow Reference for more information.</td>
</tr>
<tr>
<td>servername</td>
<td>String</td>
<td>See the <code>serverName</code> constructor for PSConnectionParmsClass.</td>
</tr>
</tbody>
</table>
About the Web DataWindow custom tag library

**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.

<table>
<thead>
<tr>
<th>Attributes of DWColumnLink tag</th>
<th>Java type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sColumnName</td>
<td>String</td>
<td>The name of the column that you want to link to a target page.</td>
</tr>
<tr>
<td>sColLink</td>
<td>String</td>
<td>The URL target of a link from a data item in the column.</td>
</tr>
<tr>
<td>sColLinkArgs</td>
<td>String</td>
<td>The arguments passed with the <code>link</code> argument.</td>
</tr>
<tr>
<td>sColLinkTarget</td>
<td>String</td>
<td>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 <code>sColLinkTarget</code> to link from a master to a detail page by specifying a different window or frame for the detail page.</td>
</tr>
</tbody>
</table>
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 “DW100” with the DataWindow100.tld file in the Web application’s /WEB-INF/tlds directory:

```xml
<taglib>
  <taglib-uri>/DW100</taglib-uri>
  <taglib-location>/WEB-INF/tlds/DataWindow100.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 Web and JSP Targets book.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<%@ taglib prefix="webdw" uri="/DW100" %>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html">
<title></title>
</head>
<body psparams="">
<%-- Use DataWindow custom tag--%>
<webdw:DataWindow argument=""
  selfLinkArg=""
  logpass=""
  jaglogpass=""
  dbms="ODBC"
```
Example using the Web DataWindow custom tag library

servername=""
cientscriptable="true"
clientsFormattign="true"
action=""
selLink="dwpage2.jsp"
jaglogid="jagadmin"
dwHtmlObjectName="dwTest"
logid="" lock=""
clientsEvents="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>
</webdw:DataWindow>
</BODY>
</HTML>

Employees.jsp

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<%@ taglib prefix="webdw" uri="/DW100" %>
<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="" jagsservername="myEASserver:9000"
jaglogid="jagadmin"
jaglogpass=""
selfLink="Employees.jsp"
selfLinkArg=""
action=""
context=""
argument="<%=strDept%>"
dwHtmlObjectName="dwTest"
pageSize="10"
>
</webdw:DataWindow>
</BODY></HTML>
Example using the Web DataWindow custom tag library
## Index

### A
- Abandon method 65
- addArg method 66
- AfterAction event 38, 56
- AfterBinding event 39
- AfterGenerate event 40
- AfterRetrive event 56
- AfterUpdate event 57
- Alert method 67

### B
- BeforeAction event 41, 57
- BeforeBinding event 42
- BeforeGenerate event 43
- BeforeRetrieve event 58
- BeforeUpdate event 58

### C
- ClearError method 68
- CreateTable method 69
- CreateConnection method 70
- CreateCursor method 72
- custom tag library, Web DataWindow 137

### D
- DataWindow custom tag 138
- DWColumn link custom tag 140

### E
- EOF method 74
- Execute method 75

### F
- File method 76
- FillRetrievalArguments method 77
- FirstTime event 44

### G
- Generate method 78
- GenerateXHTML method 79
- GenerateXMLWeb method 80
- getCharacterEncoding method 82
- GetCode method 83
- getColumn method 85
- getColumnCount method 84
- getColumnLength method 86
- getColumn method 87
- getColumn method 88
- getColumn method 89
- getConnection method 90
- GetEnv method 91
- GetError method 93
- GetMessage method 95
- getNextError method 96
- GetParam method 97
- GetParameterString method 98
- GetPrecision method 99
- GetResultSet method 100
- GetResultSetMetaData method 102
- GetRowCount method 103
- GetScale method 104
- GetValue method 105

## ItemChanged event 45
Index

M
MapPath method 108
Move 109
MoveFirst method 110
MoveLast method 111
MoveNext method 112
MovePrevious method 113

O
ObjectModelType method 114
ObjectModelVersion method 115
OnDBError event 59

P
Path method 116
PSArgClass 2
PSButtonClick 2
PSCheckBoxClass 3
PSCommandClass 4
PSConnectionClass 5
PSConnectionParmsClass 7
PSCursorClass 9
PSDataWindowClass 10
PSDataWindowSourceClass 14
PSDocumentClass 15
PSDropDownListClass 16
PSErrorClass 17
PSImageClass 18
PSJaguarConnection 18
PSLinkClass 20
PSNamedConnectionParmsClass 21
PsPage 22
PSPasswordClass 24
PSRadioGroupClass 25
PSServerClass 26
PSSessionClass 27
PSScrollBarClass 27
PSTextFieldClass 28
PSTextFieldClass 29
PSWebDataWindowClass 30

R
Redirect method 117
ReportError method 119
RequestFinish event 46
RequestStart event 46

S
ServerAction event 47
ServerError event 48
setCharacterEncoding method 120
SetColumnLink method 121
SetSQL method 122
SetTrace method 123
SetValue method 124
SetWeight method 125
Site method 126

T
TestCompError method 127
Trace method 107, 128
TraceIndent method 129
TraceOutdent method 130
Type method 131

U
URLDecode method 132

V
Validate event 50, 60
ValidationEvent event 52, 61
Version method 133

W
WebDataWindow custom tag library 137
Write method 134
WriteErrorsToDocument method  135
WriteLn method  136