SYBASE®

DataWindow® Programmer's Guide

PowerBuilder®

11.0

DOCUMENT ID: DC37775-01-1100-01

LAST REVISED: May 2007

Copyright © 1991-2007 by Sybase, Inc. All rights reserved.

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.

Sybase trademarks can be viewed at the Sybase trademarks page at http://www.sybase.com/detail?id=1011207. Sybase and the marks listed are trademarks of Sybase, Inc. ® indicates registration in the United States of America.

Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

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.

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

Contents

About This Book ..... **ix**

PART 1 **DATAWINDOW AND DATASTORE BASICS**

CHAPTER 1 **About DataWindow Technology** **3**
About DataWindow objects, controls, and components 3
Presentation styles and data sources 5
Basic process 7
Choosing a DataWindow technology 7
Solutions for client/server and distributed applications 8
Solutions for Web applications 8
PowerBuilder DataWindow control 11

CHAPTER 2 **Using DataWindow Objects** **13**
About using DataWindow objects 13
Putting a DataWindow object into a control 14
Names for DataWindow controls and DataWindow objects 15
Working with the DataWindow control in PowerBuilder 16
Specifying the DataWindow object during execution 18
Accessing the database 21
Setting the transaction object for the DataWindow control 21
Retrieving and updating data 26
Accessing a Web service data source 29
Importing data from an external source 29
Manipulating data in a DataWindow control 30
How a DataWindow control manages data 30
Accessing the text in the edit control 32
Manipulating the text in the edit control 33
Coding the ItemChanged event 33
Coding the ItemError event 34
Accessing the items in a DataWindow 34
Using other DataWindow methods 36

Accessing the properties of a DataWindow object 37
Handling DataWindow errors 39
Retrieve and Update errors and the DBError event 39
Errors in property and data expressions and the Error event.. 42
Updating the database 44
How the DataWindow control updates the database 44
Changing row or column status programmatically 46
Creating reports 47
Planning and building the DataWindow object 48
Printing the report 48
Using nested reports 49
Using crosstabs 52
Viewing the underlying data 52
Letting users redefine the crosstab 52
Modifying the crosstab's properties during execution 54
Generating HTML 55
Controlling display 59
Calling the SaveAs method 61
Displaying DataWindow objects as HTML forms 62

CHAPTER 3 Dynamically Changing DataWindow Objects 67
About dynamic DataWindow processing 67
Modifying a DataWindow object 68
Creating a DataWindow object 69
Providing query ability to users 72
How query mode works 72
Using query mode 74
Providing Help buttons 77
Reusing a DataWindow object 77

CHAPTER 4 Using DataStore Objects 79
About DataStores 79
Working with a DataStore 82
Using a custom DataStore object 82
Accessing and manipulating data in a DataStore 84
Sharing information 86
Example: printing data from a DataStore 88
Example: using two DataStores to process data 89

CHAPTER 5 **Manipulating Graphs 93**
Using graphs 93
Modifying graph properties 94
How parts of a graph are represented 95
Referencing parts of a graph 96
Accessing data properties 97
Getting information about the data 97
Saving graph data 99
Modifying colors, fill patterns, and other data 100
Using graph methods 101
Using point and click 103

PART 2 **USING THE DATAWINDOW IN JSP AND WEB APPLICATIONS**

CHAPTER 6 **Using the Web DataWindow 109**
What the Web DataWindow is 109
Web DataWindow types 110
How the Web DataWindow works 111
The Web DataWindow server component and client control 113
Using the XML Web DataWindow 115
About XML, XSLT, CSS, and XHTML 116
How the XML Web DataWindow works 117
How to use the XML Web DataWindow 121
Designing DataWindow objects for the Web DataWindow 123
Web DataWindow properties 126
Controlling the size of generated code 132
Using drop-down DataWindows 132
Callback and client-side paging support 133
Using JavaScript caching for Web DataWindow methods 135
Using expressions 138
Using foreign language character sets 139
Providing links for data 139
Rendering HTML for controls in an HTML Web DataWindow 140
Using Button and Picture controls 141
Specifying Web generation for a specific browser 143
Previewing the DataWindow 144
Setting Web DataWindow design-time control properties 144
Setting up database connections 147
Deploying DataWindow objects to the component server 149
The Web DataWindow Container project wizard 149
Creating Web page templates 152
Writing client-side scripts 154

Customizing Web DataWindow generation 157
The Export Template view for XHTML 157
What you can customize 158
The default XHTML export template 159
Managing templates 160
Template structure 164
Editing XHTML export templates 167
Selecting XHTML export templates at runtime 174
Exporting the DataWindow Web form in XML and XSLT or in XHTML 174

CHAPTER 7 Server-Side Processing for the Web DataWindow 177
Server configuration requirements 177
Instantiating and configuring the server component 182
Instantiating the component 183
Loading the DataWindow object 185
Controlling what is generated 186
Specifying the database connection and retrieving data 187
Passing page-specific data to the reloaded page 189
Passing user actions to the server component 192
Inserting the generated HTML or XHTML into the page 193
Using a custom server component 194
Creating a custom server component in EAServer 196
Setting properties for a custom component in EAServer 197
Instantiating the custom component 200
Maintaining state on the server 201
Using service classes 203
Defining a service class for PowerBuilder components 204
Defining a service class for Java components 207

CHAPTER 8 Using the DataWindow Web Control for ActiveX 211
About the Web ActiveX 211
HTML for inserting the controls on a Web page 215
Object element 215
Properties and Param elements 217
How to use the Web ActiveX in a JSP target 219
DataWindow objects for the Web ActiveX 221
What the DataWindow object can include 221
Managing DataWindow objects in PowerBuilder libraries 222
Specifying a DataWindow object for the control 222

Using the DataWindow Transaction Object control 224
Making database connections 225
Connecting and retrieving data 226
Coding for the Web ActiveX 227
Datatypes for method arguments and return values 227
Setting event return codes 228
Deploying the Web ActiveX 228

Index 231

About This Book

Subject

This book provides information about using DataWindow® technology in client/server, distributed, and Web applications. It describes how to define DataWindow® objects appropriate for your application and how to write code that interacts with those DataWindow objects.

Audience

This book is for anyone developing applications that use DataWindow technology. It assumes that:

- You are familiar with the DataWindow painter. If not, see the *PowerBuilder User's Guide*.
- You have a basic familiarity with the language that corresponds with your development environment: PowerScript®, JavaScript, or Java. If not, see the PowerBuilder® documentation and other background material for your language of choice.

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/.

Sybase certifications on the Web

Technical documentation at the Sybase Web site is updated frequently.

❖ **Finding the latest information on product certifications**

1. Point your Web browser to Technical Documents at http://www.sybase.com/support/techdocs/.
2. Click Certification Report.
3. In the Certification Report filter select a product, platform, and timeframe and then click Go.
4. Click a Certification Report title to display the report.

❖ **Finding the latest information on component certifications**

1. Point your Web browser to Availability and Certification Reports at http://certification.sybase.com/.
2. Either select the product family and product under Search by Base Product; or select the platform and product under Search by Platform.
3. Select Search to display the availability and certification report for the selection.

❖ **Creating a personalized view of the Sybase Web site (including support pages)**

Set up a MySybase profile. MySybase is a free service that allows you to create a personalized view of Sybase Web pages.

1. Point your Web browser to Technical Documents at http://www.sybase.com/support/techdocs/.
2. Click MySybase and create a MySybase profile.

Sybase EBFs and software maintenance

You can find information about EBFs and software maintenance on the Sybase Web site.

❖ **Finding the latest information on EBFs and software maintenance**

1. Point your Web browser to the Sybase Support Page at http://www.sybase.com/support.
2. Select EBFs/Maintenance. If prompted, enter your MySybase user name and password.
3. Select a product.
4. Specify a time frame and click Go. A list of EBF/Maintenance releases is displayed.

   Padlock icons indicate that you do not have download authorization for certain EBF/Maintenance releases because you are not registered as a Technical Support Contact. If you have not registered, but have valid information provided by your Sybase representative or through your support contract, click Edit Roles to add the "Technical Support Contact" role to your MySybase profile.
5. Click the Info icon to display the EBF/Maintenance report, or click the product description to download the software.

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.

PART 1

DataWindow and DataStore basics

This part describes how to create and use DataWindow and DataStore objects.

Additional information about these objects and about the DataWindow control is available in the *User's Guide* and in *Application Techniques*. Reference information is available in the *DataWindow Reference* guide and in the online Help.

CHAPTER 1

About DataWindow Technology

About this chapter

This chapter describes what DataWindow objects are and the ways you can use them in various application architectures and programming environments.

Contents

Topic	Page
About DataWindow objects, controls, and components	3
Choosing a DataWindow technology	7
PowerBuilder DataWindow control	11

About DataWindow objects, controls, and components

DataWindow technology is implemented in two parts:

- **A DataWindow object** The DataWindow object defines the data source and presentation style for the data.
- **A DataWindow control or component** The control or component is a container for the DataWindow object in the application. You write code that calls methods of the container to manipulate the DataWindow object.

DataWindow controls and components

The DataWindow was originally invented for use in PowerBuilder to provide powerful data retrieval, manipulation, and update capabilities for client/server applications. Now the DataWindow is available in several environments:

- **PowerBuilder DataWindow** A PowerBuilder control for use in client/server and distributed PowerBuilder applications.

- **Web DataWindow** A thin-client DataWindow implementation for Web applications that provides most of the data manipulation, presentation, and scripting capabilities of the PowerBuilder DataWindow, requiring the Web DataWindow component on a component server but no PowerBuilder DLLs on the client. The Web DataWindow can be implemented in three ways:
  - **XML Web DataWindow** Separate XML (content), XSLT (layout), and CSS (style) with a subsequent transformation to XHTML
  - **XHTML Web DataWindow** XHTML
  - **HTML Web DataWindow** HTML
- **Web DataWindow DTC** A design-time control used to represent the Web DataWindow on a development machine. The DTC is an ActiveX control that generates a set of instructions for the DataWindow component on a component server. The component generates a Web DataWindow for display in a client browser. The client browser does not need to support ActiveX controls to display the Web DataWindow.
- **Sybase DataWindow Web control for ActiveX** An ActiveX control for use on Web pages. The client browser must support ActiveX controls to display a DataWindow object used by this control.
- **DataWindow plug-in** A browser plug-in that displays Powersoft reports.

For a comparison of each of these environments, see "Choosing a DataWindow technology" on page 7.

You can also use DataStore objects as containers for a DataWindow object. DataStores provide DataWindow functionality for retrieving and manipulating data without the on-screen display. Uses for DataStores include specifying layouts for printing and managing data in the server component of a distributed application.

What DataWindow objects are

A DataWindow object is an object that you use to retrieve, present, and manipulate data from a relational database or other data source (such as an Excel worksheet or dBASE file). You can specify whether the DataWindow object supports updating of data.

DataWindow objects have knowledge about the data they are retrieving. You can specify display formats, presentation styles, and other data properties to make the data meaningful to users.

In the DataWindow painter, you can also make Powersoft report (PSR) files, which you can use in DataWindow controls or components. A PSR file contains a report definition—essentially a nonupdatable DataWindow object—as well as the data contained in the report when the PSR file was created. It does not retrieve data.

Where to define DataWindow objects

You define DataWindow objects in the PowerBuilder DataWindow painter. You can also define nonupdatable DataWindow objects in the InfoMaker Report painter.

Presentation styles and data sources

When you define a DataWindow object, you choose a presentation style and a data source.

Presentation styles

A presentation style defines a typical style of report and handles how rows are grouped on the page. You can customize the way the data is displayed in each presentation style. The presentation styles include:

Table 1-1: DataWindow presentation styles

Presentation style	Description
Tabular	Data columns across the page and headers above each column. Several rows are viewable at once.
Freeform	Data columns going down the page with labels next to each column. One row displayed at a time.
Grid	Row-and-column format like a spreadsheet with grid lines. Users can move borders and columns.
Label	Several labels per page with one row for each label. Used for mailing and other labels.
N-Up	Two or more rows of data next to each other across the page. Useful for periodic data, such as data for each day of the week or each month in the quarter.
Group	A tabular style with rows grouped under headings. Each group can have summary fields with computed statistics.
TreeView	A tabular style that groups data hierarchically and displays the data in a way that is collapsible and expandable.
Composite	Several DataWindow objects grouped into a single presentation. Not supported by the Web DataWindow.
Graph	Graphical presentation of data. Not supported by the Web DataWindow.
Crosstab	Data summary in a row-and-column format.

Presentation style	Description
RichText	Paragraphs of text with embedded data columns. Not supported by the Web DataWindow or the Sybase DataWindow Web control for ActiveX.
OLE	An OLE object linked or embedded in the DataWindow and associated with the retrieved data. Not supported by the Web DataWindow.

For examples of the presentation styles, see the *PowerBuilder User's Guide*.

Data sources

The data source specifies where the data in the DataWindow comes from and what data items are displayed. Data can come from tables in a database, a Web service, a file with data that you can import, or code that specifies the data. For databases, the data specification is saved in a SQL statement. In all cases, the DataWindow object saves the names of the data items to display, as well as their datatypes.

Table 1-2: Data sources you can use for a DataWindow

Data source	Description
Quick Select	The data is coming from one or more tables in a SQL database. The tables must be related through a foreign key. You need to choose only columns, selection criteria, and sorting.
SQL Select	You want more control over the select statement that is generated for the data source. You can specify grouping, computed columns, and so on.
Query	The data has already been selected and the SQL statement is saved in a query object that you have defined in the Query painter. When you define the DataWindow object, the query object is incorporated into the DataWindow and does not need to be present when you run the application.
External	The data is not stored in a database, but is imported from a file (such as a tab-separated or dBASE file) or populated from code.
Stored Procedure	The data is defined in a database stored procedure.
Web Service	The data is defined in a Web service. Support for a Web service data source is not available for the Composite, RichText, and OLE presentation styles.

Basic process

Using a DataWindow involves two main steps:

1 Use the DataWindow painter to create or edit a DataWindow object.

In the painter, you define the data source, presentation style, and all other properties of the object, such as display formats, validation rules, sorting and filtering criteria, and graphs.

2 In your development environment, put a DataWindow control in a window, visual user object, or form or a DataWindow container in a Web page and associate a DataWindow object with the control or container.

It is through the control or container that your application communicates with the DataWindow object you created in the DataWindow painter. You write code to manipulate the DataWindow control or container and the DataWindow object it contains. Typically, your code retrieves and updates data, changes the appearance of the data, handles errors, and shares data between DataWindow controls.

Choosing a DataWindow technology

Since DataWindow technology can be used in different environments, it might not be obvious what approach you should take to implement your data-enabled application. This section describes the DataWindow technologies available for the basic application architectures and the requirements for each DataWindow solution.

The basic architectures are:

- **Client/server** A program running on a client workstation accesses a database running on a server. The user interface and business logic reside together on the client computer.
- **Distributed application** The user interface on the client computer calls components on a middle-tier server, which execute business logic and access the database server.

- **Web application** A client Web browser sends requests for HTML or JSP documents to a Web server. The Web server passes control to a page or application server, where server-side scripts can access components on a transaction server that can connect to databases on a database server.
- **.NET application** PowerBuilder lets you deploy DataWindows in .NET Web Forms and Windows Forms applications. For more information about .NET applications, see *Deploying Applications and Components to .NET*.

Solutions for client/server and distributed applications

The PowerBuilder DataWindow was initially developed for use in client/server applications.

You can implement the PowerBuilder DataWindow as a control that displays a DataWindow object or as a DataStore that supports data retrieval and update without displaying the data. A complete set of events and methods programmed in PowerScript provides control over all aspects of the DataWindow, including data retrieval, display, validation, and update.

You can also deploy the PowerBuilder DataWindow as a component for use in distributed applications.

For more information, see “PowerBuilder DataWindow control” on page 11.

Solutions for Web applications

You can use these DataWindow technologies in Web applications:

- Web DataWindow and the Web DataWindow DTC
- Sybase DataWindow Web control for ActiveX
- DataWindow plug-in
- .NET applications

Web DataWindow

The Web DataWindow is a thin-client DataWindow implementation for Web applications. It provides most of the data manipulation, presentation, and scripting capabilities of the PowerBuilder DataWindow without requiring any PowerBuilder DLLs on the client.

Functionality

The Web DataWindow supports a subset of the PowerBuilder DataWindow events and methods, including dynamic modification of the DataWindow object. The user can modify and update data. Composite, Graph, OLE, TreeView, and RichText presentation styles and controls are not supported.

Client requirements

The HTML Web DataWindow works in most browsers, but the appearance of the generated HTML is usually best in Internet Explorer. Generated HTML can be dynamically optimized for Netscape or Internet Explorer, or scaled back to handle older browsers. The XML Web DataWindow and the XHTML Web DataWindow require browsers that support the following client-side technologies: XML, XSLT, XHTML, CSS, and JavaScript. For information about supported browsers, see "Browser requirements for the XML Web DataWindow" on page 120.

Server requirements

A component server and a dynamic page server work together to generate a client control with data and include it in a Web page. Each time the user requests a new page of data, updates data, or inserts or deletes rows, the server gets a request to generate a new page. Depending on how state is managed, the component might retrieve data each time it is called, causing added load on the server.

Development tools

The Web Target interface in PowerBuilder provides the Web DataWindow DTC that you can use to develop Web DataWindows without hand coding. You can also write your own server-side scripts to create Web DataWindows using the component server's API. You can write the scripts for interpretation by a specific page server type (Active Server Pages or JavaServer Pages). For Web site targets, you can use the Web Target object model to write server scripts that are type-neutral—the scripts can be deployed to Active Server Pages.

When you use the Web DataWindow DTC on a 4GL Web page, server-side events are available and page variables can enhance the dynamic capabilities of your Web application. You cannot hand code a Web DataWindow on a 4GL Web page.

For information about the Web DataWindow, see Chapter 6, "Using the Web DataWindow." For information about the Web DataWindow DTC, see *Working with JSP Targets*.

Sybase DataWindow Web control for ActiveX

The Sybase DataWindow Web control for ActiveX is an interactive DataWindow control for use with Internet Explorer that implements all features of the PowerBuilder DataWindow except rich text.

Functionality	The Web ActiveX is fully programmable and supports DataWindow events, methods, and dynamic modification of the DataWindow object. The user can modify and update data. The RichText presentation style is not supported.
Client requirements	The control uses ActiveX technology and works in Microsoft Internet Explorer only. The user must download the CAB file for the component, which is less than two megabytes in size. Database connection through JDBC occurs from the client, which must be configured with the connection software. The software can be downloaded from the Web server. DataWindow behavior that would compromise security of the client, such as the SaveAs functionality, is disabled.
Server requirements	The JDBC database connection can access databases on a remote server.
Development tools	You can use the JSP target tools for working with ActiveX components to develop your Web application. For information, see Chapter 8, "Using the DataWindow Web Control for ActiveX."

DataWindow plug-in

The DataWindow plug-in is a browser plug-in that displays Powersoft reports (PSRs).

Functionality	A PSR is a static DataWindow; it cannot be modified and the data does not stay current. However, a PSR is a compact representation of the DataWindow definition and data and can be downloaded efficiently. It is an efficient means of viewing batch-produced reports. The user cannot modify or update the data.
Client requirements	The plug-in executable must be installed in the browser plug-in directory. It cannot be downloaded and installed automatically. The plug-in works in Netscape browsers. Internet Explorer browsers prior to version 5.5 also support the plug-in. DataWindow behavior that would compromise security of the client, such as SaveAs, is *not* disabled.
Development tools	You can use PowerBuilder or InfoMaker to create PSRs. For more information about the plug-in architecture, see *Application Techniques*.

.Net applications

For information, see *Deploying Applications and Components to .NET*.

PowerBuilder DataWindow control

Features

The PowerBuilder DataWindow control is a container for DataWindow objects in a PowerBuilder application. You can use it in a window to present an interactive display of data. The user can view and change data and send changes to the database.

In addition to the DataWindow control, the DataStore object provides a nonvisual container for server applications and other situations where on-screen viewing is not necessary.

The DataWindow supports data retrieval with retrieval arguments and data update. You can use edit styles, display formats, and validation rules for consistent data entry and display. The DataWindow provides many methods for manipulating the DataWindow, including Modify for changing DataWindow object properties. You can share a result set between several DataWindow controls and you can synchronize data between a client and server.

Development environment

You can develop both parts of your DataWindow implementation in PowerBuilder. You use:

- The DataWindow painter to define DataWindow objects.
- The Window or User Object painters to add DataWindow controls to windows or visual user objects. The DataWindow control is on the drop-down palette of controls for these painters.

  In the Window or User Object painters, you can write scripts that control the DataWindow's behavior and manipulate the data it retrieves. Your scripts can also instantiate DataStore objects.

In the PowerBuilder Browser you can examine the properties, events, and methods of DataWindow controls and DataStore objects on the System tab page. If you have a library open that contains DataWindow objects, you can examine the internal properties of the DataWindow object on the Browser's DataWindow tab page.

DataWindow objects

The DataWindow control or DataStore object uses a DataWindow object defined with any presentation style. The DataWindow object determines what data is retrieved and how it is displayed. The control can also display Powersoft reports (PSRs), which do not need to retrieve data.

Database connections

The PowerBuilder DataWindow can use ODBC, JDBC, and native database drivers for database connectivity. Users can connect to a data source on any server to which they have access, including databases and middle-tier servers on the Internet.

To make a connection, you can use the internal Transaction object of the DataWindow, or you can make the connection with a separate PowerBuilder transaction object.

A PowerBuilder application provides a default Transaction object, SQLCA. You can define additional Transaction objects if you need to make additional connections. When you connect with a separate Transaction object, you can control when SQL COMMIT and ROLLBACK statements occur, and you can use the same connection for multiple controls.

For more information about using a Transaction object with a DataWindow, see Chapter 2, "Using DataWindow Objects."

For more information about PowerBuilder Transaction objects, see *Application Techniques* in the PowerBuilder documentation set.

Coding

You write scripts in the Window or User Object painter to connect to the database, retrieve data, process user input, and update data.

In PowerBuilder, you can take advantage of object inheritance by defining a user object inherited from a DataWindow control and adding your own custom functionality. You can reuse the customized DataWindow control throughout your applications.

You create DataStore objects, the nonvisual version of a DataWindow control, by creating them in a script and calling methods for the object. You can also define a user object that is inherited from a DataStore and customize it. For more information, see Chapter 4, "Using DataStore Objects."

Libraries and applications

You store DataWindow objects in PowerBuilder libraries (PBLs) during development. When you build your application, you can include the DataWindow objects in the application executable or in PowerBuilder dynamic libraries (PBDs).

For more information about designing DataWindow objects and building a PowerBuilder application, see the *PowerBuilder User's Guide* and *Application Techniques*.

CHAPTER 2

Using DataWindow Objects

About this chapter

This chapter describes how to use DataWindow objects in an application.

Contents

Topic	Page
About using DataWindow objects	13
Putting a DataWindow object into a control	14
Accessing the database	21
Accessing a Web service data source	29
Importing data from an external source	29
Manipulating data in a DataWindow control	30
Accessing the properties of a DataWindow object	37
Handling DataWindow errors	39
Updating the database	44
Creating reports	47
Using nested reports	49
Using crosstabs	52
Generating HTML	55

Before you begin

This chapter assumes that you know how to build DataWindow objects in the DataWindow painter, as described in the *PowerBuilder User's Guide*.

About using DataWindow objects

Building DataWindow objects

Before you can use a DataWindow object in an application, you need to build it. PowerBuilder has separate painters for database management, DataWindow definition, and library management.

You define and edit a DataWindow object in the DataWindow painter. You specify its data source and presentation style, then enhance the object by specifying display formats, edit styles, and more.

The DataWindow painter is also where you make Powersoft report (PSR) files, which you might also want to use in applications. A PSR file contains a report definition—essentially a nonupdatable DataWindow object—as well as the data contained in that report when the PSR file was created.

Report objects only in InfoMaker
Older versions of PowerBuilder had a Report painter as well as a DataWindow painter. A report object could retrieve but not update data; it was essentially a nonupdatable DataWindow object. The Report painter is now available only in InfoMaker.

Managing DataWindow objects

Several painters let you manage and package your DataWindow objects for use in applications.

In particular, you can maintain DataWindow objects in one or more libraries (PBL files). When you are ready to use your DataWindow objects in applications, you can package them in more compact runtime libraries (PBD files).

For further details on how to build and organize DataWindow objects, see the *PowerBuilder User's Guide*.

Using DataWindow objects

After you build a DataWindow object (or PSR file) in the DataWindow painter, you can use it to display and process information from the appropriate data source. The sections that follow explore the details of how to do this.

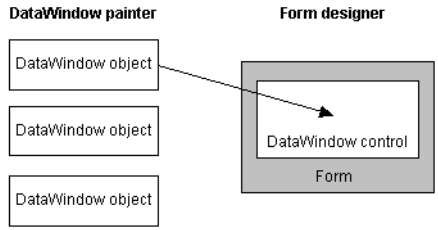
Putting a DataWindow object into a control

The DataWindow control is a container for DataWindow objects in an application. It provides properties, methods, and events for manipulating the data and appearance of the DataWindow object. The DataWindow control is part of the user interface of your application.

You also use DataWindow objects in the nonvisual DataStore and in child DataWindows, such as drop-down DataWindows and composite presentation styles. For more information about DataStores, see Chapter 4, "Using DataStore Objects." For more information about drop-down DataWindows and composite DataWindows, see the *PowerBuilder User's Guide*.

To use the DataWindow object in an application, you add a DataWindow control to a window or form, then associate that control with the DataWindow object, as illustrated in Figure 2-1:

Figure 2-1: Putting a DataWindow object into a DataWindow control



This section has information about:

- Names for DataWindow controls and DataWindow objects
- Procedures for inserting a control and assigning a DataWindow object to the control
- Specifying the DataWindow object during execution

For information about assigning a DataWindow object to a Web DataWindow control, see *Working with JSP Targets* or "Loading the DataWindow object" on page 185.

For information about assigning a DataWindow object to a Web control for ActiveX, see "Specifying a DataWindow object for the control" on page 222.

Names for DataWindow controls and DataWindow objects

There are two names to be aware of when you are working with a DataWindow:

- The name of the DataWindow control
- The name of the DataWindow object associated with the control

The DataWindow control name When you place a DataWindow control in a window or form, it gets a default name. You should change the name to be something meaningful for your application.

In PowerBuilder, the name of the control has traditionally had a prefix of dw_. This is a useful convention to observe in any development environment. For example, if the DataWindow control lists customers, you might want to name it dw_customer.

Using the name
In code, always refer to a DataWindow by the name of the *control* (such as dw_customer). Do not refer to the DataWindow *object* that is in the control.

The DataWindow object name To avoid confusion, you should use different prefixes for DataWindow objects and DataWindow controls. The prefix d_ is commonly used for DataWindow objects. For example, if the name of the DataWindow control is dw_customer, you might want to name the corresponding DataWindow object d_customer.

Working with the DataWindow control in PowerBuilder

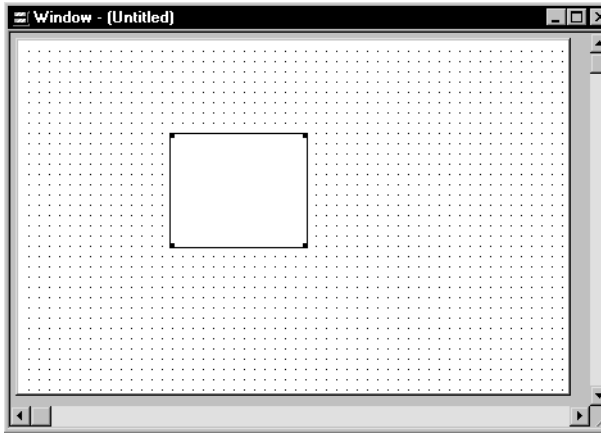
❖ **To place a DataWindow control in a window:**

1 Open the window that will contain the DataWindow control.

2 Select Insert>Control>DataWindow from the menu bar.

3 Click where you want the control to display.

PowerBuilder places an empty DataWindow control in the window:



4 (Optional) Resize the DataWindow control by selecting it and dragging one of the handles.

Specifying a DataWindow object

After placing the DataWindow control, you associate a DataWindow object with the control.

❖ **To associate a DataWindow object with the control:**

1. In the DataWindow Properties view, click the Browse button for the DataObject property.
2. Select the DataWindow object that you want to place in the control and click OK.

   The name of the DataWindow object displays in the DataObject box in the DataWindow Properties view.
3. (Optional) Change the properties of the DataWindow control as needed.

Allowing users to move DataWindow controls
If you want users to be able to move a DataWindow control during execution, give it a title and select the Title Bar check box. Then users can move the control by dragging the title bar.

Defining reusable DataWindow controls

You might want all the DataWindow controls in your application to have similar appearance and behavior. For example, you might want all of them to do the same error handling.

To be able to define these behaviors once and reuse them in each window, you should create a standard user object based on the DataWindow control: define the user object's properties and write scripts that perform the generic processing you want, such as error handling. Then place the user object (instead of a new DataWindow control) in the window. The DataWindow user object has all the desired functionality predefined. You do not need to respecify it.

For more information about creating and using user objects, see the PowerBuilder *User's Guide*.

Editing the DataWindow object in the control

Once you have associated a DataWindow object with a DataWindow control in a window, you can go directly to the DataWindow painter to edit the associated DataWindow object.

❖ **To edit an associated DataWindow object:**

- Select Modify DataWindow from the DataWindow control's pop-up menu.

  PowerBuilder opens the associated DataWindow object in the DataWindow painter.

Specifying the DataWindow object during execution

Changing the DataWindow object

The way to change the DataWindow object depends on the environment:

- **PowerBuilder** Set the DataObject property to one of the DataWindow objects built into the application.
- **Web ActiveX** Set the SourceFileName and DataWindowObject properties to select a new library file and DataWindow.
- **Web DataWindow** If you are not using the Web Target object model, you can call the SetDWObject method on the Web DataWindow generator component.

Setting the transaction object when you change the DataWindow object
When you change the DataWindow object during execution, you might need to call setTrans or setTransObject again.

For more information, see "Setting the transaction object for the DataWindow control" on page 21.

Dynamically creating a DataWindow object

You can also create a new DataWindow object during execution and associate it with a control.

For more information, see Chapter 3, "Dynamically Changing DataWindow Objects."

Changing the DataWindow in PowerBuilder

When you associate a DataWindow object with a control in the window, you are setting the initial value of the DataWindow control's DataObject property.

During execution, this tells your application to create an instance of the DataWindow object specified in the control's DataObject property and use it in the control.

Setting the DataObject property in code

In addition to specifying the DataWindow object in the Window painter, you can switch the object that displays in the control during execution by changing the value of the DataObject property in code.

For example: to display the DataWindow object d_emp_hist from the library *emp.pbl* in the DataWindow control dw_emp, you can code:

```
dw_emp.DataObject = "d_emp_hist"
```

The DataWindow object d_emp_hist was created in the DataWindow painter and stored in a library on the application search path. The control dw_emp is contained in the window and is saved as part of the window definition.

Preventing redrawing

You can use the SetRedraw method to turn off redrawing in order to avoid flicker and reduce redrawing time when you are making several changes to the properties of an object or control. Dynamically changing the DataWindow object at execution time implicitly turns redrawing on. To turn redrawing off again, call the SetRedraw method every time you change the DataWindow object:

```
dw_emp.DataObject = "d_emp_hist"
dw_emp.SetRedraw(FALSE)
```

Using PSR files

To put a PSR file into a DataWindow control at execution time, change the control's DataObject property to specify that PSR file name.

Changing the DataWindow in the Web ActiveX

When you associate a DataWindow object with a DataWindow control, you are setting the initial value of the DataWindow control's SourceFileName and DataWindowObject properties.

During execution, this tells your application to:

1 Look for DataWindow objects in the library (PBL file) or runtime library (PBD file) specified in the control's SourceFileName property.

2 Create an instance of the DataWindow object specified in the control's DataWindowObject property (which must be in the specified library) and use it in the control.

Setting the SourceFileName and DataWindowObject properties in code

In addition to specifying the DataWindow object in the Window painter, you can switch the object that displays in the control during execution by changing the value of the SourceFileName and DataWindowObject properties in code.

You might simply change the DataWindowObject property to use a different DataWindow object from the same library, or you might change both properties to use a DataWindow object from some other library.

For information about URLs for SourceFileName, see “Specifying a DataWindow object for the control” on page 222.

For more information about the SourceFileName and DataWindowObject properties, see the *DataWindow Reference*.

Using PSR files If you want to dynamically put a PSR file into a DataWindow control at execution time, change the control's SourceFileName property to an empty string and specify a URL for the PSR file as the value for the DataWindowObject property.

Examples

This example shows the code to set the properties in JavaScript. The code changes the DataWindow object in dw_emp, a DataWindow control in a form or Web page. Dw_emp is saved as part of the form or Web page definition. The value for DataWindowObject is d_emp_hist; it was created in the DataWindow painter and is stored in the library named emp.pbl, the value for SourceFileName.

Web ActiveX For the Web ActiveX on a Web page, you set the SourceFileName and DataWindowObject properties directly.

To display the DataWindow object d_emp_hist from the library emp.pbl in the DataWindow control dw_emp, you can code:

```
dw_emp.SourceFileName = "dwlibs/emp.pbl";
dw_emp.DataWindowObject = "d_emp_hist";
```

Accessing the database

Before you can display data in a DataWindow control, you must get the data stored in the data source into that control. The most common way to get the data is to access a database.

An application goes through several steps in accessing a database:

1 Set the appropriate values for the transaction object.

2 Connect to the database.

3 Set the transaction object for the DataWindow control.

4 Retrieve and update data.

5 Disconnect from the database.

This section provides instructions for setting the transaction object for a DataWindow control and for using the DataWindow object to retrieve and update data.

To learn more about setting values for the transaction object, connecting to the database, and disconnecting from the database, see:

- **PowerBuilder** *Application Techniques*, "Using Transaction Objects"
- **Web DataWindow** "Specifying the database connection and retrieving data" on page 187
- **Web ActiveX** "Using the DataWindow Transaction Object control" on page 224

Setting the transaction object for the DataWindow control

There are two ways to handle database connections and transactions for the DataWindow control. You can use:

- Internal transaction management
- A separate transaction object

The two methods provide different levels of control over database transactions.

If you are displaying a PSR file in the control
You do not need to use a transaction object or make a database connection if you are displaying a PSR file in the DataWindow control.

If you change the DataWindow object
If you change the DataWindow object associated with a DataWindow control during execution, you might need to call the SetTrans or SetTransObject method again.

PowerBuilder You always need to call one of the methods to set the transaction object.

Web ActiveX You need to call SetTransObject again only when you are using a separate transaction object.

These options are described in this section.

Internal transaction management

What it does

When the DataWindow control uses internal transaction management, it handles connecting, disconnecting, commits, and rollbacks. It *automatically* performs connects and disconnects as needed; any errors that occur cause an *automatic* rollback.

Whenever the DataWindow needs to access the database (such as when a Retrieve or Update method is executed), the DataWindow issues an internal CONNECT statement, does the appropriate data access, then issues an internal DISCONNECT.

Whether to use it

When not to use it Do not use internal transaction management when:

- Your application requires the best possible performance

  Internal transaction management is slow and uses considerable system resources because it must connect and disconnect for every database access.

- You want control over when a transaction is committed or rolled back

  Because internal transaction management must disconnect after a database access, any changes are always committed immediately.

When to use it If the number of available connections at your site is limited, you might want to use internal transaction management because connections are not held open.

Internal transaction management is appropriate in simple situations when you are doing pure retrievals (such as in reporting) and do not need to hold database locks—when application control over committing or rolling back transactions is not an issue.

How it works

PowerBuilder To use internal transaction management, you specify connection values for a transaction object, which could be the automatically instantiated SQLCA. Then you call the SetTrans method, which copies the values from a specified transaction object to the DataWindow control's internal transaction object.

```
SQLCA.DBMS = ProfileString("myapp.ini", &
       "database", "DBMS", " ")
... // Set more connection parameters
dw_employee.SetTrans(SQLCA)
dw_employee.Retrieve( )
```

Connecting to the database
When you use SetTrans, you do not need to explicitly code a CONNECT or DISCONNECT statement in a script. CONNECT and DISCONNECT statements are automatically issued when needed.

For more information about PowerBuilder transaction objects, see PowerBuilder *Application Techniques*.

Web ActiveX To use internal transaction management, set the transaction properties for the DataWindow Web ActiveX control instead of using a DataWindow Transaction Object control. You can set the properties using Param elements or in a script. This example sets the DbParm property and calls Retrieve in a script:

```
dw_employee.DbParm =
       "Driver='com.sybase.jdbc.SybDriver',
       URL='jdbc:sybase:Tds:www.domain.com:7373'";
dw_employee.Retrieve( );
```

For internal transaction management, you do not call SetTransObject. If you change the DataWindow object during execution, the connection information is still available and the DataWindow connects as needed. You can change the connection information by changing the value of the DbParm property.

Transaction management with a separate transaction object

How it works

When you use a separate transaction object, you control the duration of the database transaction. Your scripts explicitly connect to and disconnect from the database. If the transaction object's AutoCommit property is set to *false*, you also program when an update is committed or rolled back.

Typically, a script for data retrieval or update involves these statements:

Connect
SetTransObject
Retrieve or Update
Commit or Rollback
Disconnect

In PowerBuilder, you use embedded SQL for connecting and committing. For the Web ActiveX, the transaction object has methods that perform these actions.

The transaction object also stores error messages returned from the database in its properties. You can use the error information to determine whether to commit or roll back database changes.

When to use it

When the DataWindow control uses a separate transaction object, you have more control of the database processing and are responsible for managing the database transaction.

There are several reasons to use a separate transaction object:

- You have several DataWindow controls that connect to the same database and you want to make one database connection for all of them, saving the overhead of multiple connections
- You want to control transaction processing
- You require the improved performance provided by keeping database connections open

How it works

PowerBuilder The SetTransObject method associates a transaction object with the DataWindow control. PowerBuilder has a default transaction object called SQLCA that is automatically instantiated. You can set its connection properties, connect, and assign it to the DataWindow control.

The following statement uses SetTransObject to associate the DataWindow control dw_emp with the default transaction object (SQLCA):

```
// Set connection parameters in the transaction object
SQLCA.DBMS = ...
SQLCA.database = ...
CONNECT USING SQLCA;
dw_emp.SetTransObject(SQLCA)
dw_emp.Retrieve( )
```

Instead of or in addition to using the predefined SQLCA transaction object, you can define your own transaction object in a script. This is necessary if your application needs to connect to more than one database at the same time.

The following statement uses SetTransObject to associate dw_customer with a programmer-created transaction object (trans_customer):

```
transaction trans_customer
trans_customer = CREATE transaction
// Set connection parameters in the transaction object
trans_customer.DBMS = ...
trans_customer.database = ...
CONNECT USING trans_customer;
dw_customer.SetTransObject(trans_customer)
dw_customer.Retrieve( )
```

Web ActiveX To use a separate transaction object for the Web ActiveX, you add an OBJECT element for the Sybase DataWindow Transaction Object control to the Web page. You can set its connection properties using Param elements or a script.

A script that connects and retrieves data would have statements like these:

```
trans_1.Connect( );
dw_employee.SetTransObject( trans_1 );
dw_employee.Retrieve( );
trans_1.Disconnect( );
```

For more information

For more information about database transaction processing:

- **PowerBuilder** See the chapter on using transaction objects in *Application Techniques*
- **Web ActiveX** See Chapter 8, “Using the DataWindow Web Control for ActiveX”

For more information about SetTrans and SetTransObject methods, see the *DataWindow Reference*.

Retrieving and updating data

You call the following two methods to access a database through a DataWindow control:

Retrieve
Update

Basic data retrieval

After you have set the transaction object for your DataWindow control, you can use the Retrieve method to retrieve data from the database into that control:

```
dw_emp.Retrieve( )
```

The Web DataWindow server component has a second form of the method, RetrieveEx, for use when the method requires arguments. For more information about retrieving data with the Web DataWindow, see "Specifying the database connection and retrieving data" on page 187 and "Passing page-specific data to the reloaded page" on page 189.

Using retrieval arguments

About retrieval arguments

Retrieval arguments qualify the SELECT statement associated with the DataWindow object, reducing the rows retrieved according to some criteria. For example, in the following SELECT statement, Salary is a retrieval argument defined in the DataWindow painter:

```
SELECT Name, emp.sal FROM Employee
       WHERE emp.sal > :Salary
```

When you call the Retrieve method, you supply a value for Salary. In PowerBuilder, the code looks like this:

```
dw_emp.Retrieve( 50000 )
```

Special considerations for each environment are explained below.

When coding Retrieve with arguments, specify them in the order in which they are defined in the DataWindow object. Your Retrieve method can provide more arguments than a particular DataWindow object expects. Any extra arguments are ignored. This allows you to write a generic Retrieve that works with several different DataWindow objects.

Omitting retrieval arguments If your DataWindow object takes retrieval arguments but you do not pass them in the Retrieve method, the DataWindow control prompts the user for them when Retrieve is called.

More than 16 arguments

The Retrieve method is limited to 16 arguments in some environments.

PowerBuilder You can specify any number of retrieval arguments.

Web DataWindow You can specify a maximum of 16 arguments using the RetrieveEx method.

Web ActiveX You can specify a maximum of 16 arguments for Retrieve. If you need to specify more, use the RetrieveEx method for the Web ActiveX and pass an array where each array element is a retrieval argument.

Updating data

After users have made changes to data in a DataWindow control, you can use the Update method to save those changes in the database.

In PowerBuilder, the code looks like this:

```
dw_emp.Update()
```

Update sends to the database all inserts, changes, and deletions made in the DataWindow control since the last Update method. When you are using an external transaction object, you can then commit (or roll back) those database updates. In PowerBuilder, you use SQL statements. In the Web ActiveX, you use methods and properties of the transaction object. In the Web DataWindow client control, update requests call the update method in the server component, which handles the commit or rollback.

For more specifics on how a DataWindow control updates the database (that is, which SQL statements are sent in which situations), see “Updating the database” on page 44.

Examples

The following example shows code that connects, retrieves, updates, commits or rolls back, and disconnects from the database.

Although the example shows all database operations in a single script or function, most applications separate these operations. In a PowerBuilder application, for example, an application could connect to the database in the application Open event, retrieve and update data in one or more window scripts, and disconnect from the database in the application Close event.

PowerBuilder The following statements retrieve and update data using the transaction object EmpSQL and the DataWindow control dw_emp:

```
// Connect to the database specified in the
// transaction object EmpSQL
CONNECT USING EmpSQL;
```

```
// Set EmpSQL as the transaction object for dw_emp
dw_emp.SetTransObject(EmpSQL)

// Retrieve data from the database specified in
// EmpSQL into dw_emp
dw_emp.Retrieve()

// Make changes to the data...
...

// Update the database
IF dw_emp.Update() > 0 THEN
       COMMIT USING EmpSQL;
ELSE
       ROLLBACK USING EmpSQL;
END IF

// Disconnect from the database
DISCONNECT USING EmpSQL;
```

Web ActiveX The following JavaScript statements retrieve and update data using the transaction object EmpSQL and the DataWindow control dw_emp.

```
// Connect to the database specified in the
// transaction object EmpSQL
EmpSQL.Connect( );

// Set EmpSQL as the transaction object for dw_emp
dw_emp.SetTransObject(EmpSQL);

// Retrieve data from the database specified in
// EmpSQL into dw_emp
dw_emp.Retrieve();

// Make changes to the data
...

// Update the database
if (dw_emp.Update() > 0) {
       EmpSQL.Commit( );
} else {
       EmpSQL.Rollback( );
}

// Disconnect from the database
EmpSQL.Disconnect( );
```

Handling retrieval or update errors

A production application should include error tests after each database operation.

For more about checking for errors, see “Handling DataWindow errors” on page 39.

Accessing a Web service data source

You do not use a transaction object to access data from a Web service data source. However, some Web services support or require a user ID and password, and other session-related properties like firewall settings. The WSConnection object can provide this information for your DataWindow connections.

You use an instance of the WSConnection object to connect to a Web service by calling the SetWSObject method.

For more information about setting properties for a Web service connection, see WSConnection and SetWSObject in the online Help.

Importing data from an external source

PowerBuilder and Web ActiveX

If the data for a DataWindow is not coming from a database or a Web service data source (that is, the data source was defined as External in the DataWindow wizard), you can use these methods to import data into the DataWindow control:

ImportClipboard
ImportFile
ImportString

All environments

You can also get data into the DataWindow by using the SetItem method or by using a DataWindow expression.

For more information on the SetItem method and DataWindow expressions, see "Manipulating data in a DataWindow control" next.

Manipulating data in a DataWindow control

To handle user requests to add, modify, and delete data in a DataWindow, you can write code to process that data, but first you need to understand how DataWindow controls manage data.

How a DataWindow control manages data

As users add or change data, the data is first handled as text in an edit control. If the data is accepted, it is then stored as an item in a buffer.

About the DataWindow buffers

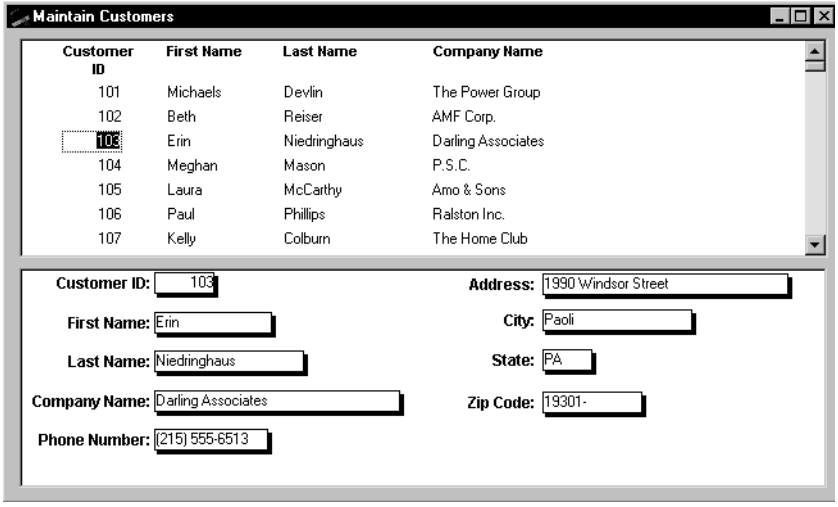
A DataWindow uses three buffers to store data:

Table 2-1: DataWindow buffers

Buffer	Contents
Primary	Data that has not been deleted or filtered out (that is, the rows that are viewable)
Filter	Data that was filtered out
Delete	Data that was deleted by the user or through code

About the edit control

As the user moves around the DataWindow control, the DataWindow places an edit control over the current cell (row and column):



About text

The contents of the edit control are called text. Text is data that has not yet been accepted by the DataWindow control. Data entered in the edit control is not in a DataWindow buffer yet; it is simply text in the edit control.

About items

When the user changes the contents of the edit control and presses ENTER or leaves the cell (by tabbing, using the mouse, or pressing UP ARROW or DOWN ARROW), the DataWindow processes the data and either accepts or rejects it, depending on whether it meets the requirements specified for the column. If the data is accepted, the text is moved to the current row and column in the DataWindow Primary buffer. The data in the Primary buffer for a particular column is referred to as an item.

Events for changing text and items

When data is changed in the edit control, several events occur. The names of the events are different in each environment, as shown in the table. This chapter refers to events using PowerBuilder names.

Table 2-2: Event names in different environments

Event		Description
For PowerBuilder, Web DataWindow client control	**For Web ActiveX**	
EditChanged (not available on client control)	onEditChange	Occurs for each keystroke the user types in the edit control
ItemChanged	beforeItemChange	Occurs when a cell has been modified and loses focus
ItemError	onItemError	Occurs when new data fails the validation rules for the column
ItemFocusChanged	onItemFocusChange	Occurs when the current item in the control changes

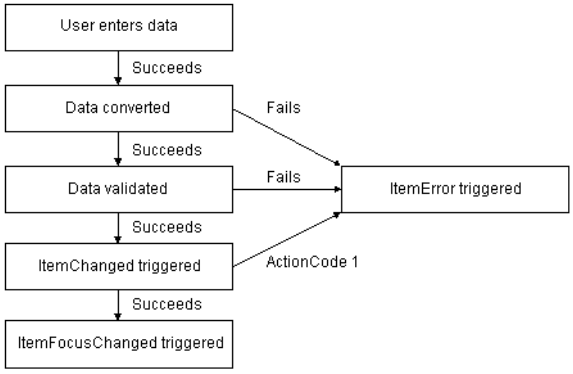
How text is processed in the edit control

When the data in a column in a DataWindow has been changed and the column loses focus (for example, because the user tabs to the next column), the following sequence of events occurs:

1. The DataWindow control converts the text into the correct datatype for the column. For example, if the user is in a numeric column, the DataWindow control converts the string that was entered into a number. If the data cannot be converted, the ItemError event is triggered.
2. If the data converts successfully to the correct type, the DataWindow control applies any validation rule used by the column. If the data fails validation, the ItemError event is triggered.
3. If the data passes validation, then the ItemChanged event is triggered. If you set an action/return code of 1 in the ItemChanged event, the DataWindow control rejects the data and does not allow the focus to change. In this case, the ItemError event is triggered.

4 If the ItemChanged event accepts the data, the ItemFocusChanged event is triggered next and the data is stored as an item in a buffer.

Figure 2-2: How text is processed in edit controls



Action/return codes for events

You can affect the outcome of events by specifying numeric values in the event's program code. For example, step 3 above describes how you can force data to be rejected with a code of 1 in the ItemChanged event.

To specify action/return codes:

- **PowerBuilder and Web DataWindow** Use a RETURN statement
- **Web ActiveX** Call the SetActionCode or setActionCode method

For information about codes for individual events, see the *DataWindow Reference*.

Accessing the text in the edit control

Using methods

The following methods allow you to access the text in the edit control:

- GetText—Obtains the text in the edit control
- SetText—Sets the text in the edit control

In event code

In addition to these methods, the following events provide access to the text in the edit control:

EditChanged
ItemChanged
ItemError

Use the Data parameter, which is passed into the event, to access the text of the edit control. In your code for these events, you can test the text value and perform special processing depending on that value.

For an example, see “Coding the ItemChanged event” on page 33.

Manipulating the text in the edit control

When you want to further manipulate the contents of the edit control within your DataWindow control, you can use any of these methods:

CanUndo	Scroll
Clear	SelectedLength
Copy	SelectedLine
Cut	SelectedStart
LineCount	SelectedText
Paste	SelectText
Position	TextLine
ReplaceText	Undo

For more information about these methods, see the *DataWindow Reference*.

Coding the ItemChanged event

If data passes conversion and validation, the ItemChanged event is triggered. By default, the ItemChanged event accepts the data value and allows focus to change. You can write code for the ItemChanged event to do some additional processing. For example, you could perform some tests, set a code to reject the data, have the column regain focus, and trigger the ItemError event.

Example

The following sample code for the ItemChanged event for a DataWindow control called dw_Employee sets the return code in dw_Employee to reject data that is less than the employee's age, which is specified in a SingleLineEdit text box control in the window.

This is the PowerBuilder version of the code:

```
int a, age
age = Integer(sle_age.text)
a = Integer(data)

// Set the return code to 1 in the ItemChanged
// event to tell PowerBuilder to reject the data
// and not change the focus.
IF a < age THEN RETURN 1
```

Coding the ItemError event

The ItemError event is triggered if there is a problem with the data. By default, it rejects the data value and displays a message box. You can write code for the ItemError event to do some other processing. For example, you can set a code to accept the data value, or reject the data value but allow focus to change.

For more information about the events of the DataWindow control, see the *DataWindow Reference*.

Accessing the items in a DataWindow

You can access data values in a DataWindow by using methods or DataWindow data expressions. Both methods allow you to access data in any buffer and to get original or current values.

The method you use depends on how much data you are accessing and whether you know the names of the DataWindow columns when the script is compiled.

For guidelines on deciding which method to use, see the *DataWindow Reference*.

Using methods

There are several methods for manipulating data in a DataWindow control.

These methods obtain the data in a specified row and column in a specified buffer (Web DataWindow methods have separate methods for overloaded versions):

- **PowerBuilder** GetItemDate, GetItemDateTime, GetItemDecimal, GetItemNumber, GetItemString, GetItemTime
- **Web ActiveX** GetItemDate, GetItemNumber, GetItemString

- **Web DataWindow server component** GetItemDate, GetItemDateByColNum, GetItemDateByColNumEx, GetItemDateEx, GetItemDateTime, GetItemDateTimeByColNum, GetItemDateTimeByColNumEx, GetItemDateTimeEx, GetItemNumber, GetItemNumberByColNum, GetItemNumberByColNumEx, GetItemNumberEx, GetItemStatus, GetItemStatusByColNum, GetItemString, GetItemStringByColNum, GetItemStringByColNumEx, GetItemStringEx, GetItemTime, GetItemTimeByColNum, GetItemTimeByColNumEx, GetItemTimeEx

This method sets the value of a specified row and column:

- **PowerBuilder and Web ActiveX** SetItem
- **Web DataWindow server component** SetItemDate, SetItemDateByColNum, SetItemDateTime, SetItemDateTimeByColNum, SetItemNumber, SetItemNumberByColNum, SetItemStatus, SetItemStatusByColNum, SetItemString, SetItemStringByColNum, SetItemTime, SetItemTimeByColNum

For example, the following statement, using PowerBuilder syntax, assigns the value from the empname column of the first row to the variable ls_Name in the Primary buffer:

```
ls_Name = dw_1.GetItemString (1, "empname")
```

This PowerBuilder statement sets the value of the empname column in the first row to the string Waters:

```
dw_1.SetItem(1, "empname", "Waters")
```

Uses You call the GetItem methods to obtain the data that has been accepted into a specific row and column. You can also use them to check the data in a specific buffer before you update the database. You must use the method appropriate for the column's datatype.

For more information about the methods listed above, see the *DataWindow Reference*.

Using expressions

DataWindow data expressions refer to single items, columns, blocks of data, selected data, or the whole DataWindow.

The way you construct data expressions depends on the environment:

- **PowerBuilder** Use dot notation
- **Web ActiveX** Data expressions are not supported

Expressions in PowerBuilder The Object property of the DataWindow control lets you specify expressions that refer directly to the data of the DataWindow object in the control. This direct data manipulation allows you to access small and large amounts of data in a single statement, without calling methods:

```
dw_1.Object.jobtitle[3] = "Programmer"
```

The next statement sets the value of the first column in the first row in the DataWindow to Smith:

```
dw_1.Object.Data[1,1] = "Smith"
```

For complete instructions on how to construct DataWindow data expressions, see the *DataWindow Reference*.

Using other DataWindow methods

There are many more methods you can use to perform activities in DataWindow controls. Here are some of the more common ones:

Table 2-3: Common methods in DataWindow controls

Method	Purpose
AcceptText	Applies the contents of the edit control to the current item in the DataWindow control
DeleteRow	Removes the specified row from the DataWindow control, placing it in the Delete buffer; does not delete the row from the database
Filter	Displays rows in the DataWindow control based on the current filter
GetRow	Returns the current row number
InsertRow	Inserts a new row
Reset	Clears all rows in the DataWindow control
Retrieve	Retrieves rows from the database
RowsCopy, RowsMove	Copies or moves rows from one DataWindow control to another
ScrollToRow	Scrolls to the specified row
SelectRow	Highlights a specified row
ShareData	Shares data among different DataWindow controls.
Update	Sends to the database all inserts, changes, and deletions that have been made in the DataWindow control

Some, but not all, of these methods are available for the Web DataWindow client control, server component, or both. Each development environment provides a reference list of methods. For a complete list of DataWindow methods, see the PowerBuilder Browser for PowerScript targets or the Components tab of the System Tree for JSP targets.

For complete information on DataWindow methods, see the *DataWindow Reference*.

Accessing the properties of a DataWindow object

About DataWindow object properties

DataWindow object properties store the information that controls the behavior of a DataWindow object. They are not properties of the DataWindow control, but of the DataWindow object displayed in the control. The DataWindow object is itself made up of individual controls—column, text, graph, and drawing controls—that have DataWindow object properties.

You establish initial values for DataWindow object properties in the DataWindow painter. You can also get and set property values during execution in your code.

You can access the properties of a DataWindow object by using the Describe and Modify methods or DataWindow property expressions. Which you use depends on the type of error checking you want to provide and on whether you know the names of the controls within the DataWindow object and properties you want to access when the script is compiled.

Note that in the Web ActiveX, only the Describe and Modify methods (not property expressions) are supported.

For guidelines on deciding which method to use and for lists and descriptions of DataWindow object properties, see the *DataWindow Reference*.

Using methods to access object properties

You can use the following methods to work with the properties of a DataWindow object:

- Describe—Reports the values of properties of a DataWindow object and controls within the DataWindow object
- Modify—Modifies a DataWindow object by specifying a list of instructions that change the DataWindow object's definition

PowerBuilder For example, the following statements assign the value of the Border property for the empname column to a string variable:

```
string ls_border
ls_border = dw_1.Describe("empname.Border")
```

The following statement changes the value of the Border property for the empname column to 1:

```
dw_emp.Modify("empname.Border=1")
```

Web ActiveX The JavaScript code is nearly identical to PowerScript. These statements get the value of the Border property for the empname column:

```
string ls_border
ls_border = dw_1.Describe("empname.Border");
```

The following statement changes the value of the Border property for the empname column to 1:

```
dw_emp.Modify("empname.Border=1");
```

About dynamic DataWindow objects

Using Describe and Modify, you can provide an interface through which application users can alter the DataWindow object during execution. For example, you can change the appearance of a DataWindow object or allow an application user to create ad hoc reports. For more information, see Chapter 3, "Dynamically Changing DataWindow Objects."

Using expressions

DataWindow property expressions provide access to properties with fewer nested strings. In PowerBuilder, you can handle problems with incorrect object and property names in the Error event:

PowerBuilder Use the Object property and dot notation. For example:

```
integer li_border
li_border = Integer(dw_1.Object.empname.Border)
dw_1.Object.empname.Border = 1
```

For reference material on the available variations for property expressions, see the *DataWindow Reference*.

Handling DataWindow errors

There are several types of errors that can occur during DataWindow processing:

- Data items that are invalid (discussed in "Manipulating data in a DataWindow control" on page 30)
- Failures when retrieving or updating data
- Attempts to access invalid or nonexistent properties or data

This section explains how to handle the last two types of errors.

Retrieve and Update errors and the DBError event

Retrieve and update testing

When using the Retrieve or Update method in a DataWindow control, you should test the method's return code to see whether the activity succeeded.

Do not test the SQLCode attribute

After issuing a SQL statement (such as CONNECT, COMMIT, or DISCONNECT) or the equivalent method of the transaction object, you should always test the success/failure code (the SQLCode attribute in the transaction object). However, you should *not* use this type of error checking following a retrieval or update made in a DataWindow.

For more information about error handling after a SQL statement, see:

- **PowerBuilder** The chapter on using transaction objects in *Application Techniques*
- **Web ActiveX** Chapter 8, "Using the DataWindow Web Control for ActiveX"

Table 2-4: Return codes for the Retrieve and Update methods

Method	Return code	Meaning
Retrieve	>=1	Retrieval succeeded; returns the number of rows retrieved.
	-1	Retrieval failed; DBError event triggered.
	0	No data retrieved.
Update	1	Update succeeded.
	-1	Update failed; DBError event triggered.

Example

PowerBuilder If you want to commit changes to the database only if an update succeeds, you can code:

```
IF dw_emp.Update() > 0 THEN
       COMMIT USING EmpSQL;
ELSE
       ROLLBACK USING EmpSQL;
END IF
```

Web ActiveX To commit changes to the database only if an update succeeds, you can code:

```
number rtn;
rtn = dw_emp.Update( );
if (rtn == 1) {
       trans_a.Commit( );
} else {
       trans_a.Rollback( );
}
```

Using the DBError event

The DataWindow control triggers its DBError event whenever there is an error following a retrieval or update; that is, if the Retrieve or Update methods return –1. For example, if you try to insert a row that does not have values for all columns that have been defined as not allowing NULL, the DBMS rejects the row and the DBError event is triggered.

By default, the DataWindow control displays a message box describing the error message from the DBMS, as shown here:

In many cases you might want to code your own processing in the DBError event and suppress the default message box. Here are some tips for doing this:

Table 2-5: Tips for processing messages from DBError event

To	Do this
Get the DBMS's error code	Use the SQLDBCode argument of the DBError event.
Get the DBMS's message text	Use the SQLErrText argument of the DBError event.
Suppress the default message box	Specify an action/return code of 1.

About DataWindow action/return codes

Some events for DataWindow controls have codes that you can set to override the default action that occurs when the event is triggered. The codes and their meaning depend on the event. In PowerBuilder, you set the code with a RETURN statement. In the Web ActiveX, you call the SetActionCode or setActionCode method.

Example

PowerBuilder Here is a sample script for the DBError event:

```
// Database error -195 means that some of the
// required values are missing
IF sqldbcode = -195 THEN
       MessageBox("Missing Information",  &
           "You have not supplied values for all " &
           +"the required fields.")
END IF
// Return code suppresses default message box
RETURN 1
```

During execution, the user would see the following message box after the error:

Web ActiveX In JavaScript, the code for the DBError event might look like this:

```
// Database error -195 means that some of the
// required values are missing
if (sqldbcode == -195) {
       alert("Missing information:\n" +
           "You have not supplied values for all " +
           "the required fields.");
}
// Action code suppresses default message box
dw_1.SetActionCode(1);
```

Errors in property and data expressions and the Error event

A DataWindow control's Error event is triggered whenever an error occurs in a data or property expression at execution time. These expressions that refer to data and properties of a DataWindow object might be valid under some execution-time conditions but not others. The Error event allows you to respond with error recovery logic when an expression is not valid.

PowerBuilder syntax checking

In PowerBuilder, when you use a data or property expression, the PowerScript compiler checks the syntax only as far as the Object property. Everything following the Object property is evaluated at execution time. For example, in the following expression, the column name emp_name and the property Visible are not checked until execution time:

```
dw_1.Object.emp_name.Visible = "0"
```

If the emp_name column did not exist in the DataWindow, or if you had misspelled the property name, the compiler would not detect the error. However, at execution time, PowerBuilder would trigger the DataWindow control's Error event.

Using a Try-Catch block

The Error event is triggered even if you have surrounded an error-producing data or property expression in a Try-Catch block. The catch statement is executed after the Error event is triggered, but only if you do not code the Error event or do not change the default Error event action from ExceptionFail!. The following example shows a property expression in a Try-Catch block:

```
TRY
   dw_1.Object.emp_name.Visible = "0"
CATCH (dwruntimeerror dw_e)
   MessageBox ("DWRuntimeError", dw_e.text)
END TRY
```

Determining the cause of the error

The Error event has several arguments that provide information about the error condition. You can check the values of the arguments to determine the cause of the error. For example, you can obtain the internal error number and error text, the name of the object whose script caused the error, and the full text of the script where the error occurred. The information provided by the Error event's arguments can be helpful in debugging expressions that are not checked by the compiler.

If you catch a DWRuntimeError error, you can use the properties of that class instead of the Error event arguments to provide information about the error condition. The following table displays the correspondences between the Error event arguments and the DWRuntimeError properties.

Table 2-6: Correspondence between Error event arguments and DWRuntimeError properties

Error event argument	DWRuntimeError property
errornumber	number
errorline	line
errortext	text
errorwindowmenu	objectname
errorobject	class
errorscript	routinename

Controlling the outcome of the event

When the Error event is triggered, you can have the application ignore the error and continue processing, substitute a different return value, or escalate the error by triggering the SystemError event. In the Error event, you can set two arguments passed by reference to control the outcome of the event.

Table 2-7: Setting arguments in the Error event

Argument	Description
Action	A value you specify to control the application's course of action as a result of the error. Values are: ExceptionIgnore! ExceptionSubstituteReturnValue! ExceptionFail! (default action)
ReturnValue	A value whose datatype matches the expected value that the DataWindow would have returned. This value is used when the value of action is ExceptionSubstituteReturnValue!.

For a complete description of the arguments of the Error event, see the *DataWindow Reference*.

When to substitute a return value
The ExceptionSubstituteReturnValue! action allows you to substitute a return value when the last element of an expression causes an error. Do not use ExceptionSubstituteReturnValue! to substitute a return value when an element in the middle of an expression causes an error.

The ExceptionSubstituteReturnValue! action is most useful for handling errors in data expressions.

Updating the database

After users have made changes to data in a DataWindow control, you can use the Update method to save the changes in the database. Update sends to the database all inserts, changes, and deletions made in the DataWindow since the last Update or Retrieve method was executed.

How the DataWindow control updates the database

When updating the database, the DataWindow control determines the type of SQL statements to generate by looking at the status of each of the rows in the DataWindow buffers.

There are four DataWindow item statuses, two of which apply only to rows:

Table 2-8: DataWindow item status for rows and columns

Status			***Applies to***
PowerBuilder name	**Web DataWindow name**	**Numeric value**	
New!	New	2	Rows
NewModified!	NewModified	3	Rows
NotModified!	NotModified	0	Rows and columns
DataModified!	DataModified	1	Rows and columns

Named or numeric constants
The constants shown in the table are used differently in each environment:

PowerBuilder The named values are values of the enumerated datatype dwItemStatus. You must use the named values, which end in an exclamation point.

Web DataWindow You can use a string value with or without the exclamation point

Web ActiveX Named values are not defined; use the numeric values.

This discussion uses the PowerBuilder names.

How statuses are set

When data is retrieved When data is retrieved into a DataWindow, all rows and columns initially have a status of NotModified!.

After data has changed in a column in a particular row, either because the user changed the data or the data was changed programmatically, such as through the SetItem method, the column status for that column changes to DataModified!. Once the status for any column in a retrieved row changes to DataModified!, the row status also changes to DataModified!.

When rows are inserted When a row is inserted into a DataWindow, it initially has a row status of New!, and all columns in that row initially have a column status of NotModified!. After data has changed in a column in the row, either because the user changed the data or the data was changed programmatically, such as through the SetItem method, the column status changes to DataModified!. Once the status for any column in the inserted row changes to DataModified!, the row status changes to NewModified!.

When a DataWindow column has a default value, the column's status does not change to DataModified! until the user makes at least one actual change to a column in that row.

When Update is called

For rows in the Primary and Filter buffers When the Update method is called, the DataWindow control generates SQL INSERT and UPDATE statements for rows in the Primary and/or Filter buffers based upon the following row statuses:

Table 2-9: Row status after INSERT and UPDATE statements

Row status	SQL statement generated
NewModified!	INSERT
DataModified!	UPDATE

A column is included in an UPDATE statement only if the following two conditions are met:

- The column is on the updatable column list maintained by the DataWindow object

  For more information about setting the update characteristics of the DataWindow object, see the *PowerBuilder User's Guide*.

- The column has a column status of DataModified!

The DataWindow control includes all columns in INSERT statements it generates. If a column has no value, the DataWindow attempts to insert a NULL. This causes a database error if the database does not allow NULLs in that column.

For rows in the Delete buffer The DataWindow control generates SQL DELETE statements for any rows that were moved into the Delete buffer using the DeleteRow method. (But if a row has a row status of New! or NewModified! before DeleteRow is called, no DELETE statement is issued for that row.)

Changing row or column status programmatically

You might need to change the status of a row or column programmatically. Typically, you do this to prevent the default behavior from taking place. For example, you might copy a row from one DataWindow to another; and after the user modifies the row, you might want to issue an UPDATE statement instead of an INSERT statement.

You use the SetItemStatus method to programmatically change a DataWindow's row or column status information. Use the GetItemStatus method to determine the status of a specific row or column.

Changing column status

You use SetItemStatus to change the column status from DataModified! to NotModified!, or the reverse.

Change column status when you change row status
Changing the row status changes the status of all columns in that row to NotModified!, so if the Update method is called, no SQL update is produced. You must change the status of columns to be updated after you change the row status.

Changing row status

Changing row status is a little more complicated. The following table illustrates the effect of changing from one row status to another:

Table 2-10: Effects of changing from one row status to another

	Specified status			
Original status	**New!**	**NewModified!**	**DataModified!**	**NotModified!**
New!	-	Yes	Yes	No
NewModified!	No	-	Yes	New!
DataModified!	NewModified!	Yes	-	Yes
NotModified!	Yes	Yes	Yes	-

In the preceding table, *Yes* means the change is valid. For example, issuing SetItemStatus on a row that has the status NotModified! to change the status to New! does change the status to New!. *No* means that the change is not valid and the status is not changed.

Issuing SetItemStatus to change a row status from NewModified! to NotModified! actually changes the status to New!. Issuing SetItemStatus to change a row status from DataModified! to New! actually changes the status to NewModified!.

Changing a row's status to NotModified! or New! causes all columns in that row to be assigned a column status of NotModified!. Change the column's status to DataModified! to ensure that an update results in a SQL Update.

Changing status indirectly
When you cannot change to the desired status directly, you can usually do it indirectly. For example, change New! to DataModified! to NotModified!.

Creating reports

You can use DataWindow objects to create standard business reports such as financial statements, sales order reports, employee lists, or inventory reports.

To create a production report, you:

- Determine the type of report you want to produce
- Build a DataWindow object to display data for the report

- Place the DataWindow object in a DataWindow control on a window or form
- Write code to perform the processing required to populate the DataWindow control and print the contents as a report

Calling InfoMaker from an application
If your users have installed InfoMaker (the Sybase reporting product), you can invoke InfoMaker from an application. This way you can let your users create and save their own reports. To do this in PowerBuilder, use the Run function. For information about invoking InfoMaker, see the *InfoMaker User's Guide*.

Planning and building the DataWindow object

To design the report, you create a DataWindow object. You select the data source and presentation style and then:

- Sort the data
- Create groups in the DataWindow object to organize the data in the report and force page breaks when the group values change
- Enhance the DataWindow object to look like a report (for example, you might want to add a title, column headers, and a computed field to number the pages)

Using fonts
Printer fonts are usually shorter and fatter than screen fonts, so text might not print in the report exactly as it displays in the DataWindow painter. You can pad the text fields to compensate for this discrepancy. You should test the report format with a small amount of data before you print a large report.

Printing the report

After you build the DataWindow object and fill in print specifications, you can place it in a DataWindow control on a window or form, as described in "Putting a DataWindow object into a control" on page 14.

To allow users to print the report, your application needs code that performs the printing logic. For example, you can place a button on the window or form, then write code that is run when the user clicks the button.

To print the contents of a single DataWindow control or DataStore, call the Print method. For example, this PowerBuilder statement prints the report in the DataWindow control dw_Sales:

```
dw_Sales.Print(TRUE)
```

For information about the Print method, see the *DataWindow Reference*. For information about using nested reports to print multiple DataWindows, see "Using nested reports" on page 49.

Separate DataWindow controls in a single print job

For PowerBuilder applications only If the window has multiple DataWindow controls, you can use multiple PrintDataWindow method calls in a script to print the contents of all the DataWindow controls in one print job.

These statements print the contents of three DataWindow controls in a single print job:

```
int job
job = PrintOpen("Employee Reports")
// Each DataWindow starts printing on a new page.
PrintDataWindow(job, dw_EmpHeader)
PrintDataWindow(job, dw_EmpDetail)
PrintDataWindow(job, dw_EmpDptSum)
PrintClose(job)
```

For information about PowerBuilder system functions for printing, see the *PowerScript Reference.*

Using nested reports

When designing a DataWindow object for a report, you can choose to nest other reports (which are also DataWindow objects) within it. The basic steps for using nested reports in an application are the same ones you follow for the other report types. There are, however, some additional topics concerning nested reports that you should know about.

Availability
Composite and nested reports are not available in the Web DataWindow.

To learn about designing nested reports, see the *PowerBuilder User's Guide*.

Printing multiple updatable DataWindows on a page

An advantage of composite reports is that you can print multiple reports on a page. A limitation of composite reports is that they are not updatable, so you cannot *directly* print several updatable DataWindows on one page. However, there is an *indirect* way to do that, as follows.

You can use the GetChild method on named nested reports in a composite report to get a reference to a nested report. After getting the reference to the nested report, you can address the nested report during execution like other DataWindows.

Using this technique, you can call the ShareData method to share data between multiple updatable DataWindow controls and the nested reports in your composite report. This allows you to print multiple updatable DataWindows on a page through the composite report.

❖ **To print multiple DataWindows on a page using a composite DataWindow:**

1. Build a window or form that contains DataWindow controls with the updatable DataWindow objects.
2. Define a composite report that has reports corresponding to each of the DataWindows in the window or form that you want to print. Be sure to name each of the nested reports in the composite report.

   Naming the nested report
   To use GetChild on a nested report, the nested report must have a name. To name a nested report in the DataWindow painter, double-click it in the workspace and enter a name in the Name box on the General property page.

3. Add the composite report to the window or form (it can be hidden).
4. In your application, do the following:
   a. Retrieve data into the updatable DataWindow controls.
   b. Use GetChild to get a reference to the nested reports in the composite report.
   c. Use ShareData to share data between the updatable DataWindow objects and the nested reports.
   d. When appropriate, print the composite report.

      The report contains the information from the updatable DataWindow objects.

Re-retrieving data
Each time you retrieve data into the composite report, all references (handles) to nested reports become invalid, and data sharing with the nested reports is terminated. Therefore, be sure to call GetChild and ShareData each time after retrieving data.

Creating and destroying nested reports during execution

You can create and destroy nested reports in a DataWindow object dynamically during execution using the same technique you use to create and destroy other controls in a DataWindow object.

Creating nested reports To create a nested report, use the CREATE keyword with the Modify method. Supply the appropriate values for the nested report's properties.

Viewing syntax for creating a nested report
The easiest way to see the syntax for creating a nested report dynamically is to export the syntax of an existing DataWindow object that contains a nested report. The export file contains the syntax you need.

For more information about exporting syntax in the Library painter, see the *PowerBuilder User's Guide*.

When creating a nested report, you need to re-retrieve data to see the report. In a composite report, you can either retrieve data for the whole report or use GetChild to get a reference to the new nested report and retrieve its data directly. For nested reports in other reports, you need to retrieve data for the base report.

Destroying nested reports To destroy a nested report, use the DESTROY keyword with the Modify method. The nested report disappears immediately.

For more about creating and destroying controls in a DataWindow object or report, see Chapter 3, “Dynamically Changing DataWindow Objects.”

For a list of properties of nested reports, see the *DataWindow Reference*.

Using crosstabs

To perform certain kinds of data analysis, you might want to design DataWindow objects in the Crosstab presentation style. The basic steps for using crosstabs in an application are the same ones you follow for the other DataWindow types, but there are some additional topics concerning crosstabs that you should know about.

To learn about designing crosstabs, see the *PowerBuilder User's Guide*.

Viewing the underlying data

If you want users to be able to see the raw data as well as the cross-tabulated data, you can do one of two things:

- Place two DataWindow controls on the window or form: one that is associated with the crosstab and one that is associated with a DataWindow object that displays the retrieved rows.
- Create a composite DataWindow object that contains two reports: one that shows the raw data and one that shows the crosstab.

Do not share data between the two DataWindow objects or reports
They have the same SQL SELECT data definition, but they have different result sets.

For more about composite DataWindows, see the *PowerBuilder User's Guide*.

Letting users redefine the crosstab

Availability
This technique is available in PowerBuilder and the Web ActiveX.

With the CrosstabDialog method, you can allow users to redefine which columns in the retrieved data are associated with the crosstab's columns, rows, and values during execution.

The CrossTabDialog method displays the Crosstab Definition dialog box for the user to define the data for the crosstab's columns, rows, and values (using the same techniques you use in the DataWindow painter). When the user clicks OK in the dialog box, the DataWindow control rebuilds the crosstab with the new specifications.

Displaying informational messages

You can display informational messages when a crosstab is rebuilt during execution as a result of the call to CrosstabDialog. (The messages are the same ones you see when building a crosstab in the DataWindow painter, such as `Retrieving data` and `Building crosstab`.) You might want to do this if you are working with a very large number of rows and rebuilding the crosstab could take a long time.

PowerBuilder In PowerBuilder, you use a user event to display the crosstab's informational messages.

❖ **To display informational messages when a crosstab is rebuilt:**

1. Define a user event for the DataWindow control containing the crosstab. Associate it with the event ID pbm_dwnmessagetext.
2. In the script for the user event, get the value of the text argument (which holds the message that PowerBuilder would display when building the crosstab in the DataWindow painter) and display it to the user.

Web ActiveX In a Web page, you use the DataWindow's onMessageText event to handle informational messages.

❖ **To display informational messages when a crosstab is rebuilt:**

1. Edit the code for the onMessageText event of your DataWindow control.
2. In that event, get the value of the Text argument and display it to the user.

Examples

PowerBuilder In the example, code for the DataWindow control's user event for pbm_dwnmessagetext displays informational messages in a static text control in the window containing the crosstab:

```
st_message.Text = text
```

With that script in place, after CrosstabDialog has been called and the user has redefined the crosstab, as the crosstab is being rebuilt, your application dynamically displays the informational messages in the static text control st_message. (You might want to reset st_message.Text to be the empty string in the line following the CrosstabDialog call.)

In this example, code in the user event for pbm_dwnmessagetext displays informational messages as MicroHelp in an MDI application (w_crosstab is an MDI frame window):

```
w_crosstab.SetMicroHelp(text)
```

The informational messages are displayed in the MDI application's MicroHelp as the crosstab is rebuilt.

For more information

For more about user events in PowerBuilder, see the *PowerBuilder User's Guide*.

For more about the CrosstabDialog method and MessageText event, see the *DataWindow Reference*.

Modifying the crosstab's properties during execution

As with other DataWindow objects, you can modify the properties of a crosstab during execution using the Modify method. Some changes require the DataWindow control to dynamically rebuild the crosstab; others do not. (If the original crosstab was static, it becomes a dynamic crosstab when it is rebuilt.)

Availability
You can use this technique in all DataWindow environments.

Changes that do not force a rebuild

You can change the following properties without forcing the DataWindow control to rebuild the crosstab:

Table 2-11: Properties you can change on a crosstab DataWindow without forcing a rebuild

Properties	Objects
Alignment	Column, Compute, Text
Background	Column, Compute, Line, Oval, Rectangle, RoundRectangle, Text
Border	Column, Compute, Text
Brush	Line, Oval, Rectangle, RoundRectangle
Color	Column, Compute, Text
Edit styles (dddw, ddlb, checkbox, edit, editmask, radiobutton)	Column
Font	Column, Compute, Text
Format	Column, Compute

Properties	Objects
Pen	Line, Oval, Rectangle, RoundRectangle
Pointer	Column, Compute, Line, Oval, Rectangle, RoundRectangle, Text

Changes that force a rebuild

If you change any other properties, the DataWindow control rebuilds the structure of the crosstab when Modify is called. You should combine all needed expressions into one Modify call so that the DataWindow control has to rebuild the crosstab only once.

Default values for properties

For computations derived from existing columns, the DataWindow control by default uses the properties from the existing columns. For completely new columns, properties (such as font, color, and so on) default to the first column of the preexisting crosstab. Properties for text in headers default to the properties of the first text control in the preexisting crosstab's first header line.

For more about the Modify method, see Chapter 3, “Dynamically Changing DataWindow Objects.” For details on the DataWindow object properties, see the *DataWindow Reference*.

Generating HTML

You can use the data in a DataWindow object to create HyperText Markup Language (HTML) syntax. Once the HTML has been created, you can display it in a Web browser.

Web DataWindow not described here
This section does not include description of the Web DataWindow. The Web DataWindow uses DataWindow object properties that are described in detail in the *DataWindow Reference*. For overview information, see the “Web DataWindow properties” on page 126.

Techniques you can use

You can use any of several techniques to generate HTML from a DataWindow object.

In a painter In both the DataWindow painter and the Output view in the Database painter, you can save retrieved data in HTML format. To do this in the DataWindow painter, select File>Save Rows As from the menu. In the Database painter, open the Output view, then select Rows>Save Rows As from the menu. In both painters, specify HTML Table as the format for the file.

In your application code You can obtain an HTML string of the DataWindow presentation and data from the Data.HTMLTable property. You can save the string in a variable and modify the HTML with string manipulation operations. In PowerBuilder, you can also use the FileOpen and FileWrite functions to save the HTML to a file.

The HTMLTable property has its own properties which you can set to control the HTML attributes and style sheet associated with the Table HTML element.

PowerBuilder only In PowerBuilder, there are two more techniques available to you. You can:

- Call the SaveAs method to save the contents of a DataWindow directly to a file on disk. To save the data in HTML format, you need to specify HTMLTable as the file type when you call SaveAs.
- Call the GenerateHTMLForm method to create an HTML form from data contained in a DataWindow control or DataStore whose DataWindow object uses the Freeform or Tabular presentation style.

Choosing presentation styles

Some DataWindow presentation styles translate better into HTML than others. The following presentation styles produce good results:

Tabular
Group
TreeView
Freeform
Crosstab
Grid

The Composite, Graph, RichText, and OLE 2.0 presentation styles produce HTML output that is based on the result only, and not on the presentation style. DataWindows that have overlapping controls might not produce the expected results. Nested reports are ignored; they are not included in the generated HTML.

Example

This example illustrates how you might use DataWindow-generated HTML in an application.

The key line of code gets the HTML from the DataWindow by referring to its HTMLTable property. Variations for each environment are shown below. In PowerBuilder, you can use the Describe method or a property expression. The Web ActiveX has to use Describe.

PowerBuilder

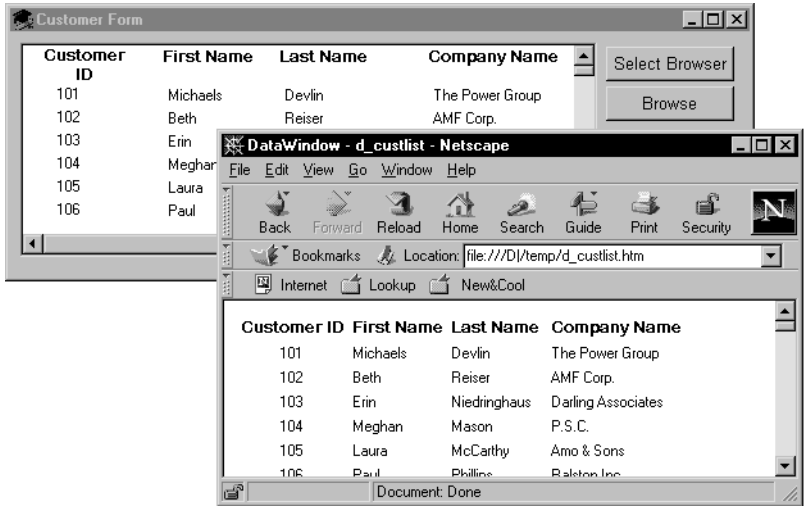
```
ls_htmlstring = dw_1.Object.DataWindow.Data.HTMLTable
```

Web ActiveX

```
str_html = dw_1.Describe("DataWindow.Data.HTMLTable");
```

The complete example that follows is implemented in PowerBuilder.

The window below displays customer data in a tabular DataWindow object. By pressing the Browse button, the user can translate the contents of the DataWindow object into HTML format and invoke a Web browser to view the HTML output. By pressing the Select Browser button, the user can tell the application which Web browser to use:



Script for the Select Browser button The script for the Select Browser button displays a dialog box where the user can select an executable file for a Web browser. The path to the executable is stored in is_Browser, which is an instance variable defined on the window:

```
String ls_BrowserName
Integer li_Result

// Open the dialog to select a browser.
li_Result = GetFileOpenName("Select Browser",  &
       is_Browser, ls_BrowserName,  &
       "exe", "Executable Files (*.EXE),*.EXE")

IF li_Result = -1 THEN
       MessageBox("No Browser", "No Browser selected")
END IF
```

Script for the Browse button The script for the Browse button creates an HTML string from the data in the DataWindow by assigning the Data.HTMLTable property to a string variable. After constructing the HTML string, the script adds a header to the HTML string. Then the script saves the HTML to a file and runs the Web browser to display the output.

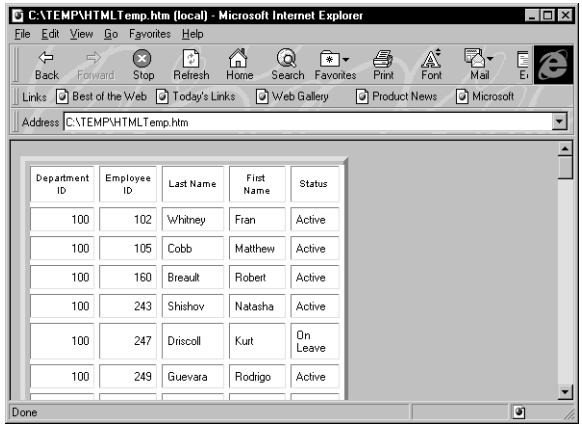
```
String ls_HTML, ls_FileName, ls_BrowserPath
Integer li_FileNumber, li_Bytes,
Integer li_RunResult, li_Result

// Generate the HTML.
ls_HTML = dw_1.Object.DataWindow.Data.HTMLTable
IF IsNull(ls_HTML) Or Len(ls_HTML) <= 1 THEN
       MessageBox ("Error", "Error generating HTML!")
       Return
ELSE
       ls_HTML ="<H1>HTML Generated From a DataWindow"&
          + "</H1><P>" + ls_HTML
END IF

//Create the file.
ls_FileName = "custlist.htm"
li_FileNumber = FileOpen(ls_FileName, StreamMode!, &
       Write!, LockReadWrite!, Replace! )

IF (li_FileNumber >= 0) THEN
       li_Bytes = FileWrite(li_FileNumber, ls_HTML)
       FileClose(li_FileNumber)
       IF li_Bytes = Len(ls_HTML) THEN
          // Run Browser with the HTML file.
          IF Not FileExists(is_Browser) THEN
             cb_selbrowser.Trigger Event Clicked()
             IF NOT FileExists(is_Browser) THEN
                MessageBox("Select Browser", "Could &
                   not find the browser.")
                RETURN
             END IF
          END IF
          li_RunResult = Run(is_Browser + " file:///"+&
             ls_FileName)
          IF li_RunResult = -1 THEN
             MessageBox("Error", "Error running
browser!")
          END IF
       ELSE
```

```
            MessageBox ("Write Error", &
                "File Write Unsuccessful")
        END IF
ELSE
        MessageBox ("File Error", "Could not open file")
END IF
```

Controlling display

You control table display and style sheet usage through the HTMLTable.GenerateCSS property. The HTMLTable.GenerateCSS property controls the downward compatibility of the HTML found in the HTMLTable property. If HTMLTable.GenerateCSS is FALSE, formatting (style sheet references) is not referenced in the HTMLTable property; if it is TRUE, the HTMLTable property includes elements that reference the cascading style sheet saved in HTML.StyleSheet.

This screen shows an HTML table in a browser using custom display features:



HTMLTable.GenerateCSS is TRUE

If the HTMLTable.GenerateCSS property is TRUE, the HTMLTable element in the HTMLTable property uses additional properties to customize table display. For example, suppose you specify the following properties:

```
HTMLTable.NoWrap=Yes
HTMLTable.Border=5
HTMLTable.Width=5
HTMLTable.CellPadding=2
HTMLTable.CellSpacing=2
```

Describe, Modify, and dot notation
You can access these properties by using the Modify and Describe PowerScript methods or by using dot notation.

The HTML syntax in the HTMLTable property includes table formatting information and class references for use with the style sheet:

```
<table cellspacing=2 cellpadding=2 border=5 width=5>
<tr>
<td CLASS=0 ALIGN=center>Employee ID
<td CLASS=0 ALIGN=center>First Name
<td CLASS=0 ALIGN=center>Last Name
<tr>
<td CLASS=6 ALIGN=right>102
<td CLASS=7>Fran
<td CLASS=7>Whitney
</table>
```

HTMLTable.GenerateCSS is FALSE

If HTMLTable.GenerateCSS is FALSE, the DataWindow does not use HTMLTable properties to create the Table element. For example, if GenerateCSS is FALSE, the HTML syntax for the HTMLTable property might look like this:

```
<table>
<tr>
<th ALIGN=center>Employee ID
<th ALIGN=center>First Name
<th ALIGN=center>Last Name
<tr>
<td ALIGN=right>102
<td>Fran
<td>Whitney
</table>
```

Merging HTMLTable with the style sheet

The HTML syntax contained in the HTMLTable property is incomplete: it is not wrapped in <HTML></HTML> elements and does not contain the style sheet. You can write code in your application to build a string representing a complete HTML page.

PowerBuilder example This example sets DataWindow properties, creates an HTML string, and returns it to the browser:

```
String ls_html
ds_1.Modify  &
("datawindow.HTMLTable.GenerateCSS='yes'")
ds_1.Modify("datawindow.HTMLTable.NoWrap='yes'")
ds_1.Modify("datawindow.HTMLTable.width=5")
```

```
ds_1.Modify("datawindow.HTMLTable.border=5")
ds_1.Modify("datawindow.HTMLTable.CellSpacing=2")
ds_1.Modify("datawindow.HTMLTable.CellPadding=2")
ls_html = "<HTML>"
ls_html +=  &
       ds_1.Object.datawindow.HTMLTable.StyleSheet
ls_html += "<BODY>"
ls_html += "<H1>DataWindow with StyleSheet</H1>"
ls_html += ds_1.Object.DataWindow.data.HTMLTable
ls_html += "</BODY>"
ls_html += "</HTML>"
return ls_html
```

This technique provides control over HTML page content. Use this technique as an alternative to calling the SaveAs method with the HTMLTable! Enumeration.

Calling the SaveAs method

Availability
The SaveAs method is not available for the Web control for ActiveX.

As an alternative to creating HTML pages dynamically, you can call the SaveAs method with the HTMLTable! Enumeration:

```
ds_1.SaveAs  &
       ("C:\TEMP\HTMLTemp.htm", HTMLTable!, TRUE)
```

This creates an HTML file with the proper elements, including the style sheet:

```
<STYLE TYPE="text/css">
<!--
.2 {COLOR:#000000;BACKGROUND:#ffffff;FONT-
STYLE:normal;FONT-WEIGHT:normal;FONT:9pt "Arial",
sans-serif;TEXT-DECORATION:none}

.3{COLOR:#000000;BACKGROUND:#ffffff;FONT-
STYLE:normal;FONT-WEIGHT:normal;FONT:8pt "MS Sans
Serif", sans-serif;TEXT-DECORATION:none}

.3{COLOR:#000000;BACKGROUND:#ffffff;FONT-
STYLE:normal;FONT-WEIGHT:normal;FONT:8pt "MS Sans
Serif", sans-serif;TEXT-DECORATION:none}
-->
</STYLE>
```

```
<TABLE nowrap cellspacing=2 cellpadding=2 border=5
width=5>
<tr>
       <td CLASS=2 ALIGN=right>Employee ID:
       <td CLASS=3 ALIGN=right>501
<tr>
       <td CLASS=2 ALIGN=right>Last Name:
       <td CLASS=3>Scott
<tr>
       <td CLASS=2 ALIGN=right>First Name:
       <td CLASS=3>David
<tr>
       <td CLASS=2 ALIGN=right>Status:
       <td CLASS=3>Active
</TABLE>
```

Displaying DataWindow objects as HTML forms

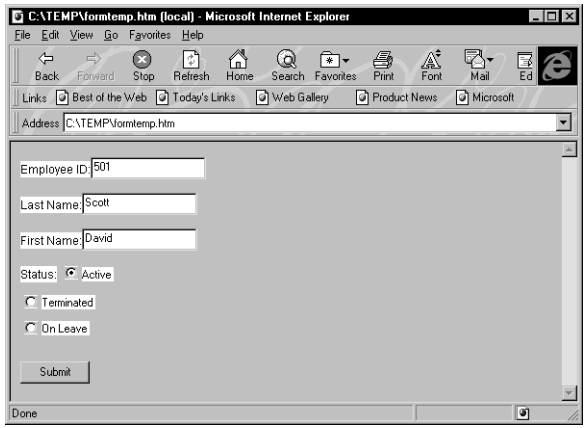
The GenerateHTMLForm method creates HTML form syntax for DataWindow objects. You can create an HTML form that displays a specified number of columns for a specified number of rows. Note the following:

- You create HTML form syntax by calling the GenerateHTMLForm method for the DataWindow control or DataStore
- The GenerateHTMLForm method creates HTML form syntax for the detail band only
- Embedded nested DataWindows are ignored; they are omitted from the generated HTML

Presentation styles

Although the GenerateHTMLForm method generates syntax for all presentation styles, the only styles that create usable forms are Freeform and Tabular.

The following HTML page shows a freeform DataWindow object converted into a form using syntax generated by the GenerateHTMLForm method:



Edit style conversion

The GenerateHTMLForm method converts column edit styles into the appropriate HTML elements:

Table 2-12: HTML elements generated for column edit styles

Column edit style	HTML element
CheckBox	Input element specifying TYPE=CHECKBOX
DropDownDataWindow	Select element with a single Option element
DropDownListBox	Select element with one Option element for each item in the DropDownListBox
Edit	Input element specifying TYPE=TEXT
RadioButton	Input element specifying TYPE=RADIO

Generating syntax

To generate HTML form syntax, you call the GenerateHTMLForm method:

```
instancename.GenerateHTMLForm ( syntax, style, action { , startrow,
    endrow, startcolumn, endcolumn { , buffer } } )
```

The method places the Form element syntax into the *syntax* argument and the HTML style sheet into the *style* argument, both of which are passed by reference.

Static texts in freeform DataWindow objects

All static texts in the detail band are passed through to the generated HTML form syntax. If you limit the number of columns to be converted using the *startcolumn* and *endcolumn* arguments, remove the headers from the detail band for the columns you eliminate.

Here is an example of the GenerateHTMLForm method:

```
String   ls_syntax, ls_style, ls_action
String   ls_html
Integer li_return
ls_action =  &
       "/cgi-
bin/pbcgi60.exe/myapp/uo_webtest/f_emplist"
li_return = ds_1.GenerateHTMLForm  &
       (ls_syntax, ls_style, ls_action)

IF li_return = -1 THEN
       MessageBox("HTML", "GenerateHTMLForm failed")
ELSE
       // of_MakeHTMLPage is an object method,
       // described in the next section.
       ls_html = this.of_MakeHTMLPage  &
           (ls_syntax, ls_style)
END IF
```

After calling the GenerateHTMLForm method, the ls_syntax variable contains a Form element. Here is an example:

```
<FORM ACTION=
       "/cgi-
bin/pbcgi60.exe/myapp/uo_webtest/f_emplist"
       METHOD=POST>
<P>
<P><FONT CLASS=2>Employee ID:</FONT>
<INPUT TYPE=TEXT NAME="emp_id_1" VALUE="501">

<P><FONT CLASS=2>Last Name:</FONT>
<INPUT TYPE=TEXT NAME="emp_lname_1" MAXLENGTH=20
VALUE="Scott">

<P><FONT CLASS=2>First Name:</FONT>
<INPUT TYPE=TEXT NAME="emp_fname_1" MAXLENGTH=20
VALUE="David">

<P><FONT CLASS=2>Status:</FONT>
<INPUT TYPE="RADIO" NAME="status_1" CHECKED CLASS=5
><FONT CLASS=5 >Active

<P>
<INPUT TYPE="RADIO" NAME="status_1" CLASS=5 >
<FONT CLASS=5 >Terminated
```

```
<P>
<INPUT TYPE="RADIO" NAME="status_1" CLASS=5 >
<FONT CLASS=5 >On Leave

<P>
<P>
<BR>
<INPUT TYPE=SUBMIT NAME=SAMPLE VALUE="OK">
</FORM>
```

The ls_stylesheet variable from the previous example contains a Style element, an example of which is shown below:

```
<STYLE TYPE="text/css">
<!--
.2{COLOR:#000000;BACKGROUND:#ffffff;FONT-
STYLE:normal;FONT-WEIGHT:normal;FONT:9pt "Arial",
sans-serif;TEXT-DECORATION:none}

.3{COLOR:#000000;BACKGROUND:#ffffff;FONT-
STYLE:normal;FONT-WEIGHT:normal;FONT:8pt "MS Sans
Serif", sans-serif;TEXT-DECORATION:none}

.5{COLOR:#000000;BACKGROUND:#ffffff;FONT-
STYLE:normal;FONT-WEIGHT:normal;FONT:8pt "MS Sans
Serif", sans-serif;TEXT-DECORATION:none}
-->
</STYLE>
```

Unique element names
The GenerateHTMLForm method creates unique names for all elements in the form (even when displaying multiple rows in one form) by adding a *_nextsequentialnumber* suffix.

Creating an HTML page

To use the syntax and style sheet returned by the GenerateHTMLForm method, you must write code to merge them into an HTML page. A complete HTML page requires <HTML> and <BODY> elements to contain the style sheet and syntax.

One way to do this is to create a global or object function that returns a complete HTML page, taking as arguments the Form and Style elements generated by the GenerateHTMLForm method. Such a function might contain the following code:

```
// Function Name: of_MakeHTMLPage
// Arguments: String as_syntax, String as_style
// Returns: String
```

```
//**********************************
String    ls_html
IF as_syntax = "" THEN
       RETURN ""
END IF

IF as_style = "" THEN
       RETURN ""
END IF

ls_html = "<HTML>"
ls_html += as_style
ls_html += "<BODY>"
ls_html += "<H1>Employee Information</H1>"
ls_html += as_syntax
ls_html += "</BODY></HTML>"
RETURN ls_html
```

CHAPTER 3

Dynamically Changing DataWindow Objects

About this chapter

This chapter describes how to modify and create DataWindow objects during execution.

Contents

Topic	Page
About dynamic DataWindow processing	67
Modifying a DataWindow object	68
Creating a DataWindow object	69
Providing query ability to users	72
Providing Help buttons	77
Reusing a DataWindow object	77

About dynamic DataWindow processing

Basics

DataWindow objects and all entities in them (such as columns, text, graphs, and pictures) each have a set of properties. You can look at and change the values of these properties during execution using DataWindow methods or property expressions. You can also create DataWindow objects during execution.

A DataWindow object that is modified or created during execution is called a dynamic DataWindow object.

About property expressions

Property expressions are available in PowerBuilder and the Web DataWindow. Property expressions use dot notation to address properties directly and are evaluated on the server component of the Web DataWindow.

For the Web ActiveX, property expressions are not available; use the Describe and Modify methods.

What you can do

Using this dynamic capability, you can allow users to change the appearance of the DataWindow object (for example, change the color and font of the text) or create ad hoc queries by redefining the data source. After you create a dynamic DataWindow object and the user is satisfied with the way it looks and the data that is displayed, the user can print the contents as a report.

Modifying a DataWindow object

During execution, you can modify the appearance and behavior of a DataWindow object by doing one of the following:

- Changing the values of its properties
- Adding or deleting controls from the DataWindow object

Changing property values

You can use the Modify method or a property expression to set property values. This lets you change settings that you ordinarily specify during development in the DataWindow painter.

Before changing a property, you might want to get the current value and save it in a variable so that you can restore the original value later. To obtain information about the current properties of a DataWindow object or a control in a DataWindow object, use the Describe method or a property expression.

Using expressions in property values

With some DataWindow properties, you can assign a value through an expression that the DataWindow evaluates during execution, instead of having to assign a value directly. For example, the following statement displays a salary in red if it is less than $12,000, and in black otherwise:

```
dw_1.Modify("salary.Color &
       = '0 ~t if(salary <12000,255,0)' ")
```

For more information

The syntax is different for expressions in code versus expressions specified in the DataWindow painter. For the correct syntax and information about which properties can be assigned expressions, see the *DataWindow Reference*.

For more information about property expressions and DataWindow object properties and examples of using Describe and Modify methods, see the *DataWindow Reference*.

Adding and deleting controls within the DataWindow object

You can also use the Modify method to:

- *Create* new objects in a DataWindow object

  This lets you add DataWindow controls (such as text, bitmaps, and graphic controls) dynamically to the DataWindow object.

  For how to get a good idea of the correct Create syntax, see "Specifying the DataWindow object syntax" on page 70.

- *Destroy* controls in a DataWindow object

  This lets you dynamically remove controls you no longer need.

PowerBuilder tool for easier coding of DataWindow syntax

PowerBuilder only Included with PowerBuilder is DW Syntax, a tool that makes it easy to build the correct syntax for property expressions, Describe, Modify, and SyntaxFromSQL statements. You click buttons to specify which properties of a DataWindow you want to use, and DW Syntax automatically builds the appropriate syntax, which you can copy and paste into your application code.

To access DW Syntax, select File>New and select the Tool tab.

Viewing DataWindow object properties in PowerBuilder

PowerBuilder only You can use the PowerBuilder Browser to get a list of DataWindow properties: on the DataWindow tab, select a DataWindow object in the left pane and Properties in the right pane. To see the properties for a control in a DataWindow object, double-click the DataWindow object name, then select the control.

Creating a DataWindow object

This section describes how to create a DataWindow object by calling the Create method in an application.

DataWindow painter
You should use the techniques described here for creating a DataWindow from syntax only if you cannot accomplish what you need to in the DataWindow painter. The usual way of creating DataWindow objects is to use the DataWindow painter.

To learn about creating DataWindow objects in the DataWindow painter, see the *PowerBuilder User's Guide*.

You use the Create method to create a DataWindow object dynamically during execution. Create generates a DataWindow object using source code that you specify. It replaces the DataWindow object currently in the specified DataWindow control with the new DataWindow object.

Resetting the transaction object
The Create method destroys the association between the DataWindow control and the transaction object. As a result, you need to reset the control's transaction object by calling the SetTransObject or SetTrans method after you call Create.

Web ActiveX If you used a connection technique that did not require you to call the SetTransObject or SetTrans method, you do not need to call it after Create either.

To learn how to associate a DataWindow control with a transaction object, see Chapter 2, "Using DataWindow Objects."

Specifying the DataWindow object syntax

There are several ways to specify or generate the syntax required for the Create method. Not all the techniques are available in all environments.

In PowerBuilder, you can:

- Use the SyntaxFromSQL method of the transaction object
- Use the LibraryExport PowerScript function

In all environments, you can:

- Use the DataWindow.Syntax property of the DataWindow object
- Create the syntax yourself

Using SyntaxFromSQL You are likely to use SyntaxFromSQL to create the syntax for most dynamic DataWindow objects. If you use SyntaxFromSQL, all you have to do is provide the SELECT statement and the presentation style.

In PowerBuilder, SyntaxFromSQL is a method of the transaction object. The transaction object must be connected when you call the method.

Setting USERID for native drivers
In PowerBuilder, table names are automatically qualified with the owner's name if you are using a native driver. To obtain the same results in an application, you must set the USERID property in the transaction object so that the table name is properly qualified and extended attributes can be looked up.

SyntaxFromSQL has three required arguments:

- A string containing the SELECT statement for the DataWindow object
- A string identifying the presentation style and other settings
- The name of a string you want to fill with any error messages that might result

SyntaxFromSQL returns the complete syntax for a DataWindow object that is built using the specified SELECT statement.

Using SyntaxFromSQL with Adaptive Server Enterprise
If your DBMS is Adaptive Server Enterprise and you call SyntaxFromSQL, PowerBuilder must determine whether the tables are updatable through a unique index. This is possible only if you set AutoCommit to TRUE before calling SyntaxFromSQL, as shown below:

```
sqlca.autocommit=TRUE
sqlca.syntaxfromsql (sqlstmt, presentation, err)
sqlca.autocommit=FALSE
```

Using LibraryExport in PowerBuilder You can use the LibraryExport PowerScript function to export the syntax for a DataWindow object and store the syntax in a string.

You can then use the exported syntax (or a modification of the syntax) in Create to create a DataWindow object.

Using the DataWindow.Syntax property You can obtain the source code of an existing DataWindow object to use as a model or for making minor changes to the syntax. Many values in the source code syntax correspond to properties of the DataWindow object.

This JavaScript example gets the syntax of the DataWindow object in the DataWindow control, dw_1, and displays it in the text box control, textb_dw_syntax :

```
var dwSyntax;
dwSyntax = dw_1.Describe("datawindow.syntax");
textb_dw_syntax.value = dwSyntax;
```

Creating the syntax yourself You need to create the syntax yourself to use some of the advanced dynamic DataWindow features, such as creating a group break.

The DataWindow source code syntax that you need to supply to the Create method can be very complex. To see examples of DataWindow object syntax, go to the Library painter and export a DataWindow object to a text file, then view the file in a text editor.

For more information on Create and Describe methods as well as DataWindow object properties and syntax, see the *DataWindow Reference*.

Providing query ability to users

When you call the Retrieve method for a DataWindow control, the rows specified in the DataWindow object's SELECT statement are retrieved. You can give users the ability to further specify which rows are retrieved during execution by putting the DataWindow into query mode. To do that, you use the Modify method or a property expression (the examples here use Modify).

Limitations
You cannot use query mode in a DataWindow object that contains the UNION keyword or nested SELECT statements.

How query mode works

Once the DataWindow is in query mode, users can specify selection criteria using query by example—just as you do when you use Quick Select to define a data source. When criteria have been defined, they are added to the WHERE clause of the SELECT statement the next time data is retrieved.

The following three figures show what happens when query mode is used.

First, data is retrieved into the DataWindow. There are 36 rows:

Rep	Quarter	Product	Units
Simpson	Q1	Stellar	12
Jones	Q1	Stellar	18
Perez	Q1	Stellar	15
Simpson	Q1	Cosmic	33
Jones	Q1	Cosmic	5
Perez	Q1	Cosmic	26
Simpson	Q1	Galactic	6

Row count: 36

Next, query mode is turned on. The retrieved data disappears and users are presented with empty rows where they can specify selection criteria. Here the user wants to retrieve rows where Quarter = Q1 and Units > 15:

Rep	Quarter	Product	Units
	Q1		>15

Row count: 36

Next, Retrieve is called and query mode is turned off. The DataWindow control adds the criteria to the SELECT statement, retrieves the three rows that meet the criteria, and displays them to the user:

Rep	Quarter	Product	Units
Jones	Q1	Stellar	18
Simpson	Q1	Cosmic	33
Perez	Q1	Cosmic	26

Row count: 3

You can turn query mode back on, allow the user to revise the selection criteria, and retrieve again.

Using query mode

❖ **To provide query mode to users during execution:**

1 Turn query mode on by coding.

In PowerBuilder:

```
dw_1.Modify("datawindow.querymode=yes")
```

In JavaScript:

```
dw_1.Modify("datawindow.querymode=yes");
```

All data displayed in the DataWindow is blanked out, though it is still in the DataWindow control's Primary buffer, and the user can enter selection criteria where the data had been.

2 The user specifies selection criteria in the DataWindow, just as you do when using Quick Select to define a DataWindow object's data source.

Criteria entered in one row are ANDed together; criteria in different rows are ORed. Valid operators are =, <>, <, >, <=, >=, LIKE, IN, AND, and OR.

For more information about Quick Select, see the *PowerBuilder User's Guide*.

3 Call AcceptText and Retrieve, then turn off query mode to display the newly retrieved rows.

In PowerBuilder:

```
dw_1.AcceptText()
dw_1.Modify("datawindow.querymode=no")
dw_1.Retrieve()
```

In JavaScript:

```
dw_1.AcceptText();
dw_1.Modify("datawindow.querymode=no");
dw_1.Retrieve();
```

The DataWindow control adds the newly defined selection criteria to the WHERE clause of the SELECT statement, then retrieves and displays the specified rows.

Revised SELECT statement
You can look at the revised SELECT statement that is sent to the DBMS when data is retrieved with criteria. To do so, look at the sqlsyntax argument in the SQLPreview event of the DataWindow control.

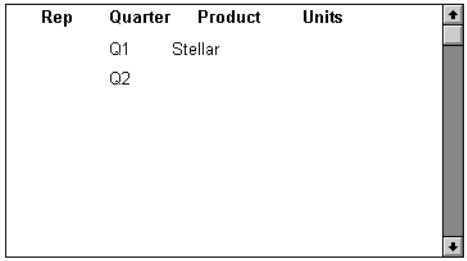
How the criteria affect the SELECT statement

Criteria specified by the user are added to the SELECT statement that originally defined the DataWindow object.

For example, if the original SELECT statement was:

```
SELECT printer.rep, printer.quarter, printer.product,
printer.units
FROM printer
WHERE printer.units < 70
```

and the following criteria are specified:



the new SELECT statement is:

```
SELECT printer.rep, printer.quarter, printer.product,
printer.units
FROM printer
WHERE printer.units < 70
AND (printer.quarter = 'Q1'
AND printer.product = 'Stellar'
OR printer.quarter = 'Q2')
```

Clearing selection criteria

To clear the selection criteria, Use the QueryClear property.

In PowerBuilder:

```
dw_1.Modify("datawindow.queryclear=yes")
```

In JavaScript:

```
dw_1.Modify("datawindow.queryclear=yes");
```

Sorting in query mode

You can allow users to sort rows in a DataWindow while specifying criteria in query mode using the QuerySort property. The following statement makes the first row in the DataWindow dedicated to sort criteria (just as in Quick Select in the DataWindow wizard).

In PowerBuilder:

```
dw_1.Modify("datawindow.querysort=yes")
```

In JavaScript:

```
dw_1.Modify("datawindow.querysort=yes");
```

Overriding column properties during query mode

By default, query mode uses edit styles and other definitions of the column (such as the number of allowable characters). If you want to override these properties during query mode and provide a standard edit control for the column, use the Criteria.Override_Edit property for each column.

In PowerBuilder:

```
dw_1.Modify("mycolumn.criteria.override_edit=yes")
```

In JavaScript:

```
dw_1.Modify("mycolumn.criteria.override_edit=yes");
```

You can also specify this in the DataWindow painter by checking Override Edit on the General property page for the column. With properties overridden for criteria, users can specify any number of characters in a cell (they are not constrained by the number of characters allowed in the column in the database).

Forcing users to specify criteria for a column

You can force users to specify criteria for a column during query mode by coding the following:

In PowerBuilder:

```
dw_1.Modify("mycolumn.criteria.required=yes")
```

In JavaScript:

```
dw_1.Modify("mycolumn.criteria.required=yes");
```

You can also specify this in the DataWindow painter by checking Equality Required on the General property page for the column. Doing this ensures that the user specifies criteria for the column and that the criteria for the column use = rather than other operators, such as < or >=.

Providing Help buttons

A DataWindow object has properties related to online Help. By initializing the DataWindow.Help.File property to the name of a Help file, you can display Help command buttons on dialog boxes that display for a DataWindow during execution.

For complete information on the Help-related DataWindow object properties, see the *DataWindow Reference*.

Reusing a DataWindow object

PowerBuilder only
This technique uses PowerScript methods, which are not available in other DataWindow environments.

You can reuse a DataWindow object by retrieving its syntax from the library it is stored in, then using the syntax to create a DataWindow object dynamically in a DataWindow control.

Here is a typical way to accomplish this in an application. Use:

- The LibraryDirectory function to obtain a list of DataWindow objects and other library entries in the current library
- A DropDownListBox to list the DataWindow objects in the library and then allow the user to select a DataWindow from the list
- The LibraryExport function to export the selected DataWindow object syntax into a string variable
- The Create method to use the DataWindow syntax to create the DataWindow object in the specified DataWindow control
- The Describe method to get the current DataWindow object syntax—for example:

```
string dwSyntax
dwSyntax = dw_1.Describe("datawindow.syntax")
```

- The Modify method to allow the user to modify the DataWindow object
- The LibraryImport function to save the user-modified DataWindow object in a library

For information about the PowerScript functions, see the *PowerScript Reference*. For information about the DataWindow methods Create, Describe, and Modify, see the *DataWindow Reference*.

CHAPTER 4

Using DataStore Objects

About this chapter

This chapter describes how to use DataStore objects in an application.

Contents

Topic	Page
About DataStores	79
Working with a DataStore	82
Using a custom DataStore object	82
Accessing and manipulating data in a DataStore	84
Sharing information	86

Before you begin

This chapter assumes you know how to build DataWindow objects in the DataWindow painter, as described in the *PowerBuilder User's Guide*.

About DataStores

A DataStore is a nonvisual DataWindow control. DataStores act just like DataWindow controls except that they do not have many of the visual characteristics associated with DataWindow controls. Like a DataWindow control, a DataStore has a DataWindow object associated with it.

Availability

In PowerBuilder, a DataStore is a nonvisual object. The Web control for ActiveX does not support DataStores.

The Web DataWindow server component uses an instance of a custom DataStore object to hold the DataWindow definition and data. See "Using a custom DataStore object" on page 82.

When to use a DataStore

DataStores are useful when you need to access data but do not need the visual presentation of a DataWindow control. DataStores allow you to:

- **Perform background processing against the database without having to hide DataWindow controls in a window**

  Suppose that the DataWindow object displayed in a DataWindow control is suitable for online display but not for printing. In this case, you could define a second DataWindow object for printing that has the same result set description and assign this object to a DataStore. You could then share data between the DataStore and the DataWindow control. Whenever the user asked to print the data in the window, you could print the contents of the DataStore.

- **Hold data used to show multiple views of the same information**

  When a window shows multiple views of the same information, you can use a DataStore to hold the result set. By sharing data between a DataStore and one or more DataWindow controls, you can provide different views of the same information without retrieving the data more than once.

- **Manipulate table rows without using embedded SQL statements**

  In places where an application calls for row manipulation without the need for display, you can use DataStores to handle the database processing instead of embedded SQL statements. DataStores typically perform faster at execution time than embedded SQL statements. Also, because the SQL is stored with the DataWindow object when you use a DataStore, you can easily reuse the SQL.

- **Perform database access on an application server**

  In a multitier application, the objects in a remote server can use DataStores to interact with the database. DataStores let you take advantage of the computer resources provided by a server machine, removing the need to perform database operations on each client.

DataStore methods

Most of the methods and events available for DataWindows are also available for DataStores. However, some of the methods that handle online interaction with the user are not available. For example, DataStores support the Retrieve, Update, InsertRow, and DeleteRow methods, but not GetClickedRow and SetRowFocusIndicator.

Prompting for information

When you are working with DataStores, you cannot use functionality that causes a dialog box to display to prompt the user for more information. Here are some examples of ways to overcome this restriction:

SetSort and SetFilter You can use the SetSort and SetFilter methods to specify sort and filter criteria for a DataStore object, just as you would with a DataWindow control. However, when you are working with a DataWindow control, if you pass a NULL value to either SetSort or SetFilter, the DataWindow prompts the user to enter information. When you are working with a DataStore, you must supply a valid format when you call the method. Moreover, you must supply a valid format when you share data between a DataStore and a DataWindow control; you cannot pass the NULL value to the DataWindow control rather than the DataStore.

Prompt for Criteria You can define your DataWindow objects so that the user is prompted for retrieval criteria before the DataWindow retrieves data. This feature works with DataWindow controls only. It is not supported with DataStores.

SaveAs When you use the SaveAs method with a DataWindow object, you can pass an empty string for the filename argument so that the user is prompted for a file name to save to. If you are working with a DataStore, you must supply the filename argument.

Prompt for Printing For DataWindow controls, you can specify that a print setup dialog box display at execution time, either by checking the Prompt Before Printing check box on the DataWindow object's Print Specifications property page, or by setting the DataWindow object's Print.Prompt property in a script. This is not supported with DataStores.

Retrieval arguments If you call the Retrieve method for a DataWindow control that has a DataWindow object that expects an argument, but do not specify the argument in the method call, the DataWindow prompts the user for a retrieval argument. This behavior is not supported with DataStores.

DataStores have some visual methods

Many of the methods and events that pertain to the visual presentation of the data in a DataWindow do not apply to DataStores. However, because you can print the contents of a DataStore and also import data into a DataStore, DataStores have some visually oriented events and methods. For example, DataStores support the SetBorderStyle and SetSeriesStyle methods so that you can control the presentation of the data at print time. Similarly, DataStores support the ItemError event, because data imported from a string or file that does not pass the validation rules for a column triggers this event.

For a complete list of the methods and events for the DataStore object and information about each method, see the *DataWindow Reference*.

DataStores require no visual overhead

Unlike DataWindow controls, DataStores do not require any visual overhead in a window. Using a DataStore is therefore more efficient than hiding a DataWindow control in a window.

Working with a DataStore

To use a DataStore, you first need to create an instance of the DataStore object in a script and assign the DataWindow object to the DataStore. Then, if the DataStore is intended to retrieve data, you need to set the transaction object for the DataStore. Once these setup steps have been performed, you can retrieve data into the DataStore, share data with another DataStore or DataWindow control, or perform other processing.

Examples

The following script uses a DataStore to retrieve data from the database. First it instantiates the DataStore object and assigns a DataWindow object to the DataStore. Then it sets the transaction object and retrieves data into the DataStore:

```
datastore lds_datastore
lds_datastore = CREATE datastore
lds_datastore.DataObject = "d_cust_list"
lds_datastore.SetTransObject (SQLCA)
lds_datastore.Retrieve()
/* Perform some processing on the data... */
```

Using a custom DataStore object

This section describes how to extend a DataStore in PowerBuilder by creating a user object.

You might want to use a custom version of the DataStore object that performs specialized processing. To define a custom DataStore, you use the User Object painter. There you specify the DataWindow object for the DataStore, and you can optionally write scripts for events or define your own methods, user events, and instance variables.

Using a custom DataStore involves two procedures:

1 In the User Object painter, define and save a standard class user object inherited from the built-in DataStore object.

2 Use the custom DataStore in your PowerBuilder application.

Once you have defined a custom DataStore in the User Object painter, you can write code that uses the user object to perform the processing you want.

For instructions on using the User Object painter in PowerBuilder, see the *PowerBuilder User's Guide*.

❖ **To define the standard class user object:**

1 Select Standard Class User Object on the PBObjects tab in the New dialog box.

2 Select datastore as the built-in system type that you want your user object to inherit from, and click OK.

The User Object painter workspace displays so that you can define the custom object.

3 Specify the name of the DataWindow object in the DataObject box in the Properties view and click OK.

4 Customize the DataStore by scripting the events for the object, or by defining methods, user events, and instance variables.

5 Save the object.

❖ **To use the user object in your application:**

1 Select the object or control for which you want to write a script.

2 Open the Script view and select the event for which you want to write the script.

3 Write code that uses the user object to do the necessary processing.

Here is a simple code example that shows how to use a custom DataStore to retrieve data from the database. First it instantiates the custom DataStore object, then it sets the transaction object and retrieves data into the DataStore:

```
uo_cust_dstore lds_cust_dstore
lds_cust_dstore = CREATE uo_cust_dstore
lds_cust_dstore.SetTransObject (SQLCA)
lds_cust_dstore.Retrieve()
/* Perform some processing on the data... */
```

Notice that this script does not assign the DataWindow object to the DataStore. This is because the DataWindow object is specified in the user object definition.

Changing the DataWindow object at execution time
When you associate a DataWindow object with a DataStore in the User Object painter, you are setting the initial value of the DataStore's DataObject property. During execution, you can change the DataWindow object for the DataStore by changing the value of the DataObject property.

4 Compile the script and save your changes.

Accessing and manipulating data in a DataStore

To access data using a DataStore, you need to read the data from the data source into the DataStore.

If the data source is a database

If the data for the DataStore is coming from a database (that is, the data source was defined as anything but External in the DataWindow painter), you need to communicate with the database to get the data. The steps you perform to communicate with the database are the same steps you use for a DataWindow control.

For more information about communicating with the database, see "Accessing the database" on page 21.

If the data source is not a database

If the data for the DataWindow object is not coming from a database (that is, the data source was defined as External in the DataWindow painter), you can use the following methods to import data into the DataStore:

ImportClipboard
ImportFile
ImportString

You can also get data into the DataStore by using a DataWindow data expression, or by using the SetItem method.

For more information on accessing data in a DataStore, see the *DataWindow Reference*.

About the DataStore buffers

Like a DataWindow control, a DataStore uses three buffers to manage data:

Table 4-1: DataStore buffers

Buffer	Contents
Primary	Data that has not been deleted or filtered out (that is, the rows that are viewable)
Filter	Data that was filtered out
Delete	Data that was deleted by the user or in a script

About the Edit control

The DataStore object has an Edit control. However, the Edit control for a DataStore behaves in a slightly different manner from the Edit control for a DataWindow. The Edit control for a DataWindow keeps track of text entered by the user in the current cell (row and column); the Edit control for a DataStore is used to manage data imported from an external source, such as the clipboard or a file. The text in the Edit control for a DataStore cannot be changed directly by the user. It must be manipulated programmatically.

Programming with DataStores

There are many methods for manipulating DataStore objects. These are some of the more commonly used:

Table 4-2: Common methods in DataStore objects

Method	Purpose
DeleteRow	Deletes the specified row from the DataStore.
Filter	Filters rows in the DataStore based on the current filter criteria.
InsertRow	Inserts a new row.
Print	Sends the contents of the DataStore to the current printer.
Reset	Clears all rows in the DataStore.
Retrieve	Retrieves rows from the database.
RowsCopy	Copies rows from one DataStore to another DataStore or DataWindow control.
RowsMove	Moves rows from one DataStore to another DataStore or DataWindow control.
ShareData	Shares data among different DataStores or DataWindow controls. See “Sharing information” on page 86.
Sort	Sorts the rows of the DataStore based on the current sort criteria.
Update	Sends to the database all inserts, changes, and deletions that have been made since the last Update.

For information about DataStore methods, see the *DataWindow Reference*.

Dynamic DataWindow objects The methods in the table above manipulate data in the DataStore but do not change the definition of the underlying DataWindow object. In addition, you can use the Modify and Describe methods to access and manipulate *the definition of a DataWindow object*. Using these methods, you can change the DataWindow object during execution. For example, you can change the appearance of a DataWindow or allow your user to create ad hoc reports.

For more information, see Chapter 3, "Dynamically Changing DataWindow Objects."

Property and data expressions You can use the same property and data expressions as for the DataWindow control. For information, see the *DataWindow Reference*.

Using DataStore properties and events This chapter mentions only a few of the properties and events that you can use to manipulate DataStores. For more information about DataStore properties and events, see the *DataWindow Reference*.

Sharing information

The ShareData method allows you to share a result set among two different DataStores or DataWindow controls. When you share information, you remove the need to retrieve the same data multiple times.

The ShareData method shares data retrieved by one DataWindow control or DataStore (called the primary DataWindow) with another DataWindow control or DataStore (the secondary DataWindow).

Result set descriptions must match

When you share data, the result set descriptions for the DataWindow objects must be the same. However, the SELECT statements can be different. For example, you could use the ShareData method to share data between DataWindow objects that have the following SELECT statements (because the result set descriptions are the same):

```
SELECT dept_id from dept

SELECT dept_id from dept where dept_id = 200

SELECT dept_id from employee
```

You can also share data between two DataWindow objects where the source of one is a database and the source of the other is external. As long as the lists of columns and their datatypes match, you can share the data.

What is shared?

When you use the ShareData method, the following information is shared:

- Primary buffer
- Delete buffer
- Filter buffer
- Sort order

ShareData does not share the formatting characteristics of the DataWindow objects. That means you can use ShareData to apply different presentations to the same result set.

When you alter the result set

If you perform an operation that affects the result set for either the primary or the secondary DataWindow, the change affects both of the objects sharing the data. Operations that alter the buffers or the sort order of the secondary DataWindows are rerouted to the primary DataWindow. For example, if you call the Update method for the secondary DataWindow, the update operation is applied to the primary DataWindow also.

Turning off sharing data

To turn off the sharing of data, you use the ShareDataOff method. When you call ShareDataOff for a primary DataWindow, any secondary DataWindows are disassociated and no longer contain data. When you call ShareDataOff for a secondary DataWindow, that DataWindow no longer contains data, but the primary DataWindow and other secondary DataWindows are not affected.

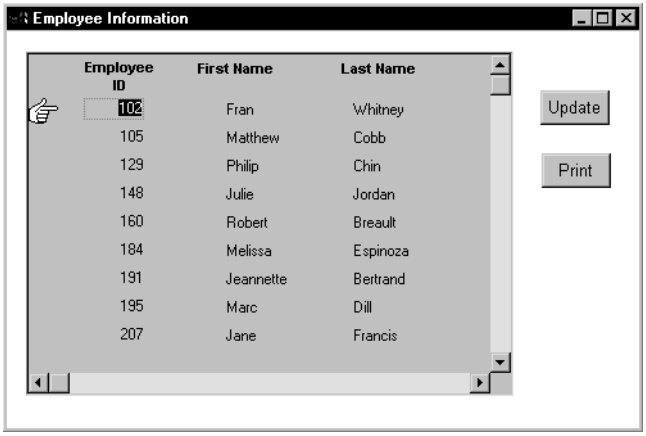
In most cases you do not need to turn off sharing, because the sharing of data is turned off automatically when a window is closed and any DataWindow controls (or DataStores) associated with the window are destroyed.

Crosstabs

You cannot share data with a DataWindow object that has the Crosstab presentation style.

Example: printing data from a DataStore

Suppose you have a window called w_employees that allows users to retrieve, update, and print employee data retrieved from the database:



The DataWindow object displayed in the DataWindow control is suitable for online display but not for printing. In this case, you could define a second DataWindow object for printing that has the same result set description as the object used for display and assign the second object to a DataStore. You could then share data between the DataStore and the DataWindow control. Whenever the user asked to print the data in the window, you could print the contents of the DataStore.

When the window or form opens

The code you write begins by establishing the hand pointer as the current row indicator for the dw_employees DataWindow control. Then the script sets the transaction object for dw_employees and issues a Retrieve method to retrieve some data. After retrieving data, the script creates a DataStore using the instance variable or data member ids_datastore, and assigns the DataWindow object d_employees to the DataStore. The final statement of the script shares the result set for the dw_employees DataWindow control with the DataStore.

This code is for the window's Open event:

```
dw_employees.SetRowFocusIndicator(Hand!)
dw_employees.SetTransObject(SQLCA)
dw_employees.Retrieve()

ids_datastore = CREATE datastore
ids_datastore.DataObject = "d_employees"
dw_employees.ShareData(ids_datastore)
```

Code for the Update button

Code for the cb_update button applies the update operation to the dw_employees DataWindow control.

This code is for the Update button's Clicked event:

```
IF dw_employees.Update() = 1 THEN
       COMMIT using SQLCA;
       MessageBox("Save","Save succeeded")
ELSE
       ROLLBACK using SQLCA;
       MessageBox("Save","Save failed")
END IF
```

Code for the Print button

The Clicked event of the cb_print button prints the contents of ids_datastore. Because the DataWindow object for the DataStore is d_employees, the printed output uses the presentation specified for this object.

This code is for the Print button's Clicked event:

```
ids_datastore.Print()
```

When the window or form closes

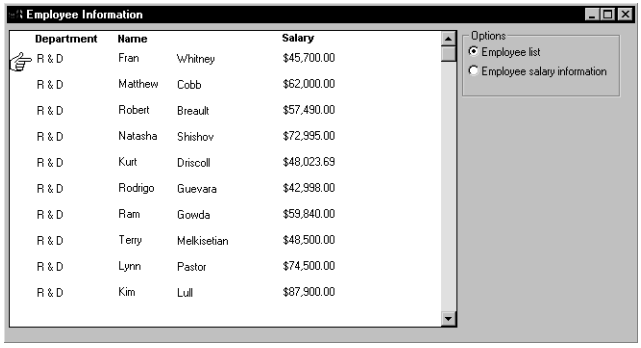
When the window closes, the DataStore gets destroyed.

This code is for the window's Close event:

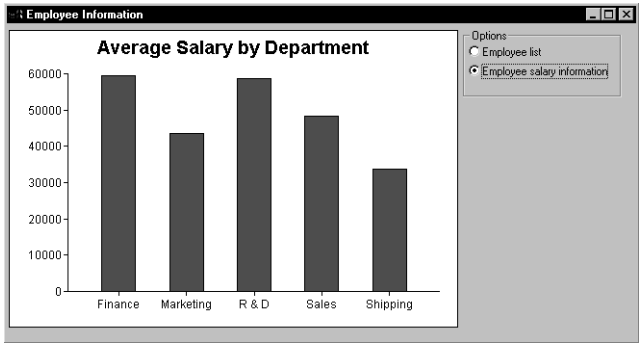
```
destroy ids_datastore
```

Example: using two DataStores to process data

Suppose you have a window called w_multi_view that shows multiple views of the same result set. When the Employee List radio button is selected, the window shows a list of employees retrieved from the database:



When the Employee Salary Information radio button is selected, the window displays a graph that shows employee salary information by department:



This window has one DataWindow control called dw_display. It uses two DataStores to process data retrieved from the database. The first DataStore (ids_emp_list) shares its result set with the second DataStore (ids_emp_graph). The DataWindow objects associated with the two DataStores have the same result set description.

When the window or form opens

When the window or form opens, the application sets the mouse pointer to the hourglass shape. Then the code creates the two DataStores and sets the DataWindow objects for the DataStores. Next the code sets the transaction object for ids_emp_list and issues a Retrieve method to retrieve some data.

After retrieving data, the code shares the result set for ids_emp_list with ids_emp_graph. The final statement triggers the Clicked event for the Employee List radio button.

This code is for the window's Open event:

```
SetPointer(HourGlass!)
ids_emp_list = Create DataStore
ids_emp_graph = Create DataStore

ids_emp_list.DataObject = "d_emp_list"
ids_emp_graph.DataObject = "d_emp_graph"

ids_emp_list.SetTransObject(sqlca)
ids_emp_list.Retrieve()
ids_emp_list.ShareData(ids_emp_graph)
rb_emp_list.EVENT Clicked()
```

Code for the Employee List radio button

The code for the Employee List radio button (called rb_emp_list) sets the DataWindow object for the DataWindow control to be the same as the DataWindow object for ids_emp_list. Then the script displays the data by sharing the result set for the ids_emp_list DataStore with the DataWindow control.

This code is for the Employee List radio button's Clicked event:

```
dw_display.DataObject = ids_emp_list.DataObject
ids_emp_list.ShareData(dw_display)
```

Code for the Employee Salary Information radio button

The code for the Employee Salary Information radio button (called rb_graph) is similar to the code for the List radio button. It sets the DataWindow object for the DataWindow control to be the same as the DataWindow object for ids_emp_graph. Then it displays the data by sharing the result set for the ids_emp_graph DataStore with the DataWindow control.

This code is for the Employee Salary Information radio button's Clicked event:

```
dw_display.DataObject = ids_emp_graph.DataObject
ids_emp_graph.ShareData(dw_display)
```

When the window or form closes

When the window closes, the DataStores get destroyed.

This code is for the window's Close event:

```
Destroy ids_emp_list
Destroy ids_emp_graph
```

Use garbage collection
Do not destroy the objects if they might still be in use by another process—rely on garbage collection instead.

CHAPTER 5

Manipulating Graphs

About this chapter

This chapter describes how to write code that allows you to access and change a graph in your application at execution time.

Contents

Topic	Page
Using graphs	93
Modifying graph properties	94
Accessing data properties	97
Using point and click	103

Using graphs

Supported environments

PowerBuilder and Web ActiveX Graphs are supported. Because you can print DataStores, PowerBuilder provides some events and functions for DataStores that pertain to the visual presentation of the data. However, graph functions such as CategoryCount, CategoryName, GetData, SeriesCount, and so forth depend on the visual graph control, which is not created for a DataStore. These functions return an error value or an empty string when used with DataStore objects.

Web DataWindow Graphs are not supported. If you use a DataWindow object that includes graphs, the graphs are ignored. If you use a DataWindow object with the Graph presentation style, nothing displays.

It is common for developers to design DataWindow objects that include one or more graphs. When users need to quickly understand and analyze data, a bar, line, or pie graph can often be the most effective format to display.

To learn about designing graphs, see the *PowerBuilder User's Guide*.

Working with graphs in your code

The following sections describe how you can access (and optionally modify) a graph by addressing its properties in code at execution time. There are two kinds of graph properties:

- **Properties of the graph definition itself** These properties are initially set in the DataWindow painter when you create a graph. They include a graph's type, title, axis labels, whether axes have major divisions, and so on.
- **Properties of the data** These properties are relevant only at execution time, when data has been loaded into the graph. They include the number of series in a graph (series are created at execution time), colors of bars or columns for a series, whether the series is an overlay, text that identifies the categories (categories are created at execution time), and so on.

Using graphs in other PowerBuilder controls
Although you will probably use graphs most often in DataWindow objects, you can also add graph controls to windows, and additional PowerScript functions and events are available for use with graph controls.

For more information, see PowerBuilder *Application Techniques*.

Modifying graph properties

When you define a graph in the DataWindow painter, you specify its behavior and appearance. For example, you might define a graph as a column graph with a certain title, divide its Value axis into four major divisions, and so on. Each of these entries corresponds to a property of a graph. For example, all graphs have a property GraphType, which specifies the type of graph.

When dynamically changing the graph type
If you change the graph type, be sure also to change the other properties as needed to properly define the new graph.

You can change these graph properties at execution time by assigning values to the graph's properties in code.

Property expressions

PowerBuilder You can modify properties using property expressions. For example, to change the type of the graph gr_emp to Column, you could code:

```
dw_empinfo.Object.gr_emp.GraphType = ColGraph!
```

To change the title of the graph at execution time, you could code:

```
dw_empinfo.Object.gr_emp.Title = "New title"
```

Modify method

In any environment , you can use the Modify method to reference parts of a graph.

Example for PowerBuilder For example, to change the title of graph gr_emp in DataWindow control dw_empinfo, you could code:

```
dw_empinfo.Modify("gr_emp.Title = 'New title'")
```

Example for Web ActiveX This example changes the label text for the Value axis of graph gr_emp in the DataWindow control dw_empinfo:

```
dw_empinfo.Modify("gr_emp.Values.Label = 'New label'");
```

For a complete list of graph properties, see the *DataWindow Reference*.

How parts of a graph are represented

Graphs consist of parts: a title, a legend, and axes. Each of these parts has a set of display properties. These display properties are themselves stored as properties in a subobject (structure) of Graph called *grDispAttr*.

For example, graphs have a Title property, which specifies the text for the title. Graphs also have a property TitleDispAttr, of type grDispAttr, which itself contains properties that specify all the characteristics of the title text, such as the font, size, whether the text is italicized, and so on.

Similarly, graphs have axes, each of which also has a set of properties. These properties are stored in a subobject (structure) of Graph called *grAxis*. For example, graphs have a property Values, of type grAxis, which specifies the properties of the Value axis, such as whether to use autoscaling of values, the number of major and minor divisions, the axis label, and so on.

Here is a representation of the properties of a graph:

```
Graph
        int Height
        int Depth
        grGraphType GraphType
        boolean Border
        string Title
        …
grDispAttr TitleDispAttr, LegendDispAttr, PieDispAttr
        string FaceName
        int TextSize
        boolean Italic
        …
grAxis Values, Category, Series
        boolean AutoScale
        int MajorDivisions
        int MinorDivisions
        string Label
        …
```

Referencing parts of a graph

You use dot notation or the Describe and Modify methods to reference the display properties of the various parts of a graph. For example, one of the properties of a graph's title is whether the text is italicized or not. That information is stored in the boolean Italic property in the TitleDispAttr property of the graph.

This example changes the label text for the Value axis of graph gr_emp in the DataWindow control dw_empinfo:

```
dw_empinfo.Object.gr_emp.Values.Label="New label"
```

For a complete list of graph properties, see the *DataWindow Reference*.

You can use the PowerBuilder Browser to examine the properties of a DataWindow object that contains a graph. For more information, see the *PowerBuilder User's Guide*.

Accessing data properties

To access properties related to a graph's data during execution, you use DataWindow methods for graphs. There are three categories of these methods related to data:

- Methods that provide information about a graph's data
- Methods that save data from a graph
- Methods that change the color, fill patterns, and other visual properties of data

How to use the methods

To call the methods for a graph in a DataWindow control, use the following syntax:

DataWindowName.methodName ("*graphName*", *otherArguments…*)

For example, there is a method CategoryCount, which returns the number of categories in a graph. So to get the category count in the graph gr_printer (which is in the DataWindow control dw_sales), write:

```
Ccount = dw_sales.CategoryCount("gr_printer")
```

Getting information about the data

There are quite a few methods for getting information about data in a graph in a DataWindow control at execution time. For all methods, you provide the name of the graph within the DataWindow as the first argument. You can provide your own name for graph controls when you insert them in the DataWindow painter. If the presentation style is Graph, you do not need to name the graph.

PowerBuilder These methods get information about the data and its display. For several of them, an argument is passed by reference to hold the requested information:

Table 5-1: Common methods for graph DataWindows in PowerBuilder

Method	Information provided
CategoryCount	The number of categories in a graph
CategoryName	The name of a category, given its number
DataCount	The number of data points in a series
FindCategory	The number of a category, given its name
FindSeries	The number of a series, given its name

Method	Information provided
GetData	The value of a data point, given its series and position (superseded by GetDataValue, which is more flexible)
GetDataPieExplode	The percentage at which a pie slice is exploded
GetDataStyle	The color, fill pattern, or other visual property of a specified data point
GetDataValue	The value of a data point, given its series and position
GetSeriesStyle	The color, fill pattern, or other visual property of a specified series
ObjectAtPointer	The graph element the mouse was positioned over when it was clicked
SeriesCount	The number of series in a graph
SeriesName	The name of a series, given its number

Web ActiveX These methods get information about the data and its display. There are additional helper methods available whenever the equivalent PowerBuilder method uses an argument passed by reference. These helper methods are identified in the second column of the following table (and are described in the *DataWindow Reference*):

Table 5-2: Common methods for graph DataWindows in Web ActiveX

Method	Information provided
CategoryCount	The number of categories in a graph.
CategoryName	The name of a category, given its number.
DataCount	The number of data points in a series.
FindCategory	The number of a category, given its name.
FindSeries	The number of a series, given its name.
ObjectAtPointer	The graph element the mouse was positioned over when it was clicked. Call ObjectAtPointerSeries and ObjectAtPointerDataPoint to get additional information.
SeriesCount	The number of series in a graph.
SeriesName	The name of a series, given its number.
Getting information about a data point's appearance	
GetDataPieExplode	The percentage at which a pie slice is exploded. Call GetDataPieExplodePercentage to retrieve the requested value.
GetDataStyleColor	The color of a specified data point. Call GetDataStyleColorValue to retrieve the requested value.
GetDataStyleFill	The fill pattern of a specified data point. Call GetDataStyleFillPattern to retrieve the requested value.

Method	Information provided
GetDataStyleLine	The line style and width of a specified data point. Call GetDataStyleLineWidth and GetDataStyleLineStyle to retrieve the requested values.
GetDataStyleSymbol	The symbol of a specified data point. Call GetDataStyleSymbolValue to retrieve the requested value.
Getting a data point's value	
GetDataDate	The value of a data point that contains a date, given its series and position. Call GetDataDateVariable to retrieve the requested value.
GetDataNumber	The value of a numeric data point, given its series and position. Call GetDataNumberVariable to retrieve the requested value.
GetDataString	The value of a string data point, given its series and position. Call GetDataStringVariable to retrieve the requested value.
Getting information about a series' appearance	
GetSeriesStyleColor	The color of a specified series. Call GetSeriesStyleColorValue to retrieve the requested value.
GetSeriesStyleFill	The fill pattern of a specified series. Call GetSeriesStyleFillPattern to retrieve the requested value.
GetSeriesStyleLine	The line style and width used by a specified series. Call GetSeriesStyleLineWidth and GetSeriesStyleLineStyle to retrieve the requested values.
GetSeriesStyleOverlay	Indication whether a series in a graph is an overlay (that is, whether it is shown as a line on top of another graph type). Call GetSeriesStyleOverlayValue to retrieve the requested value.
GetSeriesStyleSymbol	The symbol of a specified series. Call GetSeriesStyleSymbolValue to retrieve the requested value.

Saving graph data

PowerBuilder The following methods allow you to save data from the graph:

Table 5-3: PowerBuilder methods for saving data from a graph

Method	Action
Clipboard	Copies a bitmap image of the specified graph to the clipboard
SaveAs	Saves the data in the underlying graph to the clipboard or to a file in one of a number of formats

Web ActiveX You can save an image of the graph on the clipboard, but you cannot save data in a file. Writing to the file system is a security violation for an ActiveX control:

Table 5-4: Web Active X method for saving data from a graph

Method	Action
Clipboard	Copies a bitmap image of the specified graph to the clipboard

Modifying colors, fill patterns, and other data

PowerBuilder The following methods allow you to modify the appearance of data in a graph:

Table 5-5: PowerBuilder methods for modifying the appearance of data

Method	Action
ResetDataColors	Resets the color for a specific data point
SetDataStyle	Sets the color, fill pattern, or other visual property for a specific data point
SetSeriesStyle	Sets the color, fill pattern, or other visual property for a series

Web ActiveX These methods modify the appearance of data in a graph:

Table 5-6: Web ActiveX methods for modifying the appearance of data

Method	Action
ResetDataColors	Resets the color for a specific data point
SetDataColor	Sets the color of a specified data point
SetDataFill	Sets the fill pattern of a specified data point
SetDataLine	Sets the line style and width of a specified data point
SetDataPieExplode	Explodes a slice in a pie graph
SetDataSymbol	Sets the symbol of a specified data point
SetSeriesColor	Sets the color of a specified series
SetSeriesFill	Sets the fill pattern of a specified series
SetSeriesLine	Sets the line style and width used by a specified series
SetSeriesOverlay	Specifies whether a series in a graph is an overlay (that is, whether it is shown as a line on top of another graph type)
SetSeriesSymbol	Sets the symbol of a specified series

Using graph methods

You call the data-access methods after a graph has been created and populated with data. Some graphs, such as graphs that display data for a page or group of data, are destroyed and re-created internally as the user pages through the data. Any changes you made to the display of a graph, such as changing the color of a series, are lost when the graph is re-created.

Event for graph creation

To be assured that data-access methods are called whenever a graph has been created and populated with data, you can call the methods in the code for an event that is triggered when a graph is created. The event is:

- **PowerBuilder** Event ID pbm_dwngraphcreate, which you can assign to a user event for a DataWindow control (described below)
- **Web ActiveX** The onGraphCreate event

The graph-creation event is triggered by the DataWindow control after it has created a graph and populated it with data, but before it has displayed the graph. By accessing the data in the graph in this event, you are assured that you are accessing the current data and that the data displays the way you want it.

Setting up the PowerBuilder user event

PowerBuilder provides an event ID, pbm_dwngraphcreate, that you can assign to a user event for a DataWindow control.

❖ **To access data properties of a graph in a DataWindow control:**

1. Place the DataWindow control in a window or user object and associate it with the DataWindow object containing the graph.

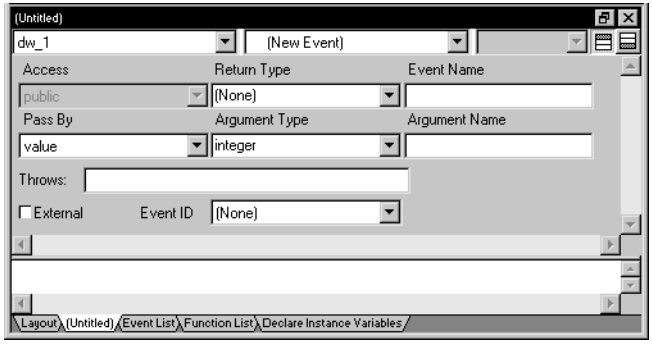
   Next you create a user event for the DataWindow control that is triggered whenever a graph in the control is created or changed.

2. Select Insert>Event from the menu bar.

   The Script view displays and includes prototype fields for adding a new event.

3. Select the DataWindow control in the first drop-down list of the prototype window.

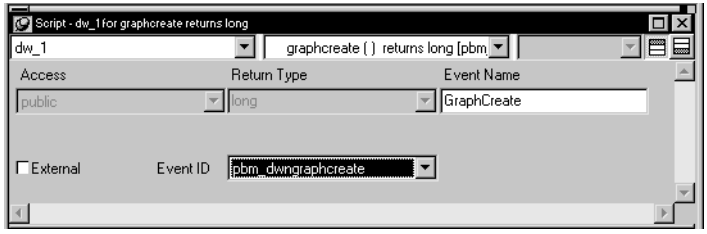
If the second drop-down list also changes to display an existing DataWindow event prototype, scroll to the top of the list to select New Event or select Insert>Event once again from the menu bar.



4 Name the user event you are creating.

For example, you might call it GraphCreate.

5 Select pbm_dwngraphcreate for the event ID.



6 Click OK to save the new user event.

7 Write a script for the new GraphCreate event that accesses the data in the graph.

Calling data access methods in the GraphCreate event assures you that the data access happens each time the graph has been created or changed in the DataWindow.

Examples

PowerBuilder The following statement sets to black the foreground (fill) color of the Q1 series in the graph gr_quarter, which is in the DataWindow control dw_report. The statement is in the GraphCreate event, which is associated with the event ID pbm_dwngraphcreate in PowerBuilder:

```
dw_report.SetSeriesStyle("gr_quarter", "Q1", &
      foreground!, 0)
```

The following statement changes the foreground (fill) color to red of the second data point in the Stellar series in the graph gr_sale in a window. The statement can be in a script for any event:

```
int SeriesNum
// Get the number of the series.
SeriesNum = gr_sale.FindSeries("Stellar")

// Change color of second data point to red
gr_sale.SetDataStyle(SeriesNum, 2, foreground!, 255)
```

Web ActiveX The following statement sets the foreground (fill) color to black in one of the series in the graph gr_quarter, which is in the DataWindow control dw_report. The statement is in the onGraphCreate event:

```
dw_report.SetSeriesStyleColor("gr_quarter", 1, 0, 0);
```

For more information

For complete information about the data-access graph methods, see the *DataWindow Reference*.

For more about PowerBuilder user events, see the *PowerBuilder User's Guide*.

Using point and click

Users can click graphs during execution. The DataWindow control provides a method called ObjectAtPointer that stores information about what was clicked. You can use this method in a number of ways in mouse events. For example, with the ObjectAtPointer information, you can call other graph methods to report to the user the value of the clicked data point. This section shows you how.

Mouse events and graphs

To cause actions when a user clicks a graph, you might:

- **PowerBuilder** Write a Clicked script for the DataWindow control
- **Web ActiveX** Write code for the MouseDown or onButtonClick event

You should call ObjectAtPointer in the first statement of the event's code.

Using ObjectAtPointer

ObjectAtPointer works differently in PowerBuilder and the Web ActiveX.

PowerBuilder ObjectAtPointer has this syntax:

DataWindowName.**ObjectAtPointer** ("*graphName*", *seriesNumber, dataNumber*)

ObjectAtPointer does these things:

- Returns the kind of object the user clicked

  The object is identified by a grObjectType enumerated value. For example, if the user clicks on a data point, ObjectAtPointer returns TypeData!. If the user clicks on the graph's title, ObjectAtPointer returns TypeTitle!.

  For a list of object values, see the chapter on constants in the *DataWindow Reference*. In PowerBuilder, you can also open the Browser and click the Enumerated tab.

- Stores the number of the series the pointer was over in the variable *seriesNumber*, which is an argument passed by reference
- Stores the number of the data point in the variable *dataNumber*, also an argument passed by reference

Web ActiveX ObjectAtPointer is used with two supporting methods to get all the information. ObjectAtPointer has this syntax:

DataWindowName.**ObjectAtPointer** ("*graphName*")

To get the information, you:

1 Call ObjectAtPointer, which returns the kind of graph element the user clicked.

  The element type is identified by a number. For example, if the user clicks on a series, ObjectAtPointer returns 1. If the user clicks on a graph's title, ObjectAtPointer returns 4.

  For a list of values for individual graph elements, see the chapter on constants in the *DataWindow Reference*.

2 Call ObjectAtPointerSeries, which returns the number of the series the pointer was over.

3 Call ObjectAtPointerDataPoint, which returns the number of the data point the pointer was over.

The second two methods must be called after ObjectAtPointer.

Example

Assume there is a graph named gr_sales in the DataWindow control dw_sales. The following code for the control's MouseDown event displays a message box:

- *If the user clicks on a series* (that is, if ObjectAtPointer returns 1), the message box shows the name of the series clicked on. The example uses the method GetSeriesName to get the series name, given the series number stored by ObjectAtPointer.
- *If the user clicks on a data point* (that is, if ObjectAtPointer returns 2), the message box lists the name of the series and the value clicked on. The example uses GetDataNumber to get the data's value, given the data's series and data point number.

PowerBuilder This code is for the Clicked event:

```
int SeriesNum, DataNum
double Value
grObjectType ObjectType
string SeriesName, ValueAsString
string GraphName
GraphName = "gr_sale"

// The following method stores the series number
// clicked on in SeriesNum and stores the number
// of the data point clicked on as DataNum.
ObjectType = &
       dw_printer.ObjectAtPointer (GraphName, &
       SeriesNum, DataNum)

IF ObjectType = TypeSeries! THEN
       SeriesName = &
           dw_printer.SeriesName (GraphName, SeriesNum)
       MessageBox("Graph", &
           "You clicked on the series " + SeriesName)

ELSEIF ObjectType = TypeData! THEN
       Value = dw_printer.GetData (GraphName, &
           SeriesNum, DataNum)
       ValueAsString = String(Value)
       MessageBox("Graph", &
           dw_printer.SeriesName (GraphName, &
           SeriesNum) + " value is " + ValueAsString)
END IF
```

Web ActiveX This code is for the MouseDown event:

```
number SeriesNum, DataNum, ObjectType, Success, Value;
string SeriesName, GraphName;

GraphName = "gr_sales";

ObjectType =
       dw_sales.GrObjectAtPointer(GraphName);

if (ObjectType == 1) {
       SeriesName =
          dw_sales.GetSeriesName(GraphName,
SeriesNum);

       alert("You clicked on the series " + SeriesName);
}
else {
       if (ObjectType == 2) {
          Success = dw_sales.GetDataNumber(GraphName,
             SeriesNum, DataNum, 1);
          if (Success == 1) {
             Value = GetDataNumberVariable( );

             alert(dw_sales.GetSeriesName(GraphName,
                SeriesNum) +" value is " + Value);
          }
       }
}
```

PART 2

Using the DataWindow in JSP and Web Applications

This part includes information about the Web DataWindow, including the server-side Web DataWindow component, and the DataWIndow Web Control for ActiveX.

The technologies described in the following chapters are not used for DataWindow objects and controls that you deploy to ASP.NET. For information on .NET Web Forms targets, see *Deploying Applications and Components to .NET* in the HTML Help.

CHAPTER 6

Using the Web DataWindow

About this chapter

This chapter describes how to use the Web DataWindow in data-based Web applications. Particular focus is given to the XML Web DataWindow because it can provide the best performance and you can customize its generation by applying a custom template to the default generation.

Contents

Topic	Page
What the Web DataWindow is	109
Using the XML Web DataWindow	115
Designing DataWindow objects for the Web DataWindow	123
Setting Web DataWindow design-time control properties	144
Setting up database connections	147
Deploying DataWindow objects to the component server	149
Creating Web page templates	152
Writing client-side scripts	154
Customizing Web DataWindow generation	157

What the Web DataWindow is

The Web DataWindow is a DataWindow that is generated for use in Web applications. The Web DataWindow offers a thin-client solution that provides most of the data manipulation, presentation, and scripting capabilities of the PowerBuilder DataWindow without requiring any PowerBuilder DLLs or plug-ins on the Web client. The DataWindow that displays in the Web browser looks very much like the DataWindow you designed in the DataWindow painter.

JavaScript keywords
You cannot use JavaScript reserved words to name fields or bands in a DataWindow control that you deploy to the Web. The list of reserved words is available on the Sun Microsystems Web site at http://docs.sun.com/source/816-6410-10/keywords.htm.

Web DataWindow types

Web DataWindow functionality includes three types of Web DataWindow implementation:

- **XML Web DataWindow** Separate XML (content), XSLT (layout), and CSS (style) with a subsequent transformation to XHTML
- **XHTML Web DataWindow** XHTML content only
- **HTML Web DataWindow** HTML content only

HTML Generation property page has been renamed
Since the Web DataWindow can be generated in XML, XHTML, and HTML, the HTML Generation property page in the DataWindow properties view is now called the Web Generation property page. Shared XHTML and HTML properties and properties specific to XML, XHTML, and HTML can be set there. For information about setting Web generation properties, see "Setting Web generation properties for the Web DataWindow" on page 129.

Web DataWindow use

The easiest way to use any type of Web DataWindow involves placing a Web DataWindow design-time control (DTC) on a Web page template in a JSP target and setting Web DataWindow DTC properties such as the generation format: XML, XHTML, or HTML.

No matter what type of Web DataWindow you want to use, the way you use it is the same. For specific information about using the XML Web DataWindow, see "Using the XML Web DataWindow" on page 115. For information about using the Web DataWindow DTC, see the chapter about the Web DataWindow design-time control in the *Working with JSP Targets* book and "Setting Web DataWindow design-time control properties" on page 144.

How the Web DataWindow works

The Web DataWindow uses a component running in a transaction server (such as EAServer or COM+) cooperating with a dynamic page server (such as Microsoft Active Server Pages in IIS or Java Server Pages in Tomcat) and communicating with a Web client by means of a Web server. No PowerBuilder DLLs or plug-ins are required on the Web client.

The built-in Web DataWindow component is the HTMLGenerator110 component that is preinstalled in EAServer 6.x. The component is used to generate the JavaScript and the HTML; XHTML; or XML, XSLT, CSS, and XHTML, depending on the type of Web DataWindow you are creating. You can also use a custom component that you develop instead. For more information, see "The Web DataWindow server component and client control" on page 113.

A DataWindow COM component can also be called directly from ASP without a transaction server, but using a transaction server usually provides better performance because database connections and instances of the component can be pooled or cached.

Disabling instance pooling when deploying Web DataWindow targets

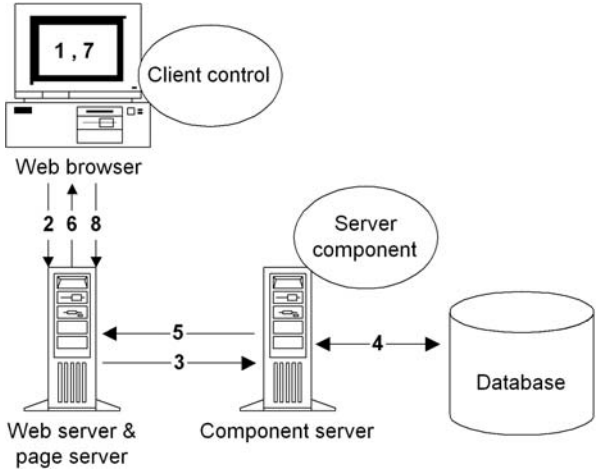
Instance pooling allows transaction server clients to reuse component instances and so improves server performance by eliminating the resource drain caused by repeated allocation of component instances. Instance pooling is turned on by default for the HTMLGenerator110 component in EAServer. This prevents you from deploying a target that uses the component a second time without stopping and restarting the server. During development, you might want to turn instance pooling off. For information on changing the pooling property of a component, see your server documentation.

In EAServer, you can disable instance pooling using EAServer Manager or by setting the com.sybase.jaguar.component.pooling property to false in the HTMLGenerator*XXX*.props file in the *\EAServer\Repository\Component\DataWindow* directory. The component pooling property should be set to *true* for production use.

What happens when a user requests a page

Figure 6-1 shows you graphically (in eight steps) what happens when a user accesses a Web page containing an XHTML or HTML Web DataWindow.

Figure 6-1: How the Web DataWindow works



The numbers 1 through 8 in the figure correspond to the events that occur after you develop and deploy a Web DataWindow and a user accesses a page containing the Web DataWindow:

1 In a Web browser, a user requests the URL for a page.

2 The Web server passes the request to the page server, which locates the template for the requested page and executes server-side scripts in the template.

3 The server-side scripts connect to the (transaction) server component, passing it information about the DataWindow and database connection.

4 Methods on the server component retrieve data required for the DataWindow from the database and translate the DataWindow definition, data, and state into JavaScript and XHTML or HTML.

5 The server component returns the JavaScript and XHTML or HTML and to the page server.

6 The page server replaces the server-side script in the requested Web page with the generated JavaScript and XHTML or HTML and returns the page to the Web browser through the Web server.

7 The user interacts with the DataWindow—for example, requesting the next page or updating the data.

8 The Web server passes the URL with added action parameters to the page server, and the cycle begins again.

The Web DataWindow server component and client control

The Web DataWindow has two main components: the server component and the client control.

Web DataWindow server component

The Web DataWindow server component retrieves data from a database and returns JavaScript and XSLT, XHTML, or HTML that represent the data and the DataWindow object definition to the page server. The server component is a PowerBuilder custom class user object that uses a DataStore to handle retrieval and updates and is deployed as an EAServer or COM component. You can use the generic component provided with PowerBuilder or a custom component.

The generic EAServer component, HTMLGenerator110, is preinstalled in EAServer in a package named *DataWindow*. PowerBuilder also provides a generic COM component called *PowerBuilder.HTMLDataWindow* that you can register and use with COM+ or EAServer. The COM component is in the file *PBDWR110.DLL*. For information about using the COM component, see "Server configuration requirements" on page 177.

Using an older version of the generic component

Earlier versions of the generic HTMLGenerator component are also installed in the EAServer DataWindow package. Because EAServer supports multiple PowerBuilder VMs, you can continue to run older Web DataWindow applications that use this component.

The generic component has methods that you call in your Web page template to instantiate and configure the component. The generic component also provides most of the methods available on the PowerBuilder DataWindow control. You should probably use the generic component when you are getting started with the Web DataWindow. Later you might want to build and deploy a custom component that uses the methods of the generic EAServer component interface or that uses only the methods you design for the component. For information, see "Using a custom server component" on page 194.

Types of server components and platforms

The following table describes all the types of Web DataWindow server components and their supported platforms:

Table 6-1: Web DataWindow server components and platforms

Web DataWindow server component	Platform (server component name)	Description
Generic	EAServer (DataWindow::HTMLGenerator110) or COM (PowerBuilder.HTMLDataWindow)	Prebuilt EAServer component or prebuilt COM component (ActiveX). No generation or compiling of stubs required.
Container (can include multiple DataWindow definitions)	EAServer (*PackageName*::*ComponentName* using generic DataWindow::HTMLGenerator110 interface)	Deploy with Web DW Container project wizard. Increases performance by reducing calls to server. No generation or compiling of stubs required.
Custom (hybrid)	EAServer (*PackageName*::*ComponentName* using generic DataWindow::HTMLGenerator110 interface)	Use to increase flexibility of generic component. Can build and deploy with PB object or project wizard. Generic methods remain available.
Custom (user-designed)	EAServer (*PackageName*::*ComponentName*) or COM (*PackageName.ComponentName*)	Potentially best for performance and scalability. Can use to avoid downloads of unneeded generic methods to client.

Web DataWindow client control

Embedding client-side scripts The Web DataWindow client control is the JavaScript plus XML, XSLT, and CSS; the JavaScript plus XHTML; or the JavaScript plus HTML that is generated by the server component and embedded in the page returned to the Web client. Client-side scripts that you add to your Web page template and wrap in SCRIPT tags are embedded as JavaScript in the client control.

JavaScript caching Some features available on the client control are optional: events, methods, data update and validation, and display formatting for newly entered data. The size of the generated JavaScript increases as you add more client-side functionality. You can cache client-side methods in JavaScript files on your Web server to reduce the size of the markup generated for Web DataWindow pages and, if the browser is configured to use cached files, improve the performance on the client machine.

For information about enabling JavaScript caching, see "Using JavaScript caching for Web DataWindow methods" on page 135. You can find additional information about client-side caching, HTMLGen properties, and other generation properties in the *DataWindow Reference*.

Using client-side events Events that are triggered on the client control and several of the client control methods do not require the server component to reload the page, so processing on the client is typically much faster than processing performed on the server.

For more information about enabling features on the client, see "Web DataWindow properties" on page 126 and "Controlling what is generated" on page 186. For more about writing scripts, see "Writing client-side scripts" on page 154.

For complete documentation of the events and methods available on the server component and the client control, see the *DataWindow Reference* or the PowerBuilder online Help.

Using the XML Web DataWindow

This section first provides you with a brief introduction to XML, XSLT, CSS, and XHTML and then describes what the XML Web DataWindow is and how to use it in Web applications.

Using the HTML Web DataWindow and the XHTML Web DataWindow The process for using the HTML Web DataWindow and the XHTML Web DataWindow is similar to the process of using the XML Web DataWindow. The only difference is in selecting the type of Web Generation you specify and and the DataWindow object properties you set.

Before an XML Web DataWindow can be generated for use in a Web application, you must create the DataWindow object you want to use in the DataWindow painter. For information about designing DataWindow objects for any type of Web DataWindow, see "Designing DataWindow objects for the Web DataWindow" on page 123.

About XML, XSLT, CSS, and XHTML

HTML is the most popular markup language in the world. The goal though of most HTML markup is appearance—the HTML tags do not provide you with any information. For example, if you see an HTML document with an element that has content as simple as <td>12345</td>, you do not know what the content represents. The content could be the zip code of a particular town, or it might be the population of the town.

An XML document:

- Contains information that is marked up with tags that describe all the pieces of information
- Models the relationships between all the pieces of information
- Is contained in a single element called the root element which becomes the root of a tree structure that contains other elements that represent the information

An XML document might include the element <zipcode>12345</zipcode>, and you know from the zipcode tag that 12345 is a zip code.

XML documents *separate the content from the presentation*, and they can be transformed (using XSLT, the Extensible Stylesheet Language for Transformations) into a variety of presentation types such as:

- An HTML page that includes <td>12345</td>
- A PDF file that includes zip code information
- A display of zip code information in wireless phones or pagers

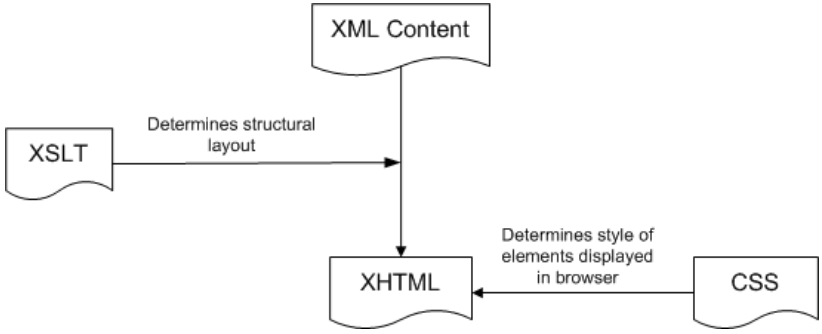
With XSLT, you can transform XML documents into other documents, which are often XML documents themselves. For example, Web pages created in XHTML (an XML-compliant version of HTML) are XML documents, and you can use XSLT to transform any XML document into a styled XHTML Web page for display in a browser.

A cascading style sheet (CSS) allows you to add style rules to the elements of a document that define how the content of the elements should be rendered. Using a CSS enables you to separate the contents of an HTML, XHTML, or XML document from its visual presentation. However, XSLT moves you beyond CSS because XSLT offers you complete flexibility to change the layout of content. XSLT also allows you to define rules that not only alter the design, but also add, change, or remove elements of the content if appropriate.

For an overview of XML, see the first section of the chapter on exporting and importing XML in the *PowerBuilder User's Guide*. For detailed information about XML and XSLT, see the O'Reilly and Associates, Inc. *Learning XML* and *XSLT* books.

How the XML Web DataWindow works

The XML Web DataWindow generates DataWindow content, layout, and style separately at runtime and renders in the browser a fully functional DataWindow in XHTML.



You can customize each of these XML Web DataWindow components at design time using a custom XHTML export template in the Export Template view for XHTML. For information, see "Customizing Web DataWindow generation" on page 157.

Server-side and client-side activity

When you have developed and deployed the pieces the XML Web DataWindow needs, here is what happens when a user requests the URL for a page containing the DataWindow.

Server-side activity

Server-side code is used to invoke the Web DataWindow generator. During the generation process:

1 Using the default XHTML export template or a custom template you created, an XHTML rendering of the DataWindow is generated in a Document Object Model (DOM) tree.

2 A CSS style sheet is generated in a DOM tree with the style information for the DataWindow elements.

   Generating as many of the style rules in CSS as possible (including all absolute positions) can increase page download speed because the stylesheet is downloaded only once and cached.

3 Client-side JavaScript files are generated for instantiating the control object and the array of row elements.

   You can improve performance by generating most of this client-side JavaScript in static files. For information about how you create and deploy the static JavaScript files, see "Using JavaScript caching for Web DataWindow methods" on page 135.

4 A reverse transformation of the XHTML DOM tree to XML (DataWindow content) and XSLT (DataWindow layout) occurs.

   XSLT also creates the structural layout of the page, saving bandwidth. Server processing is also reduced by offloading work to the client.

5 A small amount of JavaScript is generated to perform explicit transformation on the client side to render in the browser a fully functional DataWindow in XHTML.

Client-side activity

When a user accesses a Web page containing the XML Web DataWindow, the client browser:

1 Downloads the source XML file (DataWindow content for the page) and the XSLT stylesheet, which is cached locally.

2 Performs the transformation using the built-in Microsoft or Netscape XSLT processor.

3 Outputs the XHTML result into a <DIV> section on the page.

4 Downloads, caches, and applies the CSS stylesheet for display in the browser.

5 Downloads and caches JavaScript files.

6 Regenerates and downloads the XML file and JavaScript row objects file for the updated DataWindow page after a specified action by the user (HTTP Get/HTTP response).

Benefits of XHTML Web pages

XHTML Web pages are processed and rendered more quickly in the browser than HTML pages because extensive browser code is not needed to handle the more complex rules of HTML. Web users benefit from faster download of DataWindow pages because the XSLT and CSS stylesheets are downloaded only once and cached, resulting in bandwidth savings. Enterprises also benefit from the greater efficiency, scalability, extensibility, and accessibility gained by using standard W3C technologies.

Which type of Web DataWindow to use? The XML and XHTML Web DataWindow expand on the functionality provided by the HTML Web DataWindow. The following table shows you when you should use the XML or XHTML Web DataWindow and when you should use the HTML Web DataWindow:

Table 6-2: Features of Web DataWindow rendering formats

Feature	XML	XHTML	HTML
Web pages conform to industry standards	Yes	Yes	No
Pages can be customized using an XHTML export template	Yes	Yes	No
XSLT stylesheets are cached	Yes	No	No
CSS stylesheets are cached	Yes	Yes	No
Common JavaScript files can be cached	Yes	Yes	Yes
Most efficient handling of large amounts of paged data	Yes	No	No
Callback mechanism for paging and other client actions	Yes	Yes	Yes
Client-side mechanism for paging and other client actions	Yes	No	No
A Grid DataWindow page can be sorted on the client without a postback	Yes	No	No
Composite and nested DataWindows are supported	No	Yes	No

Feature	XML	XHTML	HTML
Absolute positioning is supported in Grid DataWindows	Yes	Yes	No
Greatest compatibility with accessibility software (Section 508)	No	Yes	No

The XML rendering format does not support accessibility software. Some aspects of the HTML generated using the HTML rendering format do not support accessibility software.

The caching and bandwidth savings and the client-side paging feature of the XML Web DataWindow result in better performance. In addition, you can customize the XHTML rendering of the XML Web DataWindow using XHTML export templates.

Customizing the XSLT transformation

The XSLT stylesheet that transforms the DataWindow content to XHTML can be customized by applying a custom XHTML export template to the default generation. The CSS stylesheet can be customized by applying custom *style* attributes in a custom XHTML export template. Using stylesheets to target the presentation enables the DataWindow to be rendered on virtually every device. For information about using the new Export Template view for XHTML in the DataWindow painter, see "Customizing Web DataWindow generation" on page 157.

Browser requirements for the XML Web DataWindow

The XML Web DataWindow requires browsers that support the following client-side technologies: XML, XSLT, XHTML, CSS, and JavaScript. You can select the browser to use for the XML Web DataWindow (XHTML format) in the Web Generation page in the DataWindow object property view.

Browser	XML parser/XSLT processor	XSLT version
Internet Explorer 5, 5.5	MSXML 2.0, 2.5 (update required)	XSL-WD
Internet Explorer 6.0	MSXML 3.0+	XSLT 1.0
Netscape 6+	TransforMiiX	XSLT 1.0
Mozilla 1.0+	TransforMiiX	XSLT 1.0

MSXML 2.6 or higher is required with Internet Explorer
The XML Web DataWindow requires MSXML 2.6 or higher with Internet Explorer. Internet Explorer 5 or 5.5 includes MSXML 2.0 or 2.5, so you must either update MSXML to 2.6 or higher or use Internet Explorer 6.0. For information about MSXML versions, refer to Microsoft Knowledge Base article 269238 on the Microsoft Web site.

How to use the XML Web DataWindow

The easiest way to use the XML Web DataWindow in your Web applications is to do the following:

1 Create a new DataWindow object or select an existing DataWindow object that you want to display in a Web browser.

For information, see "Designing DataWindow objects for the Web DataWindow" on page 123.

2 Set JavaScript Generation properties for the static JavaScript of the XML Web DataWindow if you have not already done so.

For information, see "Using JavaScript caching for Web DataWindow methods" on page 135.

Reusing static JavaScript for the XML Web DataWindow
If you are using the static JavaScript caching feature that was introduced with the HTML Web DataWindow, then you must regenerate and redeploy this static JavaScript for the XML Web DataWindow (and the XHTML Web DataWindow). You need to do this only once.

3 In the Web server's root publishing folder, create distinct Web publishing folders (for static JavaScript) and JavaScript publishing folders (for dynamic JavaScript) and set the Web and JavaScript Generation properties (that point to these folders) for your DataWindow.

If you do not create these folders, the generator creates them for you. If you do not set these properties, then the default object model creates a temp publishing folder *_tmp* automatically.

The next three steps
The next three steps involve creating a JSP target and a Web/JSP DataWindow page and using the Web DataWindow DTC. For information about these topics, see the *Working with JSP Targets* book.

4 Create a new JSP target using the JSP Target wizard on the Target tab page in the New dialog box.

   In the JSP Target wizard in the Select Object Model page, you specify whether you are using the default object model or not using any object model. You can also specify that the object model automatically clear the *_tmp* folder (if one is created).

5 Create a Java Server Page (JSP) using the Web/JSP DataWindow Page wizard on the JSP tab page in the New dialog box.

   In the Web/JSP DataWindow Page wizard, you specify whether you are using the Standard HTMLGenerator110 component, a custom HTMLGenerator110 component that you developed and deployed to EAServer, or a DataWindow container.

6 In the Sybase Web DataWindow DTC Properties, select the XML generation format.

   The XML generation format results in the separate generation of data and presentation in XML and XSLT for transformation to XHTML, and is the optimal format for the XML Web DataWindow.

7 Build and deploy the JSP target.

8 Display the results in the browser by typing in the URL.

Designing DataWindow objects for the Web DataWindow

The Web DataWindow supports most PowerBuilder DataWindow functionality. This section describes what features to use to take full advantage of the Web DataWindow.

Use default properties of DataWindow column edit styles
The properties of DataWindow column edit styles default to values that optimize their appearance—for example, radio buttons are left aligned. If you must change these style properties, the appearance of a column in the Web DataWindow might differ from its appearance in the DataWindow painter because the browser manages the rendering of HTML controls. You can adjust the appearance of the Web DataWindow by repositioning the control or resizing the column.

Using existing DataWindow objects

Many existing DataWindow objects work in the Web DataWindow. If a DataWindow object uses features that the Web DataWindow does not support, then the features are ignored. You can still use the DataWindow object if the remaining functionality is acceptable for your application. For example: if the DataWindow includes a graph control, the graph is ignored; if the DataWindow uses the Graph presentation style, the DataWindow object will not be useful.

Supported and unsupported features

Table 6-3: Web DataWindow supported and unsupported features

DataWindow feature	Supported and unsupported features
Presentation styles	All presentation styles except OLE, Graph, and RichText are supported. The TreeView style is supported for .NET Web Forms but not for JSP pages. Unsupported presentation styles retrieve data but display nothing. The Grid presentation style is rendered as an HTML table if you use the HTML format, and as a result absolute positioning is not supported and the display characteristics differ from those of XML and XHTML Web DataWindows.
Nested and composite reports	Supported for the XHTML format only.

DataWindow feature	Supported and unsupported features
Controls	Supported controls: Column, Computed Field, Graph, Text, Picture, Button, GroupBox, Rectangle. These controls are ignored: OLE Object, OLE Database Blob, RoundRectangle, Oval, InkPicture. Report controls are supported in XHTML Web DataWindows only. Rectangles cannot be rendered in a Label DataWindow with any rendering format when the layer of the Rectangle is foreground, unless the height of the DataWindow control is set to a fixed value. The following Rectangle properties are not supported: moveable, pointer, resizeable, slideleft, slideup, brush.hatch, pen.style GroupBoxes cannot be rendered in Crosstab and Grid style DataWindows. The following GroupBox properties are not supported: moveable, pointer, resizeable, slideleft, slideup, font.charset, font.width. Only horizontal Line controls are supported. The line's color property is always rendered, and the width property is rendered if the line is solid. Other line styles are displayed as solid lines with the default width. Vertical and slanted lines are ignored. For information on: • Expressions for computed fields, see "Using expressions" on page 138. • Images for Picture controls, see "Using Picture controls" on page 142. • Including valid HTML in a control, see "Including HTML in a control" on page 140 • Button controls and supported actions, see "Using Button controls" on page 141.
Retrieving data	Up to 16 retrieval arguments are supported. Filtering and sorting are supported by setting properties with the Modify method or calling methods on the server component. Sorting can also be specified by using a client control method. User-specified queries using the QueryMode property are not supported.
Updating data	Same as the PowerBuilder DataWindow control. The DataWindow object must contain editable columns.
Edit styles	All edit styles are supported except InkEdit and EditMask, with the exception of the DDCalendar EditMask. If the DataWindow uses the EditMask edit style, the styles specified are treated as though they were specified as a display format.
DDCalendar EditMask property	The DDCalendar EditMask property option allows for separate selections of the calendar month, year, and date. This option can be set in a check box on the Edit tab of the DataWindow painter Properties view when a Date or DateTime column with the EditMask edit style is selected. It can also be set in code, as in this example for the birth_date column: `dwEmp.Modify("birth_date.EditMask.DDCalendar='Yes'")` For more information, see "Using a drop-down calendar" on page 125.

DataWindow feature	Supported and unsupported features
DropDownDataWindows	A drop-down DataWindow must be in the same PBL as the DataWindow in which it is used. Data for drop-down DataWindows is retrieved on the server. See "Using drop-down DataWindows" on page 132. The dddw.lines property is not supported in Web pages because the browser controls how the DropDownDataWindow displays.
Display formats	Supported, including the use of color.
Validation rules	The expression might be evaluated on the client or the server, depending on the expression. For information, see "Using expressions" on page 138.
Property expressions	Evaluated on the server.
Layout	Properties that specify autosizing of height and width or allow the user to resize or move controls, such as SlideLeft and SlideRight, are ignored.
Properties	The following properties are not supported: • EditMask.Spin DataWindow object property • Sparse (Suppress Repeating Values) DataWindow object property • RightToLeft DataWindow control property • The ShowConnectLines and ShowLeafNodeConnectLines properties of the TreeView Web DataWindow The Limit property is not supported in multiline edit columns in a Web DataWindow. In JavaScript, the multiline edit column maps to a textarea object, and the limit property maps to a maxlength attribute, which the textarea object does not support.
Tab order	Supported in HTML 4 and later browsers.

Using a drop-down calendar

The drop-down calendar DataWindow option is available for use on any DataWindow column with an EditMask, and a Date, DateTime, or TimeStamp datatype. The DDCalendar EditMask property option allows for separate selections of the calendar month, year, and date. This option can be set in a check box on the Edit tab of the DataWindow painter Properties view when a column with the EditMask edit style is selected. It can also be set in code, as in this example for the birth_date column:

```
dw_1.Modify("birth_date.EditMask.DDCalendar='Yes'")
```

You can set the following properties to control the display of the calendar in a script or on the Other page in the Properties view for the column:

Painter option	Property
Drop Align Right	Column.Editmask.ddcal_alignright
CalendarBackColor	Column.Editmask.ddcal_backcolor
CalendarTextColor	Column.Editmask.ddcal_textcolor
CalendarTitleBackColor	Column.Editmask.ddcal_titlebackcolor

Painter option	Property
CalendarTitleTextColor	Column.Editmask.ddcal_titletextcolor
CalendarTrailingTextColor	Column.Editmask.ddcal_trailingtextcolor

To make sure that dates selected with the drop-down calendar option are displayed with the desired edit mask for Web DataWindows, you should specify that the Client Formatting option be included with the static JavaScript generated and deployed for the DataWindow. To conserve bandwidth, JavaScript for client formatting is not included by default. To include this script, you can select the Client Formatting check box on the Web Generation page of the DataWindow Properties view. If you do not include script for client formatting, the drop-down calendar will use a default edit mask to display the column data based on the client machine's default localization settings.

To navigate in the drop-down calendar, a user can:

- Click the arrows in the top corners to move from month to month.
- Click the month to display a list of months, then click a month to select it.
- Click the year to display a spin control, then use the spin control’s arrows to select a year.
- Click a date to select the date and close the calendar.
- Press the Esc key to close the calendar without changing the selection.

Web DataWindow properties

This section describes the XML, XHTML, and HTML DataWindow object properties for the Web DataWindow. You can set these properties in the DataWindow painter or in script.

For more detailed information about each property, see the *DataWindow Reference* or the online Help for the property name. For information about how to set properties in the DataWindow painter, including shared HTML and XHTML properties, see “Setting Web generation properties for the Web DataWindow” on page 129.

XML and XHTML data properties

Table 6-4 shows row data properties for the XML and XHTML Web DataWindow.

Table 6-4: Row properties for the XML and XHTML Web DataWindow

Property	User interface fields	Description
Data.XHTML	Read only, so no user interface field	A string containing the row data content of the DataWindow object in XHTML format
Data.XMLWeb	Read only, so no user interface field	A string containing browser-specific JavaScript that performs the XSLT transformation on the browser

XML Web DataWindow generation properties

Table 6-5 lists properties supporting XML Web DataWindow generation.

Table 6-5: Properties supporting XML Web DataWindow generation

Property	User interface fields	Allows you to
CSSGen.*property*	Web Generation property page with CSS selected as the format to configure: resource base and publish path	Specify the physical path to which a generated CSS style sheet is published and the URL indicating the location of the style sheet where the property variable is PublishPath or ResourceBase.
JSGen.*property*	JavaScript Generation property page with XHTML selected as the format to configure: resource base and publish path	Specify the physical path to which generated JavaScript (that is included in the final XHTML page) is published and the URL indicating the location of the generated JavaScript where the property variable is PublishPath or ResourceBase.
XMLGen.*property*	Web Generation property page with XML selected as the format to configure: resource base and publish path	Specify the physical path to which XML is published and the URL referenced by the JavaScript that transforms the XML to XHTML where the property variable is PublishPath or ResourceBase. You can also specify whether XML is generated inline to the XSLT transformation script and whether paging is performed on the client or server.

Property	User interface fields	Allows you to
XSLTGen.*property*	Web Generation property page with XSLT selected as the format to configure: resource base and publish path	Specify the physical path to which the generated XSLT style sheet is published and the URL referenced by the JavaScript that transforms the XML to XHTML (using the generated XSLT stylesheet) where the property variable is PublishPath or ResourceBase.
XHTMLGen.Browser	Web Generation property page with XHTML selected as the format to configure: browser	Identify the browser in which XHTML generated within an XSLT style sheet is displayed.

About PublishPath and ResourceBase
PublishPath is a string that specifies the physical path of the Web site folder to which PowerBuilder publishes generated CSS, JavaScript, XML, or XSLT. ResourceBase is a string that specifies the URL of the generated file.

HTML properties

There are four types of HTML properties you can set in the DataWindow painter. The first three apply to the DataWindow object itself. The last applies to bitmap, column, computed field, and text controls in the DataWindow object.

Some properties are now shared but property names are unchanged
In previous versions of PowerBuilder, HTML properties applied only to the HTML Web DataWindow. Now some HTML properties are shared with the XML Web DataWindow and the XHTML Web DataWindow. The HTML property names have not changed.

Table 6-6: HTML properties you can set in the DataWindow painter

Property	User interface fields	Allows you to
HTMLDW (shared)	Web DataWindow check box on the General page of the DataWindow object Property view. Selecting this check box sets this property to Yes.	View the HTML in a browser using Design>HTML Preview (or if you plan to use the DataWindow object with a custom Web DataWindow server component). The generic server component automatically sets this property to Yes.

Property	User interface fields	Allows you to
HTMLTable.*property* (HTML only)	All fields on the HTML Table page of the DataWindow object Property view.	Change the display characteristics of HTML tables, including border style and cell width and padding.
HTMLGen.*property* (shared by all Web DataWindow formats)	All fields on the Web Generation page and the JavaScript Generation page of the DataWindow object Property view. (You can also start the JavaScript Generation wizard using the Generate File button on the JavaScript Generation page).	Control the number of rows displayed on the page, generate HTML for a specific browser or HTML version, choose client-side features to incorporate into the page, and set up JavaScript caching to enhance performance.
HTML.*property* (shared by all Web DataWindow formats)	All fields on the HTML page of the Property view for a Column, Computed Field, Text, or Picture control in a DataWindow object.	Set up hyperlinks and retrieval arguments typically used to create master/detail Web pages, specify whether the content of a control should be rendered as HTML, and specify any HTML to be appended to a control.

Setting Web generation properties for the Web DataWindow

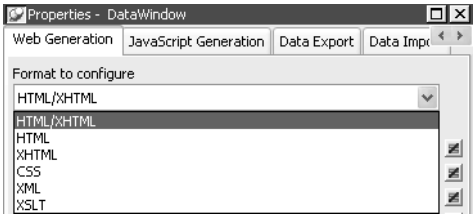
Each of the Web formats (and ultimately the files) that contribute to the generation of a Web DataWindow require configuration:

HTML
XHTML
CSS
XML
XSLT
JavaScript

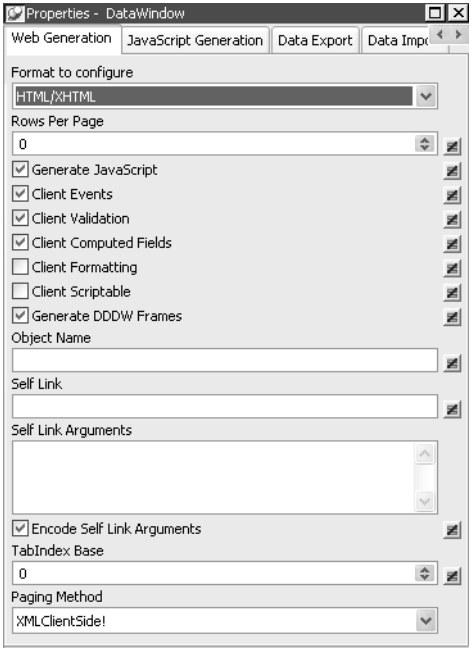
XHTML and HTML optimized for a browser
For information about generating XHTML and HTML optimized for a specific browser, see “Specifying Web generation for a specific browser” on page 143

The rest of this section describes configuration of HTML, XHTML, CSS, XML, and XSLT. For information about JavaScript configuration, see “Using JavaScript caching for Web DataWindow methods” on page 135.

To configure a particular Web format, you use the Web Generation page in the DataWindow object property view. The Web Generation page is controlled by the *Format to configure* drop-down list box at the top of the view that displays the Web formats for the Web DataWindow:



The properties that are shared by all rendering formats display in the view by default:



The properties you can set are subject to change based on the format you select:

Format to configure	Description	Properties
HTML and XHTML (shared)	Properties that are shared by all Web DataWindow rendering formats	Rows per page Generate JavaScript Client events Client validation Client computed fields Client formatting Client scriptable Generate DDW Frames Object name Self link Self link arguments Encode self link arguments TabIndex base Paging method
HTML	HTML-only Web DataWindow properties	Browser HTML version
XHTML	XHTML-only XML Web DataWindow properties	Browser
CSS	XML Web DataWindow CSS properties	Resource base Publish path Session specific file names
XML	XML Web DataWindow data and presentation properties	Resource base Publish path Generate securely inline Paging method
XSLT	XML Web DataWindow XSLT properties	Resource base Publish path

Typically you share style (CSS), layout (XSLT), and control definitions (JS) for use by all clients; however, if you use dynamic DataWindows customized for specific clients, you can force generation of the DataWindow presentation-related document names to be specific to each client. You do this by selecting the Session Specific CSS, XSLT and JS Filenames check box or by setting the CSSGen.SessionSpecific property to “yes”. This eliminates the possibility of server-side contention for presentation formats when the DataWindow generation is specific to the client.

For different DataWindows with the same name in the same application, you can eliminate the possibility of server-side contention for presentation formats and data content by entering a fully qualified file name (rather than a path) for the publish path properties of those DataWindows. If you do use a file name for a publish path property, the file extension must correspond to the type of format you are configuring. For example, if you are adding a file name to the publish path of the XML format, the file extension must be XML.

Controlling the size of generated code

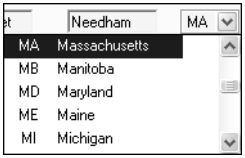
Some supported features increase the size of the generated code. If you do not use a feature such as display formatting, validation rules, or client-side scripting, you can enhance performance by preventing the server component from generating code for the unused feature. You can turn these features on or off on the Web Generation property page in the DataWindow painter or in a script. For more information, see "Controlling what is generated" on page 186.

You can also cache client-side methods in JavaScript files to reduce the size of the generated code and increase performance on both the server and the client. Without JavaScript caching, each time a Web DataWindow is rendered in a client browser, JavaScript code for DataWindow methods is generated on the server and downloaded to the client. When you set DataWindow properties to reference cached JavaScript files, the methods defined in the files are not generated with the HTML in any Web DataWindow pages that are sent to the page server (and client browser).

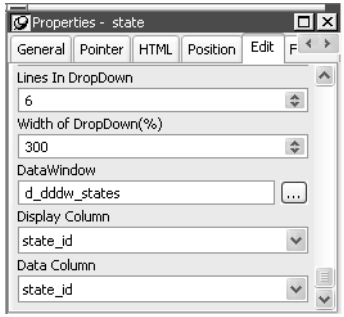
For more information, see "Using JavaScript caching for Web DataWindow methods" next.

Using drop-down DataWindows

When you tab to a column that uses the drop-down DataWindow edit style, you can use the arrow keys on the keyboard to change its value. If you click the column, the drop-down DataWindow displays so that you can scroll to a different value and click to select it.



You set the display properties for the column on the Edit page in the Properties view in the DataWindow painter. The Width of DropDown property sets the width of the drop-down display to a size that is a percentage of the width of the column. For example, 300 sets the display width to three times the column width.



The default behavior uses inline frames (iFrames), which increases the volume of markup generated. For DataWindow objects that make heavy use of drop-down DataWindows, you can save bandwidth by generating the drop-down DataWindows in HTML select elements. To do so, clear the Generate DDDW Frames check box on the Web Generation page with the Format to Configure option set to HTML/XHTML or set the HTMLGen.GenerateDDDWFrames property to "No".

For information about the HTMLGen.GenerateDDDWFrames property, see the *DataWindow Reference* or online Help.

Netscape and Mozilla browsers
The HTMLGen.GenerateDDDWFrames property is not supported in Netscape or Mozilla browsers. For information on browser support, see "Browser requirements for the XML Web DataWindow" on page 120.

Callback and client-side paging support

The PagingMethod property enables you to specify how paging requests are handled. The default setting is PostBack, which posts each page request back to the server. You can change the setting on the Web Generation page in the Properties view with the Format to Configure set to HTML/XHTML.

CallBack

CallBack paging uses a script callback feature to provide page navigation without posting the whole page back to the server. The XML data for the next requested page is downloaded as an XML string returned to the callback. A JavaScript function on the client collects the data and invokes the client-side XSLT processor to transform the data using the XSLT stylesheet that was downloaded and cached on the first request. The next page of data is displayed on demand. If you set the PagingMethod property to CallBack, you do not need to write server-side code and client-side JavaScript to take advantage of the script callback feature.

For the XML rendering format, the design of the Callback! option requires that a reusable XSLT stylesheet be generated so that the browser can cache it. The benefit from this requirement is that only the XML data for the next requested page need be returned by the callback. This XML data is always trivial in size (about a 1 to 20 ratio), resulting in significant bandwidth savings. This is unlike other implementations, where the entire presentation is always regenerated and downloaded again from every callback.

The generated XSLT stylesheet is not reusable, and therefore cannot be cached by the browser, if the DataWindow layout is inconsistent page-to-page, or it does not contain a complete first page of data. In these scenarios, the Callback! option defers to PostBack! until a stylesheet can be generated that is reusable, and can therefore be cached in the browser.

XMLClientSide

When the PagingMethod property is set to XMLClientSide, the page takes slightly longer to load on the first request because all the data in the result set is pulled down to the client, but subsequent paging requests take place entirely on the client. InsertRow, AppendRow, and DeleteRow actions take place on the client with no postback or callback to the server. However, any computed fields in the DataWindow that rely on the RowCount method are not reevaluated until the user performs an action such as an Update or Retrieve that forces a postback to the server.

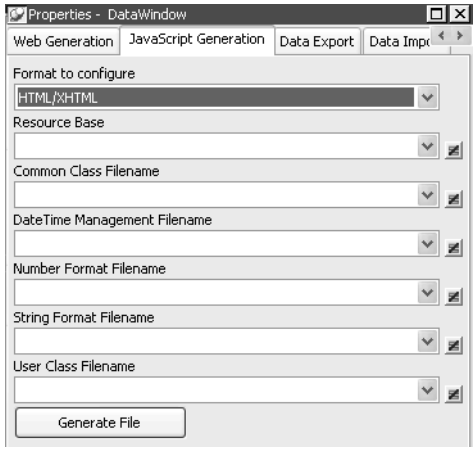
Client-side paging is available only for the XML rendering format and in button actions and client JavaScript paging functions of the Web DataWindow client control.

Using JavaScript caching for Web DataWindow methods

You can use the Web Data Window JavaScript Generator wizard to create JavaScript files (at design time) that contain the JavaScript client-side methods. You can start the Web Data Window JavaScript Generator wizard by clicking the Generate File button in the JavaScript Generation property page of the DataWindow property view or from the Tool tab of the New dialog box.

Shared JavaScript generation properties

These JavaScript files (HTML/XHTML) are shared by all Web formats of the Web DataWindow—XML, XHTML, and HTML.

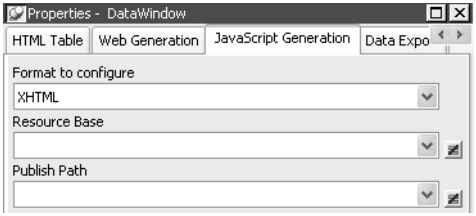


Each pass of the wizard generates only one file—which allows you to combine or separate classes of functions. Once you have generated one or more JavaScript files, you can attach them to a DataWindow object using the Filename drop-down lists (for Common Class, Date Time Management, Number Format, String Format, and User Class) in the JavaScript Generation property page.

Generating and associating JavaScript files with a DataWindow object enables the JavaScript functions to be cached and then reused each time the page containing the DataWindow object displays in the browser.

XML and XHTML JavaScript generation properties

To configure the JavaScript generation properties that are only for the XML Web DataWindow and XHTML Web DataWindow, you select XHTML and provide the resource base and the publish path:



Improving server-side and client-side performance

When you set new DataWindow properties to reference included JavaScript files, the methods defined in the referenced files are not generated with the HTML in any Web DataWindow pages that are sent to the page server and client browser. Using JavaScript files also reduces the size of the HTML page rendered in the browser.

With JavaScript caching, you improve performance on the client machine as long as the client browser is configured to use cached files. With caching enabled, the browser loads the JavaScript files from the Web server into its cache, and these become available for all the Web DataWindow pages in your application. There is no client-side performance gain if the browser does not find the JavaScript files in its cache since, in this case, it reloads the files from the Web server.

Web DataWindow JavaScript Generator wizard

With the Web DataWindow JavaScript Generator wizard, you can generate only one JavaScript file at a time. The wizard gives you the option of including all Web DataWindow methods in a single file, but you can also restrict the types of methods to include in each JavaScript file it generates every time you use the wizard. The different method types correspond to the following DataWindow HTML properties:

Table 6-7: Methods generated by JavaScript Generator wizard in cached files

HTMLGen.property	Contents of cached file
CommonJSFile	Methods used by all DataWindows.
DateJSFile	Methods used by DataWindows with date and time formatting.
NumberJSFile	Methods used by DataWindows with number formatting.
StringJSFile	Methods used by DataWindows with string formatting.
UserJSFile	User-defined client-side JavaScript methods—these cannot be generated by the Web DataWindow JavaScript Generator wizard (see “User-defined JavaScript methods” on page 137).

All of these properties are optional. You can set each of the properties from the JavaScript Generation page of the DataWindow property view, selecting the files you generate with the wizard as values. The wizard registers each file it generates, making it available for selection from the drop-down lists in the DataWindow property view.

Using the ResourceBase property

You must deploy all cached files for your Web application to your Web server. You can use relative URLs or path names for cached JavaScript files if you specify their location in the HTMLGen.ResourceBase property.

You set these on the JavaScript Generation page of the DataWindow property view in the DataWindow painter. The ResourceBase property is also used to specify the location of image files.

If you do not set the HTMLGen.ResourceBase property, you must include the complete URL in the values of any of the HTMLGen properties that you set. In either case, the URLs are rendered as SRC attributes inside SCRIPT tags in the pages generated by the Web DataWindow component and sent to the client browser.

Setting the properties in script

You can customize the DataWindow HTML Generator component (nv_remote_datawindow in *PBDWRMT.PBL* that ships with PowerBuilder), setting the HTMLGen properties in the script for the Generate method. This example sets the URL location for included files and names the files for common and date-formatted Web DataWindow methods that you deploy to the Web server (and that will be downloaded to browser clients the first time they connect to the Web site):

```
ids_datastore.Modify &
    ("DataWindow.HTMLGen.ResourceBase=" +&
    "'http://www.myserver.com/JavaScripts'")
ids_datastore.Modify &
    ("DataWindow.HTMLGen.CommonJSFile=" +&
    "'dwcomn.js'")

ids_datastore.Modify &
    ("DataWindow.HTMLGen.DateJSFile=" +&
    "'dwdate.js'")
```

User-defined JavaScript methods

You can also reference a file where you store your own client-side JavaScript methods. To use this feature, you must assign the name of the file to the DataWindow HTMLGen.UserJSFile property and make sure the file is available to your Web server. As for the wizard-generated JavaScript files, you can use the HTMLGen.ResourceBase property to set the location for the file, or you can include the complete path to the file in the property value assignment.

You can make this assignment in the DataWindow painter or in script. The following script sets the user-defined JavaScript file to *MyMethods.JS*:

```
ids_datastore.Modify &
    ("DataWindow.HTMLGen.UserJSFile=" +&
   "'http://my_server.com/JavaScripts/MyMethods.JS'")
```

This example will be rendered in the generated HTML page as:

```
<SCRIPT LANGUAGE="JavaScript" SRC=
"http://my_server.com/JavaScripts/MyMethods.JS">
</SCRIPT>
```

You can then call client-side methods stored in the *MyMethods.JS* file from the HTML syntax rendered for (or appended to) controls in a DataWindow object. For information on generating or appending HTML syntax to controls, see "Rendering HTML for controls in an HTML Web DataWindow" on page 140.

Using expressions

In general, expressions for validation rules and computed fields are translated into JavaScript and evaluated in the client browser. For validation of data entry, the user gets immediate feedback on the new data.

Some expressions have to be evaluated on the server. This might be because the evaluation involves all the rows, rather than data on the current page only, or because the expression does not translate into JavaScript.

If an expression includes these functions, it will be evaluated on the server:

- Aggregation functions, like Sum, Max, Average, First
- Case function
- External functions

If you use an aggregation function in a computed field, the value is computed on the server and displayed on the client. If the user edits data, the value is not updated. If an action occurs that reloads the page, the value is computed again based on the changed data.

ProfileInt and ProfileString return default values
The ProfileInt and ProfileString DataWindow expression functions do not examine a user's INI files if you use them in an expression evaluated on the client. Doing so would be a security violation. They always return the default value.

Using foreign language character sets

For static HTML pages

If a data source for your Web DataWindow uses foreign characters with accent marks, you might need to change the character set for the generated Web page to display the characters properly. This is also necessary if you expect to update the data with foreign character text. You can use the HTML editor to edit the <META> tags in the <HEAD> section of the generated page. The following example uses the utf8 character set instead of the default iso-8859-1 character set:

```
<META content="text/html; charset=utf-8"
http-equiv="Content-Type">
```

Changing the character set in EAServer
You can also change the character set used by your component in EAServer by setting the com.sybase.jaguar.component.code.set property for the component. By default, the component uses the server's codeset.

For JSP pages

For JSP pages, you assign the character set for the response object in a page directive:

```
<%@ page contentType="text/html; charset=UTF-8" %>
```

For the JSP request object, you must assign the character set in a server-side setCharacterEncoding call:

```
<% request.setCharacterEncoding
  (contentType="text/html; charset=UTF-8"); %>
```

Providing links for data

You can set properties that apply to all Web DataWindow formats that cause columns, text, computed fields, and Picture controls to be hyperlinks. In the painter, select the control and choose the HTML tab in the Properties view. The URL you specify must be valid when you deploy your application.

Rather than set link information in the painter, you can set the HTML properties for columns by calling methods of the server component. For information, see the SetColumnLink method in the *DataWindow Reference*.

Rendering HTML for controls in an HTML Web DataWindow

For HTML Web DataWindow only
This section applies to the HTML Web DataWindow only. XML Web DataWindow and XHTML Web DataWindow features supersede the need to perform any of the following actions.

Including HTML in a control

You can include valid HTML in some controls in a DataWindow object, including a text control, column, or computed field. To render the contents of the control as HTML when the HTML for the DataWindow is generated, set the control's ValueIsHTML property to TRUE. For example, suppose a text control's text property is <I>Name</I>. The following table shows how the text is rendered in the generated HTML and displayed in a browser.

Table 6-8: Effect of ValueIsHTML property on rendered text

ValueIsHTML	Generated HTML source	Output in browser
TRUE	<I>Name</I>	*Name*
FALSE	<I>Name</I>	<I>Name</I>

No validation
The HTML Generator does not validate the HTML you include in controls in DataWindow objects. If the HTML is not valid, the DataWindow might not display correctly.

Appending HTML to a control

The AppendedHTML property enables you to append your own HTML to the HTML generated by the HTML Generator component. You can use this feature to specify attributes and event actions. The HTML you specify for the AppendedHTML property value is appended to generated syntax for the rendering of a DataWindow control before the closing bracket of the HTML element for that control.

No validation
The HTML Generator does not validate the HTML you append to controls in DataWindow objects. If the HTML is not valid, the DataWindow might not display correctly.

You must also make sure not to use an event handler name that is already generated for a DataWindow control as a client-side event handler. These include the event handlers in Table 6-9.

Table 6-9: Generated event handler names

DataWindow control	Generated event handler names
Edit, EditMask, DropDownListBox, or DropDownDataWindow	onFocus, onClick, onChange, and onBlur
CheckBox or RadioButton	onFocus, onClick, and onBlur
TextBox, Picture with link, or Button	onClick

Using Button and Picture controls

Using Button controls

When a DataWindow object includes a Button control, the button becomes an HTML or XHTML button element in the Form element for the Web DataWindow client control. The button action becomes JavaScript code for the button's Clicked event. You do not need to write any code yourself.

You can use Button controls for:

- **Navigation** Buttons with the PageFirst, PageLast, PageNext, and PagePrior actions let the user scroll to other rows in the result set.
- **Getting and editing data** Buttons with Retrieve, Update, InsertRow, DeleteRow, and AppendRow actions let the user maintain data. There must be updatable columns in the DataWindow object.

These button actions are not supported and are ignored:

Cancel	QueryClear
Filter	QueryMode
Preview	QuerySort
PreviewWithRulers	SaveRowsAs
Print	Sort

Setting SelfLink properties to enable navigation buttons

Button actions send information back to the server, whose scripts apply the action to the DataWindow data. Then the Web page is reloaded. To complete this loop, you must set the SelfLink property for the DataWindow object so that the server knows what page to reload.

You can set this property in the DataWindow painter on the Web Generation tab in the DataWindow properties view, or you can set it in a server-side script. The value is the name of the application server template or file to be reloaded—generally, the name of the current document. If the DataWindow uses retrieval arguments, they must be provided in the SelfLinkArgs property.

For more information, see “Passing page-specific data to the reloaded page” on page 189 and the SetSelfLink method in the *DataWindow Reference*. For information on setting links on a DataWindow object (such as a button or column) in the PowerBuilder HTML editor, see the chapter on the Web DataWindow DTC in *Working with JSP Targets*.

GIF and JPEG images for buttons

The picture on a button in a DataWindow object can be rendered in the Web browser as a JPEG, GIF, or BMP image. Use a JPEG or GIF image to ensure that the image will display on all browsers. PowerBuilder provides GIF images for commonly used buttons such as Retrieve, Update, PageNext, and so on. These pictures are included in the *DWACTION.JAR* file in the Sybase\Shared\PowerBuilder directory.

To make the images available to the Web page in the Web browser, you must uncompress the JAR file, deploy the image files to the page server, and set the HTMLGen.ResourceBase property to the directory where the files are located.

Alternative to buttons: use methods of the client control

If you want to use an existing DataWindow object that does not have Button controls, you can edit the DataWindow object and save a new version with Button controls. However, if you are sharing DataWindow objects with an existing application and it is not practical to edit them, your Web page can include HTML or XHTML buttons that call methods of the Web DataWindow client control.

There are methods of the client control that correspond to each of the supported button actions. For information, see “Writing client-side scripts” on page 154.

Using Picture controls

You can use any image types the browser supports, most commonly JPEG or GIF. Use relative paths for ease of deployment.

To make sure the images are available to the Web page in the browser, place the image files in a directory on the Web server and then set the HTMLGen.ResourceBase property to that directory. You can do this in the DataWindow painter on the JavaScript Generation page of the DataWindow property view, or in a script:

```
dwMine.Modify("DataWindow.HTMLGen.ResourceBase=
       'C:\Sybase\MyApp\Images'")
```

The ResourceBase property also specifies the location of JavaScript include files. See “Using JavaScript caching for Web DataWindow methods” on page 135.

Where to deploy image files
The image files need to be deployed to the Web server, not the component server. If these servers are on different computers, the files belong with the templates and Web files of the application, not the PBL containing the DataWindow objects.

Specifying Web generation for a specific browser

About browsers and HTML version

The Web DataWindow can generate XHTML and HTML optimized for different browsers. You can use the Browser choice for the XHTML and HTML formats on the Web Generation tab of the DataWindow property sheet to preview what the DataWindow looks like in different browsers. You can also specify an HTML version that the Web generation should use if it does not recognize the browser.

At runtime, a server-side script should find out what browser the current client is using and pass that information to the server component. For information, see "SetBrowser" on page 187 and the SetBrowser method in the *DataWindow Reference* or online Help.

Using absolute positioning in Netscape

Netscape implements absolute positioning differently than Internet Explorer. To format the DataWindow with absolute positioning for Netscape browsers, you must set the DataWindow HTMLGen.NetscapeLayers property to *true*. You can do this in the DataWindow painter by selecting the Use Layers for Netscape check box on the Web Generation page of the DataWindow property view. Alternatively, you can use a DataWindow Modify call in the script for the Generate method of the HTML Generator (nv_remote_datawindow in *PBDWRMT.PBL* that ships with PowerBuilder).

The default DataWindow HTML Generator generates code for determining the browser type and version of the client browser. For Netscape browsers earlier than the 4.0 version, the DataWindow is formatted as an HTML table, whether or not the NetscapeLayers property is set.

Limitations in Netscape
Certain functionality in a Netscape browser using absolute positioning might not be identical to the functionality available with Internet Explorer. For example, you cannot tab between DataWindow columns using a Netscape browser on an NT machine, although you can do this using a Netscape browser on a Solaris machine.

Previewing the DataWindow

To see what the DataWindow will look like in a Web DataWindow application, you can use HTML Preview.

❖ **To get an HTML preview of a Web DataWindow:**

1. On the General property page of the DataWindow property sheet, check Web DataWindow.

   If you do not check Web DataWindow, the preview displays the data as an HTML table without buttons, validation rules, or other scripts.

2. On the Web Generation page, specify a value for Rows per Page.

   This sets the PageSize property for the DataWindow object. To display only one row of data, specify 1.

3. Specify a value for Browser and one for Version if you want to preview the DataWindow for a specific client configuration.

4. Select Design>HTML Preview from the menu bar.

   If the menu item is disabled, open the Preview view to enable it.

5. Enter data and see whether validation rules behave as expected.

6. Use your buttons to navigate to other pages.

Setting Web DataWindow design-time control properties

The previous section, “Designing DataWindow objects for the Web DataWindow,” began on page 123 and covered many topics about the design of DataWindow objects that are to be used as Web DataWindows. Now we look at the integrated support in PowerBuilder for creating and editing Java Server Pages (JSPs) for the Web DataWindow.

JavaServer pages

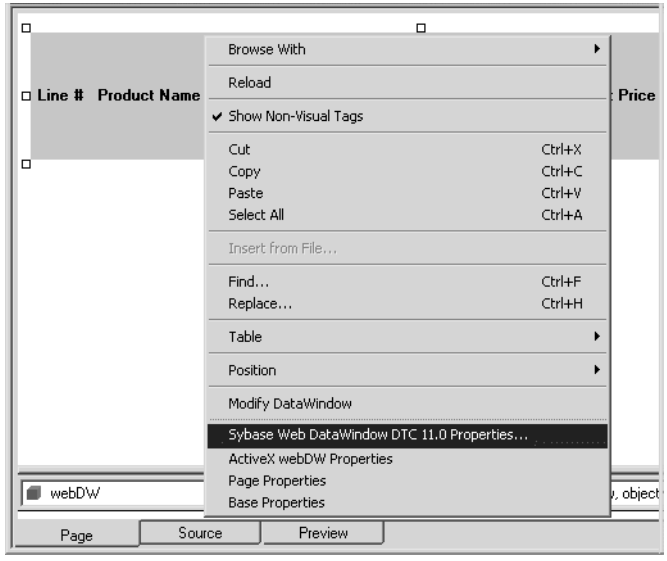
JSP pages can be written in any well-formed language, including XML, but they are usually written in HTML. In PowerBuilder, you usually create JSPs for use with Web DataWindows using the Web/JSP DataWindow Page wizard, and you edit them in much the same way as any other HTML page, using the HTML editor. In the Web/JSP DataWindow Page wizard, you specify whether you are using the Standard HTMLGenerator110 component, a custom HTMLGenerator110 component that you developed and deployed to EAServer, or a DataWindow container.

DTC properties

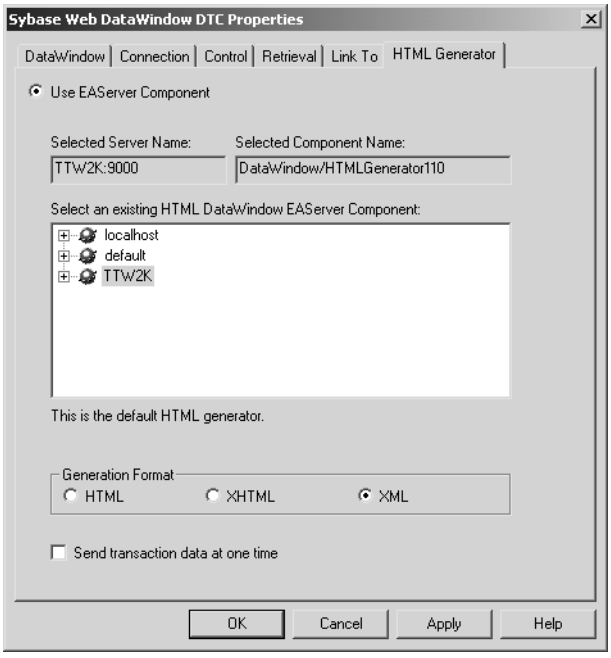
When the page is created, it contains a Web DataWindow design-time control (DTC). A Web DataWindow DTC lets you add database-driven content to Web applications.

In the Sybase Web DataWindow DTC Properties dialog box, you can specify the type of generation you want. The generation type determines the method that PowerBuilder calls in the HTMLGenerator110 component to generate an XML, XHTML, or HTML Web DataWindow.

You specify the type of generation in the DataWindow design-time control by selecting the DataWindow (JSP) in the Page tab, right-clicking the design-time control, and selecting Sybase Web DataWindow DTC 11.0 Properties:



Then in the HTML Generator tab, you select the generation format—HTML, XHTML, or XML:



For example, here the selected generation format is XML. This results in the generation of separate content (XML), layout (XSLT), and style (CSS), and a resulting transformation to XHTML. Setting the Generation Format property to XML is required for the XML Web DataWindow.

Also specify the rows per page for the XML or XHTML DataWindow
For optimal bandwidth savings when you use XML or XHTML as the generation format, you must also specify the rows per page in the Control tab.

For more information

For detailed information about working with JSPs and using the Web DataWindow DTC, see the *Working with JSP Targets* book.

Setting up database connections

When you use the Web DataWindow, it is the Web DataWindow server component that interacts with the database, so you need to set up database connections on the server where the component is running.

What database connectivity software to use

If you are using EAServer as the component server, you can use several types of connectivity software, including ODBC, Open Client, JDBC, and OCI. You need to set up a connection cache for the data source you are using. See "Creating a connection cache on EAServer" next.

If you are using COM+ as the component server, you must use ODBC to take advantage of connection pooling and transaction management features. The data source for your DataWindow objects must be configured as a system DSN on COM+ because it runs as a service.

When you are defining DataWindow objects, you can use the types of connectivity software documented in *Connecting to Your Database*. To simplify setting up database connections on the server, use the same type of connection in the DataWindow painter that will be used when the DataWindow is deployed. For example, if you plan to use the DataWindow object in EAServer with an Open Client connection, use Open Client in the DataWindow painter too.

Using Adaptive Server Enterprise
PowerBuilder and EAServer use slightly different versions of the CT-Lib software to connect to Adaptive Server Enterprise through Open Client. In the PowerBuilder development environment you use the SYC native database interface to connect to the database, but to connect to an Adaptive Server Enterprise database in EAServer you must use the SYJ database interface.

Creating a connection cache on EAServer

To use the HTMLGenerator110 component (or your own custom component) in EAServer, you need to define a connection cache for the database it will use.

❖ **To create an EAServer connection cache:**

1 Start EAServer and EAServer Manager.

Instructions are in the *EAServer Getting Started* book for your platform.

2 Define the connection type you want, using the appropriate database configuration software.

   For an ODBC connection on Windows, for example, open the ODBC control panel and define a Data Source Name (DSN) for the database. If you installed SQL Anywhere 10, you can use the predefined user DSN to connect to the SQL Anywhere Sample database for testing purposes. If EAServer is running as a service, you must use a system DSN, not a user DSN. On UNIX, see your driver documentation.

3 In EAServer Manager's left pane, right-click the Connection Caches node under the server name and select Install Connection Cache.

4 In the dialog box, select Create and Install a New Connection Cache.

5 Enter a name for the cache and click OK. (You cannot change this name.)

6 In the Connection Cache Properties dialog box, specify database connection information on the General tab and the driver you want to use on the Driver tab.

   For ODBC on Windows: for Database Connection:Server Name on the General tab, specify the DSN you defined in step 1. You do not need to specify a user name or password if they are specified as part of the DSN.

   On the Driver tab, click the ODBC radio button and specify ODBC32.dll for the Driver Class or File.

   For other connection types: see the *EAServer Getting Started* book for your platform.

7 Select the Enable Cache-by-Name Access check box on the Cache tab.

8 Click Refresh on the General tab (the Refresh button is available only on the Connection Cache Properties dialog box accessed from the server node).

9 To test the connection, click Ping.

For more detailed information about setting up connection caches, see the EAServer documentation.

Deploying DataWindow objects to the component server

When you run a Web DataWindow application, the definitions of your DataWindow objects must be available on the component server. You can ensure this by copying the PBL or PBD file that contains the definition of the objects to the server. If you are using EAServer, the files must be in the server's path, or if the server component is running as a service, in the system path. COM+ always runs as a service, so the files must be in the COM+ server's system path.

For more information about setting up your development environment, see "Server configuration requirements" on page 177.

The Web DataWindow Container project wizard

In PowerBuilder, you can use the Web DataWindow Container project wizard to create a project that deploys a custom version of the Web DataWindow server component (HTMLGenerator110) to EAServer with all the DataWindow objects in your library list built into the deployed PBD.

Because you are using a custom component instead of the preinstalled HTMLGenerator110 component, you can set properties for your component in EAServer Manager. Setting properties reduces the number of method calls required to configure the component and can result in improved performance, maintainability, and scalability.

Using the wizard

When you have defined the DataWindow objects you need, start the Web DataWindow Container wizard from the Project page of the New dialog box and follow the instructions in the wizard.

The choices you make in the wizard are similar to those you make for other EAServer components, such as whether to support instance pooling or live editing. The package and component can have any names you want, with the exception that you cannot name the package *DataWindow* (the name of the package that contains the preinstalled HTMLGenerator110 component). You cannot set component type or transaction support properties.

Building the Web DataWindow Container project presets some of the properties of the component that you would otherwise need to set in scripts or in EAServer Manager. You can specify a database profile in the wizard or Project painter to set the com.sybase.datawindow.trans.dbms and other database transaction properties. If you want calls to component methods to be written to the server log, select Enable Trace to set the com.sybase.datawindow.trace property.

Connection cache
Specifying a profile sets database transaction properties for the component, but you still need to create a connection cache in EAServer Manager for the database the component will access.

Service classes

You can add a custom class user object to your project to perform any special processing you require. This object acts as a service class. The custom class user object must be in the same library as the DataWindow objects or on the target's library list. When you build the Web DataWindow Container project, select the class in the Select Objects dialog box in the Project painter.

For more information, see "Using service classes" on page 203.

Instantiating the container component

When you create an instance of the container component, you must either provide a Java proxy for the component or pass DataWindow/HTMLGenerator110 as the last argument to the CreateComponent method. This specifies that the component uses the proxy provided for the generic component. For example:

```
dwObj = java.CreateComponent
       ("ContainerPkg/ContComponent",
       "iiop://testMachine:9000", "Jagadmin", "",
       "DataWindow/HTMLGenerator110" );
```

This is the same way you instantiate custom components. For more information on custom components, see "Instantiating the custom component" on page 200.

Selecting a DataWindow from the container component

To select a DataWindow from the container component, you must use the SetDWObjectEx method rather than the SetDWObject method:

```
//Only the DW is needed, as package already specified
//above when connecting
retVal = dwObj.SetDWObjectEx("d_mydw");
document.write("SetDwObject = " + retVal);
```

For more information on the SetDWObject method, see "Loading the DataWindow object" on page 185 and the *DataWindow Reference*.

Using the component with the Web DataWindow DTC

You can select the container component when you use the Web/JSP DataWindow Page wizard to create a new Web page. On the Choose EAServer Profile page in the wizard, check the Use Custom Web DW Component check box. Then, when you click Next, the wizard prompts you for the component name and the DataWindow in the component that you want to include on your Web page.

On the DataWindow page of the Web DataWindow DTC property sheet, you can also specify a container component and the DataWindow in the container component that you want to use on a Web page. Selecting a container component on the DataWindow page automatically selects it on the HTML Generator page of the DTC property sheet. (If you want, it is possible just to use the container component to generate XHTML or HTML for a DataWindow that you select from a PBL, PSR, or SRD file).

❖ **To use the container component with the Web DataWindow DTC**

1. Select the Web DataWindow Container Component radio button on the DataWindow page of the Web DataWindow DTC property sheet.
2. Click the browse button (...) next to the Component text box.

   The Web DataWindow Container Component dialog box displays. It has a list box with nodes for each EAServer profile defined in PowerBuilder.
3. Expand the node for the server and package that contains the container component you want to use.
   Select the container component and click OK.

   The Web DataWindow DTC property sheet redisplays. The server name and the container package and component are displayed in the Server and Component text boxes on the DataWindow page of the property sheet.
4. Select a DataWindow from the DataWindow drop-down list.

   The drop-down list displays all the DataWindows in the container component.
5. (Optional) Click the Connection tab and override the database connection property stored by the component.

   This is recommended only for testing purposes. If you want to change the connection property, you can do it directly from EAServer Manager.
6. (Optional) Click the Control tab and override the weight settings stored with the DataWindow.
7. Click OK.

Creating Web page templates

Different approaches to creating Web page templates are described below. You can use all of the following techniques in a JSP target.

Using the Web DataWindow DTC

The Web DataWindow DTC makes it easy to add a DataWindow to a Web page without writing the code required to connect to EAServer and configure the server component. The DTC accesses the server and creates a Web page template for the generated HTML. You set up the DTC to reference your DataWindow object and the HTML Generator component in a wizard or on a property sheet.

The Web DataWindow DTC embedded in the page includes the code required to connect to EAServer, establish database connections, and render the page as HTML. The generated scripts use the Web Target object model. If you use a 4GL Web page, you can use the EAServer extensions to server-side scripting that are built into the Web Target object model. You cannot hand-code a Web DataWindow on a 4GL Web page.

For how to use the DTC, see *Working with JSP Targets*.

Using the Web Target object model

The Web DataWindow DTC uses a set of objects in the Web Target object model. If you want more flexibility than the Web DataWindow DTC provides, you can write your own server-side scripts to instantiate and manipulate the server component using the Web Target object model.

For example, the following script defines the DataWindow object name and the PBL that contains it, sets up database and EAServer connection parameters, instantiates the server component, specifies client control properties, and generates HTML:

```
String dwMine_pbl = "htgenex.pbl";
// Get DataWindow name from page parameter if defined
String dwMine_name = psDocument.GetParam("dwName");
if (dwMine_name + "" == "undefined"){
       dwMine_name = "d_tabular_emp";
}
// Set up DataWindow object variables
PSDataWindowSourceClass dwSrcLoc;
dwSrcLoc = new PSDataWindowSourceClass(
       dwMine_pbl, dwMine_name);

// Set database connection parameters
// Set ConnectOption to return an error if
// connection fails
PSConnectionParmsClass dwDbParm;
```

```
dwDbParm = new PSConnectionParmsClass(
       "ConnectString='DSN=EAS Demo DB V110;UID=dba;
       PWD=sql', ConnectOption='SQL_DRIVER_CONNECT,
       SQL_DRIVER_NOPROMPT'");

// Set EAServer connection parameters
PSJaguarConnection dwJagParm;
dwJagParm = new PSJaguarConnection("myServer:9000",
       "jagadmin", "", "DataWindow/HTMLGenerator110",
       false);

// Create the Web DataWindow server component
PSDataWindowClass dwMine;
dwMine = new PSDataWindowClass(pageContext, request,
      "dwMine", false, dwJagParm, dwSrcLoc,
      dwDbParm,0);

// Specify what features to include in client control
dwMine.SetWeight(true, true, true, true, true);

// Get generated HTML
dwMine.Generate("dwName");
```

The example above uses the PSDataWindowClass object to create the server component. For EAServer, you can also use the PSWebDataWindowClass object, which provides server-side events and additional methods.

For more information about the Web DataWindow DTC and the Web Target object model, see *Working with JSP Targets*. For information about methods and events of Web Target object model classes, see the *JSP Target Reference*.

Extending the DTC code

The code generated by the Web DataWindow DTC can be modified by changing DataWindow object properties in the DataWindow painter or changing DTC properties and then regenerating the code. If you want to include additional customizations, you can modify the generated code in the editor's Source view. To prevent your customizations from being overwritten when the Web page is regenerated, remove the METADATA tags that surround the code.

Calling the Web DataWindow server component methods

Instead of using the DTC or Web Target object model, you can use any HTML or JSP editor (including PowerBuilder's HTML/JSP editor) to write code that instantiates the server component and calls its methods directly. See "Instantiating and configuring the server component" on page 182.

Writing client-side scripts

Responding to events

If you want to provide additional processing of newly entered data or have more control over user interactions with the data, you can choose to enable events in the Web DataWindow client control. To do so, you set the Client Events property on the Web Generation page in the DataWindow painter or call the SetWeight method in a server-side script.

The client control supports several events:

ButtonClicking	ItemError	RowFocusChanging
ButtonClicked	ItemFocusChanged	UpdateStart
Clicked	OnSubmit	
ItemChanged	RowFocusChanged	

Most of these events have similar arguments and the same return codes as the events of the PowerBuilder DataWindow control. For information, see the *DataWindow Reference* or online Help.

Implementing an event

To write a script for an event of the client control, you define a function whose name is the control name plus the event name, separated by an underscore:

HTMLGenObjectName_eventname (*arguments*)

The control name is the one you specified using the SetHTMLObjectName method or the Object Name property on the Web Generation page in the DataWindow painter. The script must be enclosed in SCRIPT tags. You can include client methods in the script if client scripting is enabled (described next).

This example prevents focus from changing if the user tries to go back to an earlier row. In this case the name of the DataWindow control is dwMine:

```
<SCRIPT Language=JavaScript>
function dwMine_RowFocusChanging(curRow, newRow)
{
       if (newRow < curRow) { return 1; }
}
</SCRIPT>
```

You can put the script anywhere in your Web page template. If you are using a Web or JSP target, you do not need to define a function to wrap your event code. You can code client-side events directly in the editor by selecting the DataWindow control name in the leftmost drop-down list in the toolbar and the event name in the center drop-down list.

Calling client methods

To write scripts that call methods of the client control, you must enable client scripting. To do so, you can set the Client Scriptable property in the DataWindow painter or call the SetWeight method in a server-side script.

Several client methods accomplish the same tasks as actions of Button controls. If your DataWindow object uses Button controls to implement scrolling and updating, you might not need to do any client scripting.

You can use the following methods on the client (methods marked with an asterisk force the Web page to be reloaded):

AcceptText	GetItem	ScrollNextPage *
DeletedCount	GetItemStatus	ScrollPriorPage *
DeleteRow *	InsertRow *	SelectRow
GetClickedColumn	IsRowSelected	SetItem
GetClickedRow	ModifiedCount	SetColumn
GetColumn	Retrieve *	SetRow
GetFullContext	RowCount	SetSort
GetNextModified	ScrollFirstPage *	Sort *
GetRow	ScrollLastPage *	Update *

GetNextModified
The GetNextModified method finds modified rows in the current page only.

For information about these methods, see the *DataWindow Reference* or online Help.

This example includes a form with a button that causes data to be updated on the server:

```
<FORM NAME="update">
<INPUT type="button" value="Update"
onClick="{dwMine.Update();}">
```

Note that you can get the same functionality with the Update action for a Button control in the DataWindow object.

Multiple DataWindows on a page

If you have multiple updatable Web DataWindows on the same Web page, you can script the OnSubmit client-side event to synchronize them before the changes on the page are submitted. You call the GetFullContext method to get a string that represents the context of the client side control that would be passed on a submit, and transfer the context to the other DataWindow control.

To enable the second DataWindow to create the required fields on the submit form, each of the DataWindows must have two arguments defined as self-link arguments:

```
dw_1.SetSelfLink(document.name,
        "dw_2_context=''|dw_2_action=''")
dw_2.SetSelfLink(document.name,
        "dw_1_context=''|dw_1_action=''")
```

This client-side script transfers the context and action from dw_2 to dw_1 when dw_1 is submitted, and from dw_1 to dw_2 when dw_2 is submitted:

```
<SCRIPT>
    function dw_1_OnSubmit()
    {
        dw_1.submitForm.dw_2_context.value =
            dw_2.GetFullContext();
        dw_1.submitForm.dw_2_action.value = "";
    }

    function dw_2_OnSubmit()
    {
        dw_2.submitForm.dw_1_context.value =
            dw_1.GetFullContext();
        dw_2.submitForm.dw_1_action.value = "";
    }
</SCRIPT>
```

Customizing Web DataWindow generation

You can customize the XHTML that is generated at runtime by the XML Web DataWindow and the XHTML Web DataWindow using an XHTML export template in the DataWindow painter's Export Template view for XHTML.

XML and XHTML Web DataWindow customization
This section focuses on customizing the XHTML generated by the XML Web DataWindow.

The Export Template view for XHTML

Each DataWindow object that you create has a default XHTML export template associated with it. You can see the default template in the DataWindow painter's Export Template view for XHTML.

Displaying the XHTML export template
The Export Template view for XHTML coexists with the Export Template view for XML, each on its own tab page with XML on the top by default. To display the view for XHTML, click the XHTML tab. If you have any problems displaying the view, select View>Export/Import Template>XHTML from the menu bar or select View>Layouts>Default and then click the XHTML tab.

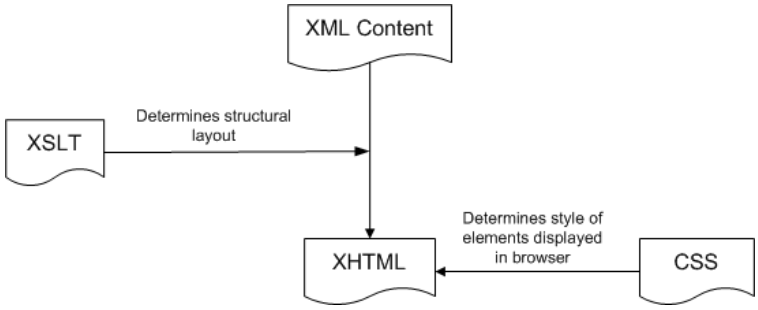
The XHTML export template is a single-instance document of the <form> element. It stores only the structural layout and any changes that you make to the elements, attributes, and style declarations. When XHTML or XSLT is generated, these changes are incorporated into the <form> element and the CSS stylesheet used to render the DataWindow in the browser. More than one export template can be created for a DataWindow.

Default style rules and most default attributes are not stored in the template. Any changes to style declarations are stored in the template, but at runtime they are removed and applied to the separately generated CSS stylesheet.

In the Export Template view for XHTML, you can reference DataWindow column, computed field, and text controls, and DataWindow expressions for each row in the XHTML, wherever character data is allowed. At runtime, these are replaced with text.

What you can customize

The XML Web DataWindow generates DataWindow content, layout, and style separately at runtime and renders in the browser a fully-functional DataWindow in XHTML.



At design time, you can customize each of these XML Web DataWindow components:

- Elements or contents of the <form> element
- CSS stylesheet declarations
- Other XHTML element-specific attributes (including style attributes) in the DataWindow form
- JavaScript event handlers

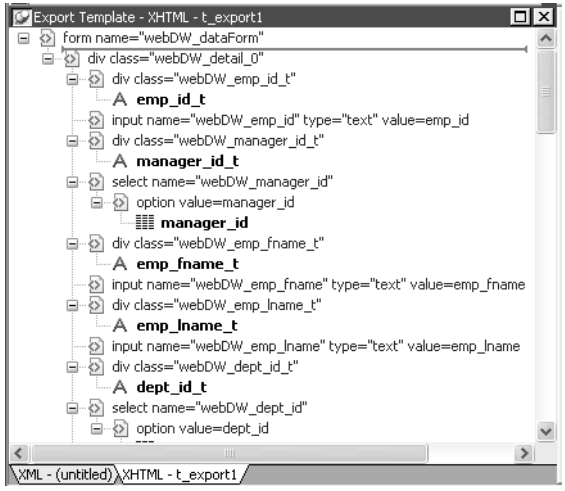
Examples of customization

Your customizations can include these types of modifications:

Customization	Example
Structural layout	Add or remove elements and content (input fields of the XHTML <form> element) from the header, detail, summary, and footer bands
Style rules of data input field elements in the <form> element	Modify the default CSS stylesheet property values and add or remove CSS stylesheet declarations
Other attributes of elements of the DataWindow	Override attribute values and remove or add attributes specific to these elements
JavaScript event handlers	Override, redirect, add, or remove JavaScript event handlers

The default XHTML export template

In the default XHTML export template, export XHTML entities (markup and character data) are displayed as single tree view items that denote the type of entity. The default template has one element for each column in the DataWindow object:



You can create multiple templates and save them by name with the DataWindow object. Each template is uniquely associated with the DataWindow object open in the painter. For information, see “Managing templates” on page 160.

How tree view items are represented

Each item in the XHTML export template displays as a single tree view item with an image and font color that denotes its type. Elements are represented by a yellow icon that resembles a luggage tag. The end tags of elements and the markup delimiters (angle brackets) used in an XHTML document do not display.

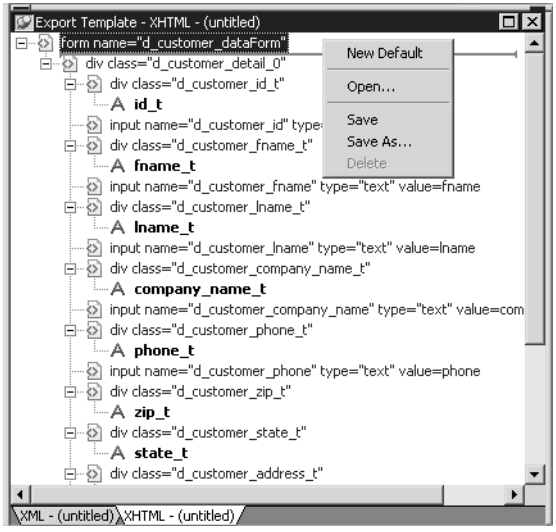
Table 6-10 shows the icons used in the Export Template view for XHTML.

Table 6-10: Icons used in the Export Template view for XHTML

Icon	Description
	Root or child element
	Group header element
	DataWindow column reference
	Static text control reference
	Computed field or DataWindow expression reference
	Literal text
	CDATA section
	Nested report

Managing templates

From the pop-up menu for the default XHTML export template (with no items selected), you can create multiple templates and save them by name with the DataWindow object open in the painter. You can also open and edit existing templates that are associated with the current DataWindow object and, when more than one template is associated with the DataWindow, delete the current template:



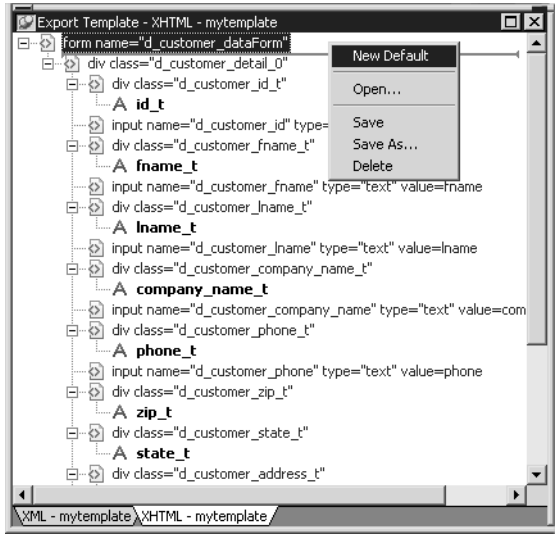
The pop-up menu has these options for managing templates:

Menu item	Description
New Default	Define a new default XHTML export template based on the current DataWindow layout
Open	Open a saved template
Save	Save the current template; if the template has no name, name it
Save As	Save the current template with a new name
Delete	Delete the current template (enabled only if more than one template exists for the current DataWindow object)

Creating and saving templates

Creating a new default template

To create a new default XHTML export template, select New Default from the pop-up menu in the Export Template view for XHTML.



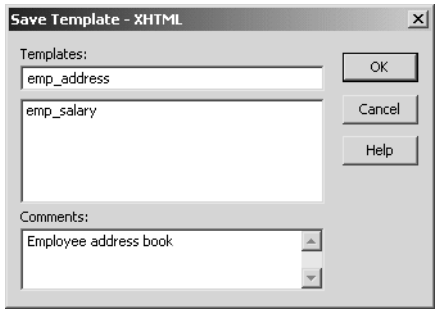
A new default XHTML export template has the following elements:

Elements	Name defaults to
Root <form>	*DataWindow name*_dataForm
Header	*DataWindow name*_band1
Detail	*DataWindow name*_band*n*
Summary	*DataWindow name*_band*n*

Elements	Name defaults to
Footer	*DataWindow name*_band*n*
Child elements of the Header, Detail, Summary, and Footer elements	Name of each DataWindow control.

Saving the template

To save a new default template, select Save from the pop-up menu in the Export Template view for XHTML, name the template, and provide a comment that identifies its use.



The template is stored inside the DataWindow object in the PBL. After saving a template with a DataWindow object, you can see its definition in the Source editor for the DataWindow object. For example, this is part of the source for a DataWindow that has two templates. The templates have required elements only:

```
export.xhtml(usetemplate = "t_phone"
template = (name = "t_address"
       comment = "Employee Address Book" xhtml = "<…>")
template = (name = "t_phone"
       comment = "Employee Phone Book" xhtml = "<…>") )
```

Defining multiple templates

You can define multiple templates for a single DataWindow object. One reason you might do this is to vary the edit styles generated for the same DataWindow edit control.

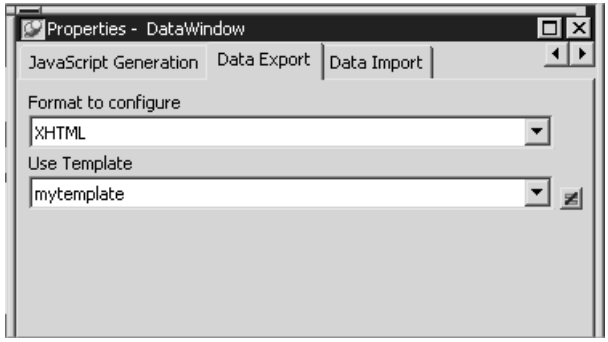
Selecting the template to use

Using the Export.XHTML. UseTemplate property

The Data Export page in the Properties view lets you set properties for exporting data in XHTML. The names of all templates that you create and save for the current DataWindow object display in the Use Template drop-down list.

In addition to the properties that you can set on this page, you can use the Export.XHTML.TemplateCount and Export.XHTML.Template[ ].Name properties to let the user of an application select an export template at runtime. See "Selecting XHTML export templates at runtime" on page 174.

You can specify the template you want to apply to the default XML Web DataWindow or XHTML Web DataWindow generation at runtime by setting the Export.XHTML.UseTemplate property. You set the property using the Data Export tab in the DataWindow painter's Properties view by selecting XHTML as the format and then selecting the XHTML export template's name from the Use Template drop-down list box.



You can also set the Export.XHTML.UseTemplate DataWindow property in script. For information, see "Selecting XHTML export templates at runtime" on page 174.

Incorrect setting of the UseTemplate property
If you set the Export.XHTML.UseTemplate property at runtime to the name of a template that does not exist, the built-in default Template is used on an export.

Properties related to XHTML export templates

Table 6-11 shows properties related to XHTML export templates.

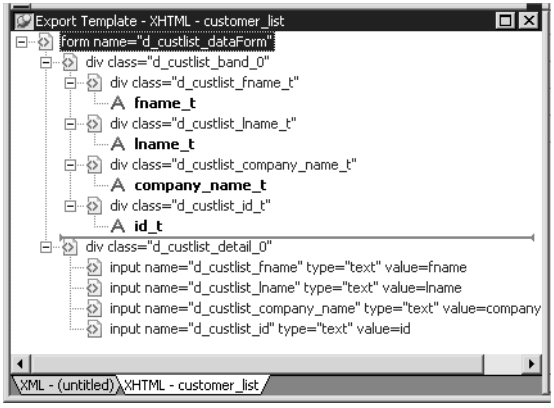
Table 6-11: Properties for XHTML export templates

Property	User interface fields	Description
Export.XHTML.TemplateCount	Read only, so no user interface field.	The number of XHTML export templates associated with a DataWindow object
Export.XHTML.Template[num].Name	Read only, so no user interface field.	The name of an XHTML export template associated with a DataWindow object returned by index value that ranges from 1 to the value of the Export.XHTML.TemplateCount property
Export.XHTML.UseTemplate	Select a template from the Use Template drop-down list box in the Data Export tab in the DataWindow painter's Properties view.	The name of an XHTML export template (previously saved in the DataWindow painter) that optionally controls the logical structure of the XHTML generated by a DataWindow object

For detailed information about DataWindow properties, see the *DataWindow Reference*.

Template structure

An XHTML export template has a Header section and a Detail section separated graphically by a line across the tree view. Other DataWindow bands are incorporated into these sections.



The Detail Start element

A line across the Export/Import Template view separates the Header section from the Detail section. The first element after this line, d_dept_list_row in the previous screen shot, is called the Detail Start element.

There can be only one Detail Start element, and it must be inside the document's root element. Each band of the DataWindow is wrapped by a element. When the DataWindow is exported in XHTML, this element and all children and/or siblings after it are generated iteratively for each row.

Header section

The Header section can contain the items listed in Table 6-12. Only the root XHTML <form> element is required:

Table 6-12: Items permitted in the Header section of an XHTML document

Item	Details
Root <form> element (start tag)	The XHTML <form> element is the root element of the XHTML template. See "Root element" on page 168.
XHTML elements	Additional elements below the root element.
DataWindow control references	Text. See "DataWindow controls" on page 169.
DataWindow expressions	Text. See "DataWindow expressions" on page 169.
Literal text	Text that does not correspond to a DW control.
Attributes	Can be assigned to all elements. See "Element attributes" on page 170.
CDATA sections	See "CDATA sections" on page 172.
Child elements	Child elements in the Header section cannot be iterative except for the Group DataWindow.

Detail section in root element
The root element displays in the Header section, but the entire content of the Detail section is contained in the root element.

The items in the Header section are generated only once at runtime (when the DataWindow is exported to XHTML), unless the DataWindow is a Group DataWindow. For Group DataWindows, the corresponding XHTML fragment in the Header section is repeated so that it iteratively heads each group detail—the group of XHTML rows corresponding to the group specified in the DataWindow.

The Header section contains the rendering of the DataWindow header band and any group header bands. Bands are generated within elements. The controls rendered in the Header section (such as computed titles and text control column headings) are typically also generated within elements, with referenced content.

These entities are generated only once and are not iterated for each row. However, for DataWindows with group headers, the corresponding XHTML fragment in the Header section is repeated, iteratively heading each group of XHTML rows corresponding to the group specified in the DataWindow.

Detail section

The Detail section contains the rendering of the DataWindow Detail band, delimited by the first element. The element's contents represent a single row instance to be generated iteratively. Any group trailers, summary band, and footer band are also appended and enclosed by elements. The controls rendered in the Detail section (for example, column, computed field, DropDownDataWindow, DropDownListBox, checkbox, and button controls) are usually also generated within , <input>, or <select> elements with referenced content.

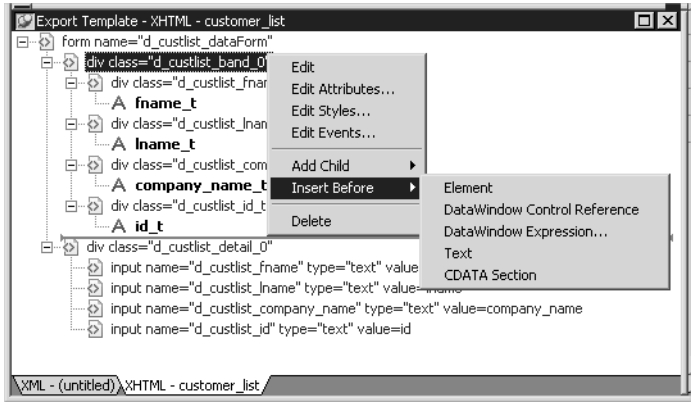
The Detail section can contain the items listed in Table 6-13.

Table 6-13: Items permitted in the Detail section of an XHTML document

Item	Details
First element	The contents of the element represent a single row instance to be generated iteratively.
XHTML elements	Additional elements below the root element.
DataWindow control references	Text. See "DataWindow controls" on page 169.
DataWindow expressions	Text. See "DataWindow expressions" on page 169.
Literal text	Text that does not correspond to a DW control.
Attributes	Can be assigned to all elements. See "Element attributes" on page 170.
CDATA sections	See "CDATA sections" on page 172.
Child elements	Child elements in the Header section cannot be iterative except in the case of group DataWindows.

Editing XHTML export templates

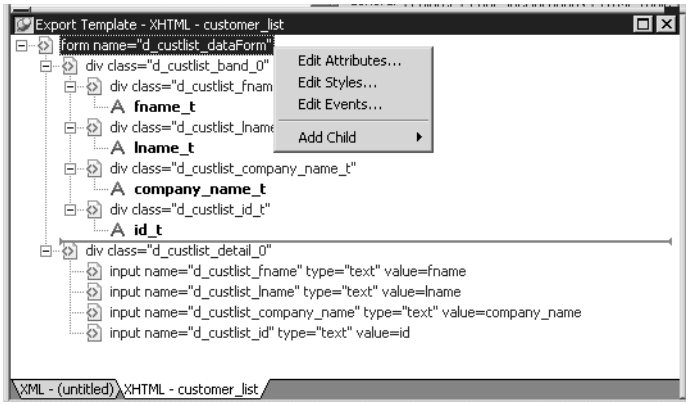
Every item in the Export Template view for XHTML has a pop-up menu for modifying the structural layout of the XHTML document that will be generated at runtime. Using the pop-up menu, you can perform actions appropriate to that item, such as editing or deleting the item, adding or editing attributes, adding child elements or other items, and inserting elements, CDATA sections, and so forth, before the current item.



If an element has no attributes, you can edit its tag in the Export Template view for XHTML by selecting it and left-clicking the tag or pressing F2. Literal text nodes can be edited in the same way. You can delete items (and their children) by pressing the Delete key.

Root element

The root element of the XHTML export template is the XHTML <form> element. You can change the name of the root element and add attributes and children.



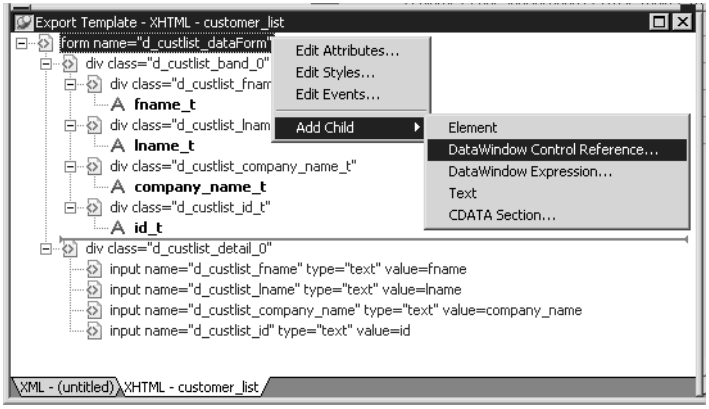
Changing the name of the root element changes the name of its start and end tags. You can change the name using the Edit Attributes menu item to display the Element Attributes dialog box. For information about editing attributes, see “Element attributes” on page 170.

You can add the following kinds of children to the root element:

- Elements
- Text
- DataWindow control references
- DataWindow expressions (including column references)
- CDATA sections

DataWindow controls

Adding a DataWindow control reference opens a dialog box containing a list of the columns, computed fields, report controls, and text controls in the document.



Control references can also be added to empty attribute values or element contents using drag and drop from the Control List view. Column references can also be added using drag-and-drop from the Column Specifications view.

Drag-and-drop cannot replace
You cannot drag-and-drop an item on top of another item to replace it. For example, if you want to replace one control reference with another control reference, or with a DataWindow expression, you first need to delete the control reference you want to replace.

DataWindow expressions

Adding a DataWindow expression using the Add Child>DataWindow Control Reference menu item opens the Modify Expression dialog box. This enables you to create references to columns from the data source of the DataWindow object. It also enables the calling of global functions. One use of this feature is to return a fragment of XHTML to embed, providing another level of dynamic XHTML generation.

Using Date and DateTime with strings

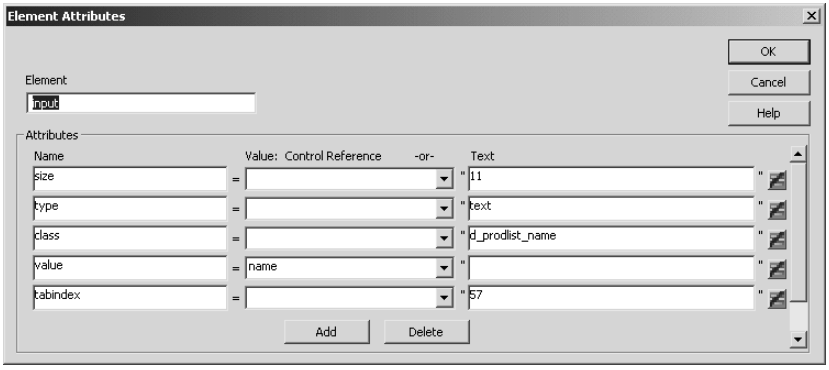
If you use a control reference or a DataWindow expression that does not include a string to represent Date and DateTime columns in a template, the XHTML output conforms to ISO 8601 date and time formats. For example, consider a date that displays as `12/27/2002` in the DataWindow, using the display format `mm/dd/yyyy`. If the export template does not use an expression that includes a string, the date is exported to XHTML as `2002-12-27`.

However, if the export template uses an expression that combines a column with a Date or DateTime datatype with a string, the entire expression is exported as a string and the regional settings in the Windows registry are used to format the date and time.

Using the previous example, if the short date format in the registry is `mm/dd/yy`, and the DataWindow expression is: `"Start Date is " + start_date`, the XHTML output is `Start Date is 12/27/02`.

Element attributes

Select Edit Attributes from the pop-up menu for elements to edit an existing attribute or add a new one. The attributes that display include all the default attributes for the elements with any template changes applied. The name attribute (and in some cases the class attribute) used to identify the element is omitted and cannot be changed.



You can change or delete the default attribute values or add new ones. Controls or expressions can also be referenced for element attribute values.

For each attribute specified, you can select a control reference from the drop-down list or enter a literal text value. A literal text value takes precedence over a control reference. You can also use the expression button to the right of the Text box to enter an expression.

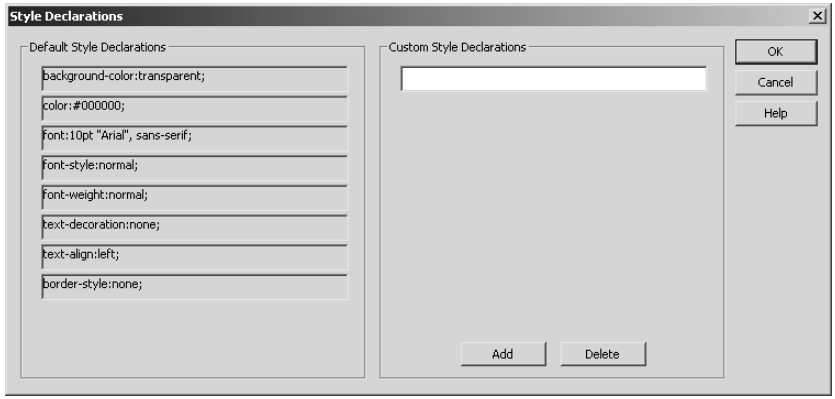
The expression button and entry operates similarly to DataWindow object properties in the Properties view. The button shows a green equals sign if an expression has been entered, and a red not-equals sign if not. A control reference or text value specified in addition to the expression is treated as a default value. In the template, this combination is stored with the control reference or text value, followed by a tab, preceding the expression. For example:

```
attribute_name=~"text_val~~tdw_expression~"
```

When you finish modifying element attributes and you click OK, only changes are stored in the template. Default attributes that are deleted are added in the template and marked with an empty value.

Style declarations

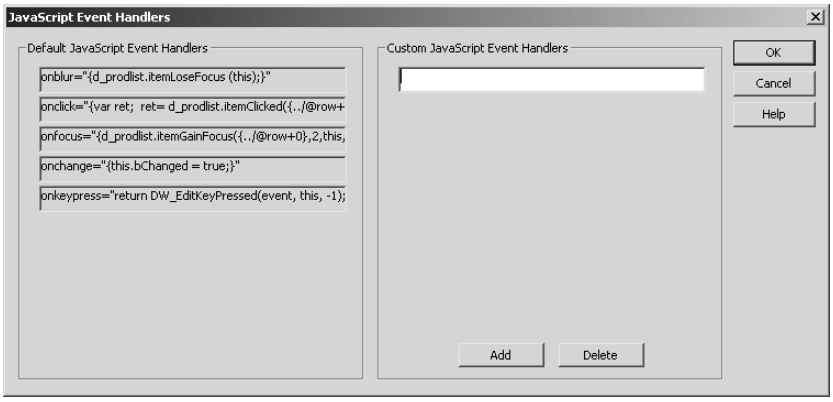
If you right-click an element and select Edit Styles from the pop-up menu, the Style Declarations dialog box displays the read-only set of default style declarations for the element on the left:



For clarity, style declarations are omitted from the XHTML export template. You can add new style declarations or override the existing ones by declaring them on the right side, or remove them by adding them with an empty value.

JavaScript event handlers

If you right-click an element and select Edit Events from the pop-up menu, the JavaScript Event Handlers dialog box displays a read-only set of event handlers for the element on the left:



This dialog displays the current JavaScript event handlers, if any. You can add new event handlers or override the existing ones by declaring them on the right side, or remove them by adding them with an empty value.

CDATA sections

Everything inside a CDATA section is ignored by the parser. If text contains characters such as less than or greater than signs (< or >) or ampersands (&) that are significant to the parser, it should be defined as a CDATA section. A CDATA section starts with `<![CDATA[` and ends with `]]>`. CDATA sections cannot be nested, and there can be no white space characters inside the `]]>` delimiter—for example, you cannot put a space between the two square brackets.

Example

```
<![CDATA[
   do not parse me
]]>
```

Element Context Menus

The tree view in the Export Template view for XHTML represents a real-time DOM tree. Each XHTML element of the tree in the Header and Detail sections has a pop-up menu. The pop-up menu items perform DOM-based actions for modifying the structural layout of the XHTML document that will be generated. The menu options include:

Menu item	DOM-based action
Edit	DOMNode::SetNodeName
Add Child	DOMNode::AppendChild
Insert Before	DOMNode::InsertBefore
Delete	DOMNode::RemoveChild

DOM-based actions

Edit allows changing the label of the tree view item representing the XHTML element name. All element items that display no attributes, as well as literal text nodes selected in the tree view, can also be edited with a single mouse-click or with the shortcut key F2. Add Child allows appending an entity as a last child. The submenu option DataWindow Control Reference invokes a dialog containing a filtered list box of Column, Computed Field, and Text controls for user selection. Control references can also be added to empty attribute values or element contents using drag-and-drop from the existing Control List View. DataWindow Expressions can also be added using the existing dialog. DataWindow column references (in the form of expressions) can also be added using drag-and-drop from the Column Specification View. Tree view items, except the <form> element, can also be deleted with the Delete key.

Presentation and function

The remaining context menu items invoke dialogs that allow overriding presentational and functional specifications of each element. These include:

- Style declarations
- Element attributes
- JavaScript event handlers

The dialogs first display these specifications as they would be generated at runtime by default. The painter gets these from the XML Web Generator in DWE in real-time, read-only display on one half of the dialog. Within input field(s) on the other half of the dialog, the developer can override these specifications at the atomic declaration or attribute level. This includes resetting included declarations/attributes, setting declarations/attributes not included, or removing declarations/attributes. These change specifications will then persist in the XHTML export template, and be applied to the default presentation generated by the XML Web Generator at runtime.

Selecting XHTML export templates at runtime

Two DataWindow properties, Export.XHTML.TemplateCount and Export.XHTML.Template[ ].Name, enable you to provide a list of templates from which the user of the application can select at runtime.

The TemplateCount property gets the number of templates associated with a DataWindow object. You can use this number as the upper limit in a FOR loop that populates a drop-down list with the template names. The FOR loop uses the Template[ ].Name property.

```
string ls_template_count, ls_template_name
long i

ls_template_count=dw_1.Describe
  ("DataWindow.Export.XHTML.TemplateCount")

for i=1 to Long(ls_template_count)
  ls_template_name=
  dw_1.Object.DataWindow.Export.XHTML.Template[i].Name
  ddlb_1.AddItem(ls_template_name)
next
```

Before generating the XHTML, set the export template using the text in the drop-down list box:

```
dw_1.Object.DataWindow.Export.XHTML.UseTemplate=
   ddlb_1.text
```

Exporting the DataWindow Web form in XML and XSLT or in XHTML

Exporting in XML and XSLT

You can export the DataWindow or DataStore object in XML and XSLT using PowerScript dot notation or the Describe method:

```
ls_xmlstring = dw_1.Object.DataWindow.Data.XMLWeb
ls_xmlstring = dw_1.Describe("DataWindow.Data.XMLWeb")
```

When you export the DataWindow or DataStore object, PowerBuilder uses an export template to specify the content of the generated XSLT and CSS style sheets.

Default export format
If you have not created or assigned an export template, PowerBuilder uses the default XSLT export format. This is the same format used when you create a new default export template. See “Creating and saving templates” on page 161.

Exporting in XHTML

You can export the DataWindow or DataStore object in XHTML using PowerScript dot notation or the Describe method:

```
ls_xmlstring = dw_1.Object.DataWindow.Data.XHTML
ls_xmlstring = dw_1.Describe("DataWindow.Data.XHTML")
```

When you export the DataWindow or DataStore object, PowerBuilder uses an export template to specify the content of the generated XHTML and CSS style sheet.

Default export format
If you have not created or assigned an export template, PowerBuilder uses the default XHTML export format. This is the same format used when you create a new default export template. See “Creating and saving templates” on page 161.

CHAPTER 7

Server-Side Processing for the Web DataWindow

About this chapter

This chapter describes configuration requirements for the server-side Web DataWindow component and design and programming techniques.

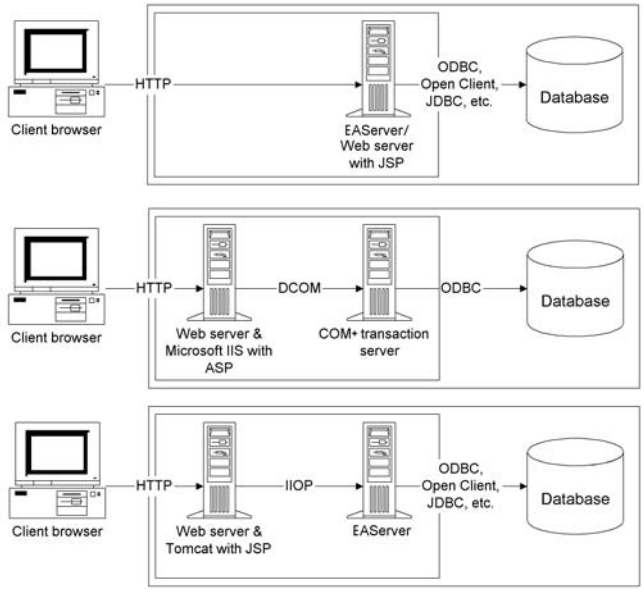
Contents

Topic	Page
Server configuration requirements	177
Instantiating and configuring the server component	182
Using a custom server component	194
Using service classes	203

Server configuration requirements

The servers and clients used by the Web DataWindow can run on the same or different machines. The following diagram shows typical configurations with the Web server and page server on the same machine and the component server and database on separate machines, but any or all of the servers can run on the same computer. In your development environment, the client browser could be on the same computer too.

Figure 7-1: Typical client and server configurations



Web server is built in to EAServer
As shown in the first configuration, EAServer has a built-in Web server, so this configuration demonstrates the full capability of EAServer.

Using the Web DataWindow with EAServer

If you are not running your Web server as a service and you have a user CLASSPATH environment variable, make sure that this variable includes the paths to all classes needed for communication between the Web server and EAServer. This includes the Sybase\EAServer\html\classes directory and the path to any component stubs that you generate for use with your Web pages (if these are in a different directory). When you install EAServer, the path information for the server is placed only in the system CLASSPATH variable, not the user variable.

Using the Web DataWindow with Tomcat

To use Web DataWindows and access EAServer components in JSP pages that you deploy to a Tomcat server, you must add the following JAR files and classes to the class path used by Tomcat at startup:

- jconn2.jar
- easclient.jar
- easj2ee.jar
- *EAServer_home*\html\classes
- *EAServer_home*\java\classes

where *EAServer_home* is the full path to the EAServer where the Web DataWindow component is installed. To add the required files to the Tomcat class path, you need to edit the *catalina.bat* file in the Tomcat bin directory.

Using the Web DataWindow with a COM component

You can also use the Web DataWindow with Microsoft servers, but more configuration tasks are required.

Only the HTML Web DataWindow is supported
No changes were made to the ASP Object model for PowerBuilder 11.0. The GenerateXMLWeb and GenerateXHTML methods are therefore not supported.

On COM+, you access the COM version of the generic server component (PowerBuilder.HTMLDataWindow) using automation methods called from the page server. For example, to instantiate the generic COM component from ASP:

```
dwGen = Server.CreateObject(
   "PowerBuilder.HTMLDataWindow")
```

The COM server component is provided in *PBDWR110.DLL*. You can also write your own custom class user object based on the sample source code in *PBDWRMT.PBL*.

For more information about building Web pages that use the Web DataWindow server component, see “Instantiating and configuring the server component” on page 182. For information on using the Web DataWindow COM server component from a JSP page, see the white paper on the Sybase Web site at http://www.sybase.com/detail?id=1029911.

You need to register the COM server component and install PowerBuilder runtime files and the PBL or PBD containing your DataWindow objects on the COM+ server computer.

Transaction server configuration tasks

Table 7-1 summarizes the configuration tasks you need to perform on the transaction server.

Table 7-1: Configuration tasks for transaction server

Server	Tasks
EAServer or COM+	Copy the PBLs, PBDs, SRDs, or PSRs containing the definitions of your DataWindow objects to a directory on the EAServer server's path or the system path if the server component is running as a service (always true for COM+).
EAServer	Set up a DSN and a connection cache for your data source.
ASP (IIS) with COM+	• Copy the following PowerBuilder runtime files to the *system* path: *PBVM110.DLL*, *LIBJCC.DLL*, *LIBJUTILS.DLL*, *PBDWE110.DLL*, *PBODB110.DLL*, *PBODB110.INI* • Copy the file containing the generic COM server component to the *system* path and register it: `regsvr32.exe pbdwr110.dll` • If you designed your own custom class user object and built a COM server component, you need to copy your component DLL to the server and register it • If necessary, install the ODBC driver for your DBMS and use the ODBC Administrator to define an ODBC data source for your DataWindow objects as a *system* DSN If COM+ is hosting the server component and running on a different computer from IIS, create a client install package using COM+ Explorer and install it on the IIS machine. For more information, see the COM+ documentation.
JSP (Tomcat) with COM+	To use a COM+ component with a JSP page running on Tomcat or another JSP server, you need to use a Java-COM bridge. For more information, see the white paper on "How to set up a JSP that uses a DW DTC to access the Web DW on MTS via Tomcat" on the Sybase Web site at http://www.sybase.com/detail?id=1029911.

Page server configuration tasks

Table 7-2 summarizes the configuration tasks you need to perform on the page server.

Table 7-2: Configuration tasks for page server

Server	Tasks
ASP	Copy the HTML pages, scripts, images, and other application files to the page server. You can use the Web target deployment controller for ASP to copy them to the server.
JSP	Copy the JSPs, custom tag libraries, scripts, images, and other files to the page server. You can use the Web target deployment controller for JSP to copy them to the server. Although a custom tag library you add to a page is placed in the Web-Inf\tld directory and deployed with the target, you must manually copy any classes or archives required by the tag library. Tag library class files should be copied to the Web-Inf\classes directory, and tag library archive files should be copied to the Web-Inf\lib directory. For information about the Web DataWindow custom tag library, see the "Custom Tag Reference" chapter in the *JSP Target Reference*.
ASP (with EAServer as the component server)	If ASP and EAServer are not installed on the same computer: • Copy *JAGCLIENT.EXE* from the EAServer client subdirectory to the ASP server, run it, and select ActiveX Runtime. • Copy *PBDWR110.TLB* and *PBDWR110.REG* to the ASP server from the Sybase\Shared\PowerBuilder directory on the computer where PowerBuilder is installed. • Edit the HKEY_CLASSES_ROOT\TypeLib lines in *PBDWR110.REG* that show the location of the *PBDWR110.TLB* file to show the correct location on the ASP server. • Double-click *PBDWR110.REG* to install its information into the Windows registry. If you designed your own custom class user object, generate TLB and REG files for the component in EAServer Manager and register them on the ASP server. For more information, see the chapter on creating ActiveX clients in the *EAServer Programmer's Guide*.

Instantiating and configuring the server component

If you are not using the Web DataWindow DTC or the Web Target object model, you can write code to create an instance of the Web DataWindow server component, and you can call its methods to create a Web DataWindow application.

For information on the types of Web DataWindow server components, see "The Web DataWindow server component and client control" on page 113. For information on the Web Target object model and its 4GL extensions, see *Working with JSP Targets*.

Two sets of methods

Two sets of methods are available on the generic Web DataWindow server component:

- Methods that are available for other DataWindow controls
- Methods used to configure the component and generate HTML

DataWindow control methods

DataWindow control methods supported by the generic server component include sorting, filtering, validation, and get and set methods. When you call one of these methods on the server component, the server reloads the page in the browser.

Methods with more than one syntax have a different form for each syntax to overcome restrictions on the use of overloading. For example, the ClearValues method takes a string as an argument and the ClearValuesByColNum method takes a number.

For a complete list of supported DataWindow control methods, see the *DataWindow Reference* or the online Help.

Examining server component methods

You can view the generic EAServer component methods on the Components page of the System Tree or in EAServer Manager. You can examine the methods in the COM server component in an OLE viewer (look for PowerBuilder.HTMLDataWindow in the Programmable Objects list on the OLE tab of the PowerBuilder Browser, or in the Automation Objects list in the Microsoft OLE/COM Object Viewer).

Configuration and generation methods

Other methods are available to set up the component, retrieve data, establish persistent values needed by your Web page, and generate HTML.

If you use a custom server component, there are additional configuration tasks. For more information, see "Using a custom server component" on page 194.

Mixed case method names
The methods of the generic EAServer server component use mixed case names and all the examples in this section use mixed case. If you write your own server component, the methods of the component you generate are all lowercase. (You can use the sample *PBDWRMT.PBL* as a starting point if you want the methods described here.) The generic COM component method names are also all lowercase.

Coding steps

In your server-side script, you will code these tasks:

1 Instantiate the component.

2 Load the DataWindow object.

3 Control what HTML is generated (for example, by specifying what functionality to include and what browser to target).

4 Specify the database connection and retrieve data.

5 Pass page-specific data to the reloaded page.

6 Pass user action information to the server component.

7 Insert the generated HTML in the page template.

Sample code for some of these tasks follows. For detailed information about the methods used in the examples, see the *DataWindow Reference* or the online Help.

Instantiating the component

JSP targets If you are not using the *jspobject110.jar* object model, you instantiate a Web DataWindow component in the following manner:

```
<%@ page contentType="text/html; charset=UTF-8" %>
<%@ page import="org.omg.CORBA.ORB" %>
<%@ page import="org.omg.CosNaming.NamingContext" %>
<%@ page import=
        "org.omg.CosNaming.NamingContextHelper" %>
<%@ page import="org.omg.CosNaming.NameComponent" %>
<%@ page import="DataWindow.*" %>

String dwGenerator = "DataWindow/HTMLGenerator110";
HTMLGenerator110 dwGen = null;
```

```
java.util.Properties props =new java.util.Properties();
props.put("org.omg.CORBA.ORBClass",
   "com.sybase.CORBA.ORB");
props.put("com.sybase.CORBA.NameServiceURL",
   "iiop://testmachine:9000");
ORB orb = ORB.init((String[])null, props);
try {
     NamingContext cntx = NamingContextHelper.narrow
      (orb.resolve_initial_references("NameService"));
     NameComponent[] name = {new
       NameComponent(dwGenerator,"")};
     SessionManager.Factory factory =
       SessionManager.FactoryHelper.narrow
         (cntx.resolve(name));
     dwGen =
        HTMLGenerator110Helper.narrow(factory.create
          ("jagadmin",""));
} catch(org.omg.CORBA.ORBPackage.InvalidName ie) {
      out.print("Error: " + ie.getMessage());
} catch
(org.omg.CosNaming.NamingContextPackage.NotFound ne) {
      out.print("Error: " + ne.getMessage());
} catch
(org.omg.CosNaming.NamingContextPackage.CannotProceed
    ce) {
      out.print("Error: " + ce.getMessage());
} catch
(org.omg.CosNaming.NamingContextPackage.InvalidName
    ie1) {
      out.print("Error: " + ie1.getMessage()); }
```

ASP targets You can access the registered COM component using its Program ID. For example, if you use the generic component:

```
dwGen = Server.CreateObject(
   "PowerBuilder.HTMLDataWindow")
```

Loading the DataWindow object

SetDWObject The next step is to specify the PBD or PBL file that contains the DataWindow object and the name of the DataWindow object. You do not need to specify the location of the file, but it must be available on the component server in a directory on the server's path (or on the system path if the EAServer component is running as a service or if you are using COM+):

```
retVal = dwGen.SetDWObject ("htgenex.pbl",
      "d_tabular_dept");
```

You can test the return value for and write an error message to the HTML page in a JSP target with the following code:

```
   if (retVal != 1){
      out.print ("SetDWObject failed " +
         retVal + "<BR>");
}
```

You could use similar code in an ASP target:

```
   if (retVal != 1){
      Response.Write("SetDWObject failed " +
         retVal + "<BR>");
}
```

You can also specify a:

- Source definition (SRD) file containing the source for a DataWindow object. You can export a DataWindow definition to an SRD file in the Library painter or System Tree.
- Powersoft report (PSR) file containing a DataWindow object plus data. You can save a PSR file from the DataWindow painter.
- DataWindow Container component on EAServer containing multiple DataWindow object definitions.

For SRD and PSR files, specify an empty string for the DataWindow name:

```
dwServer.SetDWObject("myreport.psr", "" );
```

For DataWindow Container components, use the the SetDWObjectEx method:

```
dwServer.SetDWObjectEx ("d_emp");
```

Controlling what is generated

Disabling features of the client control

SetWeight Although the server component generates a considerable amount of HTML or XHTML and JavaScript for the Web DataWindow client control, it is still no more than an average image file. However, to reduce the size of the control on the client, you can instruct the component to leave out code for features you are not using. You can tell the component to omit code for:

- Updating data
- Validating newly entered data
- Client-side events
- Allowing client-side scripts to call methods of the client control
- Applying display formats to newly entered data

You can disable any of these on the Web Generation property page in the DataWindow painter or with the SetWeight method. False for a particular argument means no code for that feature is generated.

This statement enables all features:

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

If updating of data is false, no validation or display formatting code is generated either. In this statement, it does not matter what the second and fifth arguments are, because the first argument for updating data is false:

```
dwGen.SetWeight(false, false, true, true, false);
```

This statement turns off the client-side scripting capability:

```
dwGen.SetWeight(true, true, true, false, true);
```

Updating data and display formatting add the most code to the client-side control. Date processing also generates additional code. For the smallest client control, turn on only the features you need and make sure your DataWindow object does not have any date columns.

Naming the client control

SetHTMLObjectName You need to provide a name for the Web DataWindow client control. The name is used for page parameters and client-side events. If there is more than one Web DataWindow client control on the Web page, each needs a unique name.

This code uses the same name for the server component variable and the client control:

```
dwGen.SetHTMLObjectName ("dw_1");
```

XML Web DataWindow
If you are using an XML Web DataWindow with a custom XHTML template, the object name in the template must match the name of the client control (dw_1 in the example).

Optimizing HTML for a browser

SetBrowser The Web DataWindow can generate HTML optimized for particular browsers and versions. In particular, it can generate code for Microsoft and Netscape browsers. The browser might be different each time the server component is instantiated by a different client, so this information cannot be preset in the DataWindow painter. You can tell it what browser and version to target in the server-side script. In the painter, you can set the HTML Version property to specify what level of HTML to generate if the browser is not recognized.

For information on what HTML features the DataWindow uses for different browsers, see the *DataWindow Reference* or the HTMLGen.property topic in online Help.

At runtime, the HTTP header sent from the client browser to the Web server contains the User-Agent or HTTP_USER_AGENT value, which the server component can use to identify the client browser.

In JSP, you call the getHeader method of the request object to get the User-Agent value:

```
String browser=(String)request.getHeader("User-Agent");
 dwGen.SetBrowser(browser);
```

In ASP, you can use the ServerVariables method of the Request object to get the HTTP_USER_AGENT value:

```
var clientbrowser =
   Request.ServerVariables("HTTP_USER_AGENT");
dwGen.SetBrowser(clientbrowser);
```

Specifying the database connection and retrieving data

Specifying connection information

SetTrans You provide connection information for the server component with the SetTrans method. The arguments you specify depend on the type of connection. For an ODBC connection to Adaptive Server Anywhere, you specify all the connection information in the dbParm argument.

In EAServer, you must also set up a connection cache for the component, described in “Creating a connection cache on EAServer” on page 147.

The data source must be defined on the server machine. It must be a system DSN on EAServer if the component is running as a service, and it must always be a system DSN on COM+. This statement connects to the EAS Demo DB sample database:

```
dwGen.SetTrans("ODBC","ConnectString='DSN=EAS Demo
DB V110;UID=dba;PWD=sql'","", "", "", "", "");
```

If you are using the XML Web DataWindow, it is best to call the SetPageSize method to limit the number of rows per page:

```
dwGen.SetPageSize(2);
```

Using Adaptive Server Enterprise
PowerBuilder and EAServer use slightly different versions of the CT-Lib software to connect to Adaptive Server Enterprise via Open Client. In the PowerBuilder development environment, you use the SYC native database interface to connect to the database, but you must use SYJ as the first argument to SetTrans to connect to ASE in EAServer.

Retrieving data

Retrieve To tell the server component to retrieve data when the DataWindow object does not have retrieval arguments, you call the Retrieve method:

```
retVal = dwGen.Retrieve();
```

This ASP code calls Retrieve and, if the return value reports an error, calls GetLastErrorString for the error message text:

```
    retVal = dwGen.Retrieve();
if (retVal < 0 )
    {
       Response.Write("Error on Retrieve: " + retVal +
              "<BR>");
       Response.Write(dwGen.GetLastErrorString() +
              "<BR>")
    }
```

Specifying retrieval arguments

RetrieveEx If the DataWindow object expects retrieval arguments, call RetrieveEx:

```
dwGen.RetrieveEx("60000");
```

Typically, the retrieval arguments are not constants. They are page parameters passed to the page from another page where the user filled in a form or clicked a hyperlink. If the DataWindow expects more than one retrieval argument, the arguments must be passed in a single string. The arguments in the string must be separated by newline characters (\n), and individual values cannot contain newline characters as part of the value. Array values must be separated by tab characters (\t).

This example form prompts the user for a salary:

```
<FORM ACTION="salaryrpt.jsp" NAME="log" METHOD="POST">
<p>Please enter a salary ceiling:</p>
<P><INPUT TYPE="text" NAME="salary" SIZE="15"></p>
<p><Center>
<INPUT TYPE="Submit" VALUE="Register">
</Center></p>
</FORM>
```

Code in server-side script in salaryrpt.jsp can then call the RetrieveEx method for the server component using the salary value passed as a page parameter:

```
String Salary =(String)request.getParameter("salary");
dwGen.RetrieveEx(Salary);
```

Getting the retrieval argument from another page works the first time the page is loaded. The retrieval arguments have to be page parameters each time the page is reloaded. To specify page parameters for the reloaded page, you use the SetSelfLink method, described next.

Passing page-specific data to the reloaded page

Using self link information

The first time the client browser requests the page template, it can pass page-specific information using GET or POST, and the page can use those values in the server-side scripts. However, when the page is reloaded because of user interactions with the Web DataWindow, that information is not passed to the page automatically.

To make the information available, you specify a *selflinkargs* string with values that become page parameters in the reloaded page. Typically, you would use self-link parameters to keep available:

- Login information from another page
- The DataWindow object name
- Retrieval arguments for the DataWindow object

To provide these values when the page is reloaded, you use the SetSelfLink method, which takes as arguments the URL of the page template as well as the selflinkargs string.

Getting the URL for the page

To reload the page correctly in response to user actions, the server component needs to know the URL of the page template. You can get this information from the name property of the document object header or the SCRIPT_NAME server variable.

In JSP, you must parse the return value from a request.getRequestURI call:

```
String URI = request.getRequestURI();
String [] myArray = URI.split ("/");
String pageName = myArray [myArray.length-1];
```

In ASP, use the ServerVariables method of the Request object:

```
var pageName =Request.ServerVariables( "SCRIPT_NAME" );
```

Building a self-link argument string

Self-link arguments become page parameters in the reloaded page. Your script typically looks at an existing page parameter and re-creates it using a self-link argument. The syntax for specifying a self-link argument string is:

pageparam1='*expr1*'|*pageparam2*='*expr2*'...|*pageparamn*='*exprn*'

The string can contain one or more page parameter and expression pairs separated by pipes (|). Each expression is a DataWindow expression that evaluates to a string. Usually you specify constant string values that are already values of page parameters rather than expressions.

The expression is enclosed in quotes, and if the value is a constant, it must also be enclosed in quotes. For example, if a page parameter has the value *Johnson*, the value of the expression must be enclosed in two sets of quote marks: `'"Johnson"'`

To get the value from the current Logname parameter, which is already defined for the page, you build the expression using the Logname page parameter. The single quotes and inner double quotes are embedded in the expression. The current value is inserted between the quotes:

```
String logname = (String)
   request.getParameter("Logname");
String linkargs =
       "logname='\"" + logname + "\"'";
```

An expression does not need the inner quotes:

```
String linkargs = "date='String(Today())'";
```

Passing the URL and argument string to SetSelfLink

SetSelfLink Use the URL and the link arguments string as arguments to the SetSelfLink method:

```
dwGen.SetSelfLink(pageName, linkargs);
```

Retrieval arguments as self-link values

The first time the page is loaded, the retrieval argument might be:

- A page parameter passed from another page. The user might have clicked a URL that included the value or filled in a form that posted the value.
- A new value calculated in the current script.

If the value is a page parameter, then you can re-create the page parameter using SetSelfLink. If the value is from some other source, you need to write code that gets the value from the source (which might be a page parameter) the first time the page is loaded and from a page parameter when it is reloaded.

Examples

These JSP examples show code that works with the types of values listed above. They illustrate how to get each type of value and use it in both RetrieveEx and SetSelfLink method calls.

Value from another page If the user entered a product ID in a form to get detailed information on the product, the product ID is passed to the product report template as a page parameter. The page parameter should always exist because it comes from the calling page, but the code provides a default value anyway:

```
String prod_id;
prod_id=(String) request.getParameter("ProdID");
if (prod_id == null){
   prod_id = "1";
}
dwGen.RetrieveEx(prod_id);
dwGen.SetSelfLink("ProdID=" + "'\"" + prod_id + "\"'");
```

Multiple values In this example, a Web page with a form prompts the user for a user name and a product category and the level of detail the user wants to see. The code uses the product category as a retrieval argument for the Web DataWindow. The script selects a DataWindow object based on the level of detail. All three values are carried over each time the page is reloaded:

```
// Get product category as a retrieval arg
String retrievearg, username, rptlevel, dw;
retrievearg =
    (String)request.getParameter("category");
if (retrievearg == null) {
retrievearg = "all";
}
int rtn = dwGen.RetrieveEx(retrievearg);
```

```
if (rtn < 0) {
    ... // Check for error
}
// Get the user name
username =(String)request.getParameter("username");
if (username != null){
out.print("<P>Dear " + username + "</P>");
}
out.print("<P>Here is the report you
    requested.</P>");

// Choose DW based on detail level requested
rptlevel=(String)request.getParameter("reportlevel");
if (rptlevel == "detail"){
    dw = "d_product_detail";
} else if (rptlevel == "summary"){
    dw = "d_product_summary";
} else {
    dw = (String) request.getParameter("dw");
    if (dw == null} {
    out.print ("<P>Error selecting report");
    //handle error or halt processing ...
}
dwGen.SetDWObject("productrpt.pbd", dw);

// Tell the server component to recreate the
// page parameters generated for the browser
String linkargs = "username='\"" + username + "\"'"
 + "|category= '\"" + retrievearg + "\"'"
 + "|dw= '\"" + dw + "\"'";

dwGen.SetSelfLink( pageName, linkargs );
```

Passing user actions to the server component

SetAction When the user clicks a DataWindow button, action information is passed back to the page server as context and action page parameters. Your server-side script needs to access those page parameters and call SetAction so the server component can apply the action to the generated HTML.

The names of the parameters use the object name specified in the SetHTMLObjectName method, for example: dw_1_action and dw_1_context. You can also specify the object name on the Web Generation tab page in the DataWindow painter.

You can include buttons for scrolling to other pages of data and for retrieving and updating data and inserting and deleting rows. When these button actions occur, the change is sent back to the server component and the change is made in the DataWindow buffer. If the user clicks an update button, the update method is called in the component without any other scripting needed.

No need to call methods
You can call server component methods directly for retrieving data, updating, inserting and deleting rows, and so forth. However, remember that button clicks invoke the actions. You do not need to call the methods too.

This JSP code checks whether parameters have been defined (meaning that the page is a reloaded page) and if so, calls SetAction to send the action information to the server component:

```
int retVal;
String dw_1_action =(String)request.GetParameter
    ("dw_1_action");
String dw_1_context = (String)request.GetParameter
    ("dw_1_context");
if (dw_1_context == null){
   dw_1_context = " ";
}
// Check if we need to perform the action
if (dw_1_action!=null){
  retVal = dwGen.SetAction(dw_1_action, dw_1_context);
  if (retVal < 0 ) {
    out.print("Error on SetAction: "+ retVal + "<BR>");
    out.print(dwGen.GetLastErrorString()+ "<BR>");
  }
}
```

Inserting the generated HTML or XHTML into the page

Generate After the server script has done all the setup, it calls the Generate function, which returns the generated HTML as a string.

In JSP, use out.print to insert code in the page template:

```
out.print( dwGen.Generate() );
```

In ASP, use Response.Write to insert the code in the page template:

```
Response.Write( dwGen.Generate() );
```

GenerateXHTML and GenerateXMLWeb In JSP targets, you return the Web DataWindow in XHTML with the GenerateXHTML command:

```
out.print( dwGen.GenerateXHTML() );
```

The GenerateXMLWeb method generates the content, layout, style, and client-side functionality of the DataWindow separately in XML, XSLT, CSS, and JS files. It returns the browser-specific XSLT transformation script that uses the generated files to render the DataWindow in XHTML on the client side:

```
out.print( dwGen.GenerateXMLWeb() );
```

Using a custom server component

If you are using EAServer as the component server, you can deploy a custom component that uses methods of the generic server component interface. You can also write a server component with its own DataWindow methods for use with EAServer or COM+.

Some advantages of a custom component

You can use a custom component to enhance:

- **Maintainability** Keep connection information on the server by specifying values for transaction properties.
- **Performance** Specify the source file and DataWindow object on the server so that the DataWindow object is loaded when the component instance is created, resulting in fewer method calls from server-side scripts in the Web page. You can also improve performance by having your custom component maintain its state.

  For information about changing the state property of a custom component, see “Maintaining state on the server” on page 201.
- **Scalability** Specify the source file and DataWindow object and use EAServer instance pooling so that the component is reused and loading the DataWindow object occurs only once.

Contention between the DataWindow painter and EAServer
If you are working in the DataWindow painter and testing the same PBL in EAServer, your PBL might be locked when the EAServer component loads the DataWindow. To avoid this, disable instance pooling for the component in EAServer Manager. After you have finished testing and editing the DataWindow object, you can enable instance pooling.

Instance pooling provides better performance in a production environment when a component instance can be reinitialized and reused for multiple clients.

For information on creating a custom component that uses the generic Web DataWindow interface, see "Creating a custom server component in EAServer" on page 196 or "Deploying DataWindow objects to the component server" on page 149.

Writing your own custom component

For full control of Web generation and the state of the DataStore object that holds the DataWindow definition and data, you can write your own custom class user object in PowerBuilder and deploy it as an EAServer or COM component. Using a custom component that includes only the processing you need can reduce the size of the client control returned to the Web client.

The source code for the generic component is available in *PBDWRMT.PBL* in the PowerBuilder code examples directory so that you can examine or reuse it. You can modify or add to the code in this PBL or start from scratch, using the sample PBL as a model for your own component.

Mixed-case method names
The methods of the generic EAServer component use mixed-case names and all the examples in this section use mixed case. If you write your own server component, the methods of the EAServer component you generate are all lowercase. However, you can change the case of the method names in the IDL file for your component after it is deployed to EAServer. When you use your own component, you must generate and compile stubs for the component, and you must do this after you make any changes to the IDL.

Your server component will use methods on a DataStore object to retrieve data and return the data and state to the client as HTML. To get the HTML and JavaScript that represents the state, data, and presentation of the DataWindow object, use the Describe method:

```
ls_html = ds_1.Describe("DataWindow.Data.HTML");
```

To update the HTML according to user actions, use the SetHTMLAction method:

```
li_rtn = ds_1.SetHTMLAction(arg_action, arg_context);
```

SetHTMLAction restores the state of the DataStore based on the context passed in as an argument and then changes the state based on the passed action.

For more information about SetHTMLAction, see the *DataWindow Reference* or online Help. For information about working with DataStore objects, see Chapter 4, "Using DataStore Objects".

If you create your own server component and deploy it to EAServer, you must also generate and compile the stubs. For information on generating and compiling stubs, see the EAServer documentation.

If you deploy the user object as a COM component, you need to register it. For information about registering COM components, see "Server configuration requirements" on page 177.

Creating a custom server component in EAServer

You can install and configure a custom Web DataWindow server component in EAServer Manager. In this procedure, you create a custom version of the EAServer server component with custom properties preset in EAServer Manager.

Using the Web DataWindow Container Project wizard

The procedure in this section describes how to create a custom component in EAServer Manager that uses the generic DataWindow::HTMLGenerator110 interface. You can also create a custom component with a Web DataWindow Container project. For more information, see "The Web DataWindow Container project wizard" on page 149.

❖ **To create a Web DataWindow custom component in EAServer:**

1 In the left pane of EAServer Manager, under the server name, right-click Packages and select New Package from the pop-up menu.

2 In the New Package dialog box, type a package name (you cannot change it later) and click Create A New Package.

For example, use EmpListPkg to identify a package for an application called EmpList.

3 On the General tab of the Package Properties dialog box, enter a description of the package and click OK.

   For example, enter *DataWindow Components for EmpList app* to describe EmpListPkg in terms of the application that you are customizing it for.

4 In the left pane of EAServer Manager, under the server name, right-click the new package and choose Install Component from the pop-up menu.

5 In the wizard, choose the Define New Component radio button and click Next.

6 Specify the name of your custom component (you cannot change it later) and click Finish.

   For example, use EmpListDW to identify a component that uses the d_emplist DataWindow object.

7 In the Component Properties dialog on the General tab, specify:

Module and interface	DataWindow::HTMLGenerator110
Component Type	PowerBuilder NVO
PowerBuilder Class Name	nv_remote_datawindow
PowerBuilder Library List	pbdwr110.pbd
PowerBuilder Application	remote_datawindow_appl

8 On the All Properties tab, add the properties for which you want preset values.

Setting properties for a custom component in EAServer

You add as many of the following properties as needed for your custom component. The properties are divided into two groups: general and database connection.

For boolean properties, values can be true or false, or yes or no.

General properties These properties specify settings that take effect when the component is instantiated.

Table 7-3: General properties to add for custom component

General property	Description
com.sybase.datawindow.sourceFileName	Specifies the PBL, the PBD that contains the DataWindow object for the component, or the SRD or PSR file that is the DataWindow object. See also the SetDWObject method in the online Help or *DataWindow Reference*.
com.sybase.datawindow.dwObjectName	The name of the DataWindow object in the PBL or PBD specified for sourceFileName. See also the SetDWObject method.
com.sybase.datawindow.fixed	Whether component properties can be modified from server-side script (SetDWObject, Create, Modify, and SetTrans methods) that instantiates the component. Values are: • Yes — Properties are fixed and cannot be changed. • No — Properties can be changed.
com.sybase.datawindow.serverServiceClasses	A list of PowerBuilder user objects that are in the PBL or PBD specified in sourceFileName. The class names should be separated by semicolons (;). The user objects implement custom events for data validation. For information on custom events, see the SetServerServiceClasses method in the online Help or *DataWindow Reference*.
com.sybase.datawindow.serverSideState	Specifies whether the server attempts to maintain its state between method calls. Values are: • Yes — The server component keeps the result set and keeps the transaction open if possible. • No — (Default) The result set is not saved and the server component uses information passed back from the client to retrieve the result set again and restore any uncommitted changes.
com.sybase.datawindow.trace	Whether calls to component methods are included in the EAServer server log. Values are: • Yes — Calls to component methods are listed in the log. • No — (Default) Calls to component methods are not logged.

General property	Description
com.sybase.datawindow.HTMLObjectName	The name used for the Web DataWindow client control in the generated code. The name is used to implement client-side events and to allow client-side scripting Set this property when there will be more than one Web DataWindow on a Web page so that they do not conflict. See also the SetHTMLObjectName method.
com.sybase.datawindow.modifyString	A string that is used as an argument to the Modify method for setting properties of the DataWindow object. The component calls the Modify method when it is initialized. For information on syntax, see the Modify method.

Database connection properties For the database connection properties, you must add com.sybase.datawindow.trans.dbms. This property must be set for any of the other trans properties to be recognized. When trans.dbms is set, any unspecified connection properties default to an empty string.

For more information about database connections, see the SetTrans method in the *DataWindow Reference*.

Table 7-4: Database connection properties to add for custom component

Database connection property	Description
com.sybase.datawindow.trans.dbms	A database vendor identifier, as displayed in the PowerBuilder Connection Profiles dialog box.
com.sybase.datawindow.trans.dbparm	DBMS-specific connection parameters.
com.sybase.datawindow.trans.lock	The isolation level. See the online Help for information about database preferences.
com.sybase.datawindow.trans.logid	The name or ID of the account the component uses when it logs on to the database server.
com.sybase.datawindow.trans.logpass	The password used to log on to the database server.
com.sybase.datawindow.trans.database	The name of the database to which the component is connecting. Ignored for ODBC.
com.sybase.datawindow.trans.servername	The name of the server on which the database resides.

Instantiating the custom component

To use the custom component in your server-side scripts, you specify your package and component name in the form YourPackage/YourComponent. If you are using methods of the generic component installed with EAServer, you must use the narrow method on the generic component helper class to reference the DataWindow/HTMLGenerator110 interface.

The following code in a JSP instantiates a custom component called EmpListDW that uses the generic HTMLGenerator110 component interface. EmpListDW is deployed in the EmpListPkg package. You substitute this code for the line that instantiates the generic component in the example for "Instantiating the component" on page 183, but you do not change the narrow method called on the generic component's helper class:

```
String dwGenerator = "EmpListPkg/EmpListDW";
   EmpListDW dwGen = null;
...
dwGen = HTMLGenerator110Helper.narrow(factory.create
          ("jagadmin",""));
```

Using OneTrip to set up the component and get the generated HTML

"Instantiating and configuring the server component" on page 182 described several items your server script should include to set up the Web DataWindow correctly. Instead of coding all these things separately, you can do all the setup and get the generated HTML with a single method when the EAServer component has been configured with a DataWindow definition and transaction information.

This technique is especially useful for improving performance without requiring the server component to maintain state.

```
String browser=(String)request.getHeader("User-Agent");
 dwGen.SetBrowser(browser);
 String URI = request.getRequestURI();
 String [] myArray = URI.split ("/");
 String selfLink = myArray [myArray.length-1];
 int retVal;
 String dw_1_action =(String)request.GetParameter
     ("dw_1_action");
 String dw_1_context = (String)request.GetParameter
     ("dw_1_context");
 if (dw_1_action == null){
  dw_1_action = "";
  }
 if (dw_1_context == null){
  dw_1_context = "";
```

```
 }
// Pass setup info to server
String dwHTML = dwGen.OneTrip("dw_1", browser,
       selfLink, "", dw_1_action, dw_1_context);
// Insert HTML returned from OneTrip in the page
out.print (dwHTML);
```

Using OneTripEx for retrieval arguments If your DataWindow requires retrieval arguments, use OneTripEx instead of OneTrip. The code checks for a page parameter that has the retrieval argument value. It also makes sure the value will still be available in a reloaded page by providing a *selflinkargs* expression:

```
String retrievearg = (String) request.getParameter
("RetArg"):
if (retrievearg == null){
// Provide some meaningful default value
   retrievearg = "default";
}
String selflinkarg = "RetArg='\"" + retrievearg + "\"'";
String dwHTML = dwGen.OneTripEx("dw_1", retrievearg,
   browser, selfLink, selflinkarg, action, context);
out.print (dwHTML);
```

Maintaining state on the server

Using a stateless component

The Web DataWindow can run in a fully stateless server environment. Variables in the Web page keep information about the rows being viewed and any changes the user makes; this information is communicated to the server component as needed so that the component can restore its state each time it is called. Restoring its state includes retrieving data from the database each time the page is reloaded, including each time the user navigates to another page.

Operating in a stateless mode minimizes use of server resources but can decrease performance. The client maintains the state of the server component in string form and the information is sent back and forth with every request. Also, when state is not maintained on the server, the component must connect to the database and retrieve data each time it is called. If the component server does not do connection caching, response time for the client could be very slow.

Using a stateful component

You can increase performance by maintaining state on the server. To maintain state, the page server's session object keeps a reference to the server component. If the server component is running in EAServer, you must mark the component as a stateful object and set a timeout value for the component. Failing to set a timeout value if you are using the component as a stateful object will result in orphaned instances of the component on the server.

Maintaining state on the server provides faster response time if the same component is accessed again. However, it also increases the server resources used for each client connection.

To minimize impact on server resources, a short timeout on a session lets the server get rid of a component that might not be requested again. If the component is called again, its state can be restored from the client state information. When the number of hits on a page is expected to be large, setting a shorter timeout reduces the number of instances that need to be maintained simultaneously.

Marking the object as stateful

To mark the component as a stateful object, set the component's com.sybase.datawindow.serverSideState property in EAServer Manager or call the SetServerSideState method in a server-side script:

```
dwGen.SetServerSideState( true );
```

You should *not* set the com.sybase.jaguar.component.stateless property for the HTMLGenerator110 component in EAServer Manager.

If you are using the Web Target object model, you specify whether the database state is managed on the server by passing the boolean value TRUE as the second parameter to the PSDataWindowClass constructor:

```
dw_1 = new PSDataWindowClass("dw_list", true,
       jagConn, dwSource, dbConn, 5);
```

Passing this parameter calls the SetServerSideState method, but it does not set a timeout value—you need to set a timeout value whether or not you are using the Web Target object model.

Setting timeout values

To set the timeout value for the HTMLGenerator110 component, open its Component Properties dialog box in EAServer Manager and set the com.sybase.jaguar.component.timeout property. Timeout values are specified in seconds; a value of 0 means the component never times out.

Using service classes

You can use the methods available on the server component to perform most server-side processing, including validation routines and error handling. The Web DataWindow also provides another way to add specialized processing.

To include server-side processing not available on the server component, you can define one or more PowerBuilder custom class user objects called service classes. These service classes are stored in the same PBL or PBD as the DataWindow object for the server component. They can be used whenever you want to include additional processing on the server. For example, you might want to use this technique to access the SQLPreview event so that you can examine the syntax of a SQL statement before it is committed to the database.

Where you implement the code

The service classes implement user-defined events with prescribed signatures. These events correspond to standard DataWindow events. In the user-defined events, you perform the processing and specify return codes that tell the server component whether to cancel the corresponding DataWindow event.

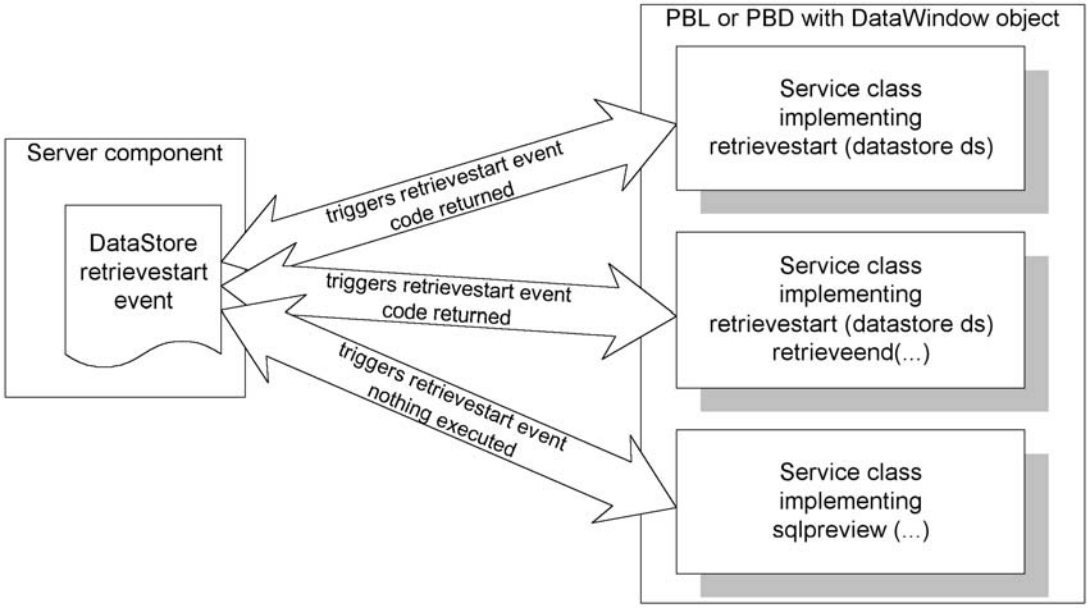
In the server component, you set a property or call a method that identifies these user objects as service classes for the server component.

How the code is called

Service classes work like this:

1. Service classes are instantiated when the component is instantiated (if they are specified in an EAServer property) or when they are first registered by the SetServerServiceClasses method.
2. An event occurs in the server component for the DataStore.
3. The server component calls an event of the same name in each registered service class.
4. If the service class implements the event, the event script is executed and a return code is sent back to the server component.
5. If the event can be canceled via a return code and if any of the service classes returns that code, the event is canceled in the server component.

Figure 7-2: How service classes work



Defining a service class for PowerBuilder components

❖ **To create and register a service class for PowerBuilder components:**

1 In the PBL that contains the DataWindow object for the server component, define one or more PowerBuilder custom class user objects.

2 In each custom class user object, define one or more user-defined events. The event signatures must match one of these (all these events return a long):

- DBError (long sqldbcode, string sqlerrtext, string sqlsyntax, DWBuffer buffer, long row, DataStore ds)
- HTMLContextApplied (string action, DataStore ds)
- RetrieveEnd (long rowcount, DataStore ds)
- RetrieveStart (DataStore ds)
- SQLPreview (SQLPreviewFunction request, SQLPreviewType sqltype, string sqlsyntax, DWBuffer buffer, long row, DataStore ds)

- UpdateEnd (long rowsinserted, long rowsupdated, long rowsdeleted, DataStore ds)
- UpdateStart (DataStore ds)

The arguments are the same as those documented for the similarly named DataWindow events in the *DataWindow Reference*, with the exception of the additional DataStore argument, which gives the user object access to the Web DataWindow data.

3 In the event script, use return codes to specify whether the server component should cancel the event.

The return codes are also the same as those documented in the *DataWindow Reference*. Any of the service classes that implements the event can specify that the event be canceled.

4 Register the service classes for the component.

There are two ways to make the user object available as a service class:

- For any component in EAServer or another supported server such as COM+, call the SetServerServiceClasses method in the Web page template's server-side script:

```
dwGen.SetServerServiceClasses
    ("uo_update_validate;uo_retrieve_process");
```

- For a custom component in EAServer, add this property in EAServer Manager:

```
com.sybase.datawindow.serverServiceClasses
```

Set its value to the list of user object names, with names separated by semicolons. For example:

```
uo_update_validate;uo_retrieve_process
```

Using server component methods with the DTC

If you are using the Web DataWindow DTC or the JSP Target object model, you can use methods of the server component by using the Component property. For example:

```
webDW.connectToComponent();
webDW.Component.SetServerServiceClasses
       ("uo_updatestart");
```

Example

Suppose that you want to check that data did not exceed a budgeted total before it was updated in the database. You might set up a service class that implements the UpdateStart event.

In the custom class user object in PowerBuilder, select Insert>Event and declare a new event called UpdateStart that returns a long and has one argument of type DataStore called ds:

```
UpdateStart (DataStore ds) returns long
```

This script for the UpdateStart event has a DataStore that retrieves data from a budget table and compares it to the component's data:

```
DataStore ds_budget
double darray[], total
long ll_upper
integer i

ds_budget = CREATE datastore
ds_budget.DataObject = "d_budget"
ds_budget.SetTransObject(...)
ds_budget.Retrieve( )

// Get data to be validated

darray[] = ds.Object.expenses.Primary
// Add up values in darray
ll_upper = UpperBound(darray)

FOR i = 1 to ll_upper
       total = total + darray[i]
NEXT
IF ds_budget.Object.cf_expense_total < total THEN
       RETURN 1
END IF
```

Defining a service class for Java components

❖ **To create and register a service class for Java components:**

1. Make sure your Java service class is in the system classpath.
2. In your Java service class, define one or more methods. Method prototypes must match one of these (all event datatypes are in the powersoft.datawindow.event package):
   - DBError (DatabaseEvent event, DataStore ds)
   - RetrieveEnd (RetrieveEvent event, DataStore ds)
   - RetrieveStart (RetrieveEvent event, DataStore ds)
   - SQLPreview (DatabaseEvent event, DataStore ds)
   - UpdateEnd (UpdateEvent event, DataStore ds)
   - UpdateStart (UpdateEvent event, DataStore ds)

   The arguments are the same as those documented for the similarly named DataWindow, Java Edition events in the *DataWindow Reference*, with the exception of the additional DataStore argument, which gives the Java class access to the Web DataWindow data.
3. In the class methods, set the return codes to specify whether the server component should cancel the event.

   The return codes are also the same as those documented in the *DataWindow Reference*. Any of the service classes that implements the event can specify that the event be canceled.
4. Register the service classes for the component.

   There are two ways to make the Java class available as a service class:

   - For any component in EAServer or another supported server such as COM+, call the SetServerServiceClasses method in the Web page template's server-side script:

```
dwGen.SetServerServiceClasses
       ("UpdateValidate;RetrieveProcess");
```

- For a custom component in EAServer, add this property in EAServer Manager:

  ```
  com.sybase.datawindow.serverServiceClasses
  ```

  Set its value to the list of user object names, with names separated by semicolons. For example:

  ```
  UpdateValidate;RetrieveProcess
  ```

Using server component methods with the DTC

If you are using the Web DataWindow DTC or the Web Target object model, you can use methods of the server component by using the Component property. For example:

```
webDW.connectToComponent();
webDW.Component.SetServerServiceClasses
       ("UpdateStart");
```

Example

Suppose that you want to check that data did not exceed a budgeted total before it was updated in the database. You might set up a service class that implements the UpdateStart event.

The method declaration would be:

```
public void UpdateStart (UpdateEvent event,
   DataStore ds)
```

The body of this method has a DataStore that retrieves data from a budget table and compares it to the component's data:

```
import powersoft.datawindow.event.*;
import powersoft.datawindow.*;

public void UpdateStart (UpdateEvent event, DataStore
ds)
{
       DataStore ds_budget;
       ds_budget = new DataStore();
       ds_budget.setSourceFileName
           ("c:\\mydirectory\\mypbd.pbd");
       ds_budget.setDataWindowObjectName("d_object");
       ds_budget.setTransObject(...);
       ds_budget.retrieve( );
       // Get data to be validated
       int rowcount = ds.getRowCount();
       int total = 0;
```

```
            for (int i = 1; i<=rowcount;i++){
                total=total + ds.getItemNumber(i, "expenses",
                    ds.Primary);
            }
            String expense_total = ds_1.describe (...);
            double d_expense_total = Double.parseDouble
                (expense_total);
            if (d_expense_total<total){
                event.setReturnCode(1);
            }
        )
```

CHAPTER 8

Using the DataWindow Web Control for ActiveX

About this chapter

This chapter describes how to use the Sybase DataWindow Web control for ActiveX (Web ActiveX).

Contents

Topic	Page
About the Web ActiveX	211
HTML for inserting the controls on a Web page	215
How to use the Web ActiveX in a JSP target	219
DataWindow objects for the Web ActiveX	221
Using the DataWindow Transaction Object control	224
Making database connections	225
Coding for the Web ActiveX	227
Deploying the Web ActiveX	228

About the Web ActiveX

Features

The Sybase DataWindow Web control for ActiveX is a fully interactive DataWindow control for use with Microsoft Internet Explorer. It implements all standard DataWindow features except rich text.

The Web ActiveX supports data retrieval with retrieval arguments and data update. You can use edit styles, display formats, and validation rules. Most of the standard methods for manipulating the DataWindow are available, including Modify for changing DataWindow object properties. Several functions that involve file system interactions, such as SaveAs and SaveAsAscii, are not supported, allowing the Web ActiveX to be in the *safely scriptable* category of ActiveX controls.

Included with the Web ActiveX is the Sybase DataWindow Transaction Object control for making database connections that can be shared by several Web ActiveX controls.

Browser support
The Web ActiveX and Transaction Object control are designed to work in browsers that support ActiveX controls, such as Microsoft Internet Explorer version 3 and higher.

The DataWindow controls are not designed to work in Netscape browsers, which do not support ActiveX controls. Even if you use third-party plug-ins to enable ActiveX support, scripting for the controls works differently and is not tested.

Development environment

When you install PowerBuilder, the Setup program registers the Web ActiveX and the Transaction Object controls in the Windows registry. The class information entered in the registry is visible in the PowerBuilder Browser under OLE Custom Controls on the OLE tab. You can also examine the properties, events, and methods of the controls on the OLE tab of the Browser.

To use the Web ActiveX, your development system must meet the following requirements, all of which are met when you install any PowerBuilder component that requires the Java VM:

- The Sun JRE 1.2 or later must be installed on your system
- The path to the *jvm.dll* file must be in your system PATH environment variable

  The path is ...*JRE**bin**client* for JRE 1.4 and ...*JRE**bin**classic* for JRE 1.2 or 1.3
- The following files must be in a directory in your system PATH environment variable: *pbjvm110.dll*, *pbshr110.dll*, and *pbjdbc12110.jar*
- If you are using Internet Infomation Services (IIS) 6.0, you need to configure IIS to recognize the *.pbl*, *.pbd*, and *.psr* extensions as MIME types. See “Adding MIME types to IIS 6.0” on page 213.

In addition, the Java classes required by your database vendor's client layer must be installed on your system.

In a JSP target, you can develop JSP pages that use the Web ActiveX and Transaction Object controls. You can drag and drop these controls from the Components tab of the System Tree to a Web page. You can also examine the properties, events, and methods of these controls from the System Tree. In the HTML editor, you can insert the control, set property values using the control property sheets, and write event scripts in the script editor.

For more information, see "How to use the Web ActiveX in a JSP target" on page 219.

In addition to PowerBuilder's HTML editor, you can use any text editor to write HTML for the controls. However, other development environments are not supported and do not provide advanced authoring support.

For information about the required HTML, see "HTML for inserting the controls on a Web page" on page 215.

DataWindow objects

The Web ActiveX uses a DataWindow object, which determines what data is retrieved and how it is displayed. The Web ActiveX can also display Powersoft reports (PSRs), which do not need to retrieve data.

A DataWindow object for the Web ActiveX can have any presentation style except RichText.

DataWindow objects are stored in PowerBuilder libraries (PBLs) or PowerBuilder dynamic libraries (PBDs). The DataWindow libraries are stored on the Web server and downloaded as needed by the Internet Explorer browser. You use a URL to point to the library. It can be relative or absolute, using any supported protocol—for example, http, ftp, or file.

For more information, see "DataWindow objects for the Web ActiveX" on page 221.

Adding MIME types to IIS 6.0

To use the Web ActiveX with IIS 6.0, which is the IIS version included in Windows Server 2003, you need to configure IIS to recognize the *.pbl*, *.pbd*, and *.psr* extensions. Previous versions of IIS include a wildcard character MIME mapping that allows IIS to serve any file.

You must be a member of the Administrators group on the local computer to perform the following procedure, or you must have been delegated the appropriate authority. For more information, see the Microsoft documentation for IIS 6.0 or this Microsoft support document at http://support.microsoft.com/default.aspx?scid=kb;en-us;326965.

❖ **To add a global MIME type to IIS 6.0:**

1. In IIS Manager, right-click the computer on which you want to add a MIME type, and click Properties.
2. Click MIME Types.
3. Click New.
4. In the Extension box, type `pbl`.
5. In the MIME type box, type `application/octet-stream`.

6 Repeat steps 4 and 5 for the *.pbd* and *.psr* extensions, using `application/octet-stream` and `application/datawindow` respectively as the MIME types.

7 Click OK.

8 Restart IIS to apply the new settings.

You can also add a MIME type to a specific Web site or directory from its HTTP Headers property page.

Database connections

The Web ActiveX uses JDBC for database connectivity. Users can connect to a data source on any server to which they have access, including databases and middle-tier servers on the Internet.

You can use internal transaction properties for specifying a connection; or you can make the connection with a separate Transaction object, the Sybase DataWindow Transaction Object control. When you connect using a separate transaction object, you can control when SQL COMMIT and ROLLBACK statements occur and you can use the same connection for multiple Web ActiveX controls.

For more information, see "Using the DataWindow Transaction Object control" on page 224.

Scripting

Since the Web ActiveX is designed for Internet Explorer, you can use Jscript or another ECMAScript-compatible scripting language for scripting purposes.

In general, you can use the same methods as in a PowerBuilder application. However, there are a few differences:

- Datatypes are mapped to the basic JavaScript types of string, number, boolean, and various object types
- ECMAScript languages do not support arguments passed by reference; so instead of checking the value of a reference argument, you call a separate method to retrieve the value
- Enumerated datatypes are not supported; instead, specify the integer that corresponds to the enumerated data value

For more information, see "Coding for the Web ActiveX" on page 227.

Events

The Web ActiveX supports the same events as a standard DataWindow control, with these differences:

- Event names are different to conform to Web conventions
- Events in ECMAScript languages do not have return values; instead you can call SetActionCode to affect the outcome of an event

Deployment

The Web ActiveX is provided as a CAB file, which allows the client browser to install and register the control. When the user downloads a Web page that refers to the CAB file, the browser also downloads the CAB file if necessary, unpacks it, and registers the control. Some additional files must also be deployed to the client.

For more information, see "Deploying the Web ActiveX" on page 228.

HTML for inserting the controls on a Web page

You include the controls on a Web page with an Object element and associated Param elements. Then you write scripts that direct the control to make database connections, retrieve and update data, and respond to user actions.

The way the Object and Param elements look in a Web page are described next. When you insert the controls and set their properties using the JSP Target user interface in PowerBuilder, these elements are coded for you.

Object element

How it works

ActiveX controls use the Object element to specify the GUID (a unique identifier) of the control as well as the space the control takes on the page. The Object tag looks like this:

```
<OBJECT id=PSDWC1 height=357 classid="CLSID:BBBB1503-
BBBB-1000-8000-080009AC61A9" width=343>
</OBJECT>
```

In a JSP target, when you insert the Web ActiveX control by dragging from the System Tree or by using the Insert>Component menu item, the Object tag is written for you.

CODEBASE attribute

If your users need to download the controls, you can include the CODEBASE attribute in the Object element to identify the file to be downloaded. After the browser downloads the CAB file, it unpacks it and registers the ActiveX controls in the user's system registry.

The value for CODEBASE has the format:

url#version

A typical value for CODEBASE uses a relative URL:

```
CODEBASE="cabs/psdwc110.cab#10,5,0,3514"
```

URL The URL is the location of the DataWindow control's CAB file on your Web server. It can be an absolute or relative URL.

Version The version is a set of four numbers, separated by commas. The numbers must match the version of the CAB file. The version number of the CAB file is the same as the version number for PowerBuilder.

❖ **To find out the 4-part version number in Windows:**

1 Select the PowerBuilder executable or a PowerBuilder DLL in Windows Explorer.

2 Select File>Properties from the menu bar.

3 On the Version tab, look at File Version. A typical number is 11.0.0.3514.

Example

The Object element with a CODEBASE attribute looks like this:

```
<OBJECT codeBase=
"http://www.domain.com/psdwc110.cab#Version=10,5,0,
3514" id=PSDWC1 height=357 classid="CLSID:BBBB1503-
BBBB-1000-8000-080009AC61A9" width=343>
</OBJECT>
```

New versions

When you get new versions of the CAB file, you can change the version numbers on the Web page and cause the browser to install a new version of the control.

For more information about how to deploy new versions, see "Deploying the Web ActiveX" on page 228.

Putting the Object tag in a separate file

If the Cumulative Security Update for Internet Explorer (912812) or a subsequent update is installed on the computer where the Web ActiveX control is running, a browser refresh does not refresh the control correctly. This update is described in Microsoft Security Bulletin MS06-013, published in April 2006.

To work around this issue, put the <OBJECT> tag in a separate JavaScript file instead of the main HTML file, as shown in this example:

```
// HTML file
<HTML>
<HEAD>
   <TITLE>test</TITLE>
</HEAD>
<BODY bgColor="white" PSPARAMS="">
<P>Put your data here </P>
```

```
<P> </P>
<P>
   <div id="DivID">
      <script src="createElement.js"></script>
   </div>
</BODY>
</HTML>
```

The *createElement.js* JavaScript file contains the Object tag:

```
// createElement.js file
var d = document.getElementById("DivID");
d.innerHTML =
'<OBJECT id="OBJECT1" style="WIDTH: 627px; HEIGHT:
320px" codeBase="psdwc110.cab"
classid="CLSID:BBBB1503-BBBB-1000-8000-080009AC61A9">'
+'<PARAM NAME="_Version" VALUE="65536"></PARAM>'
+'<PARAM NAME="_ExtentX" VALUE="16589"></PARAM>'
+'<PARAM NAME="_ExtentY" VALUE="8467"></PARAM>'
+'<PARAM NAME="_StockProps" VALUE="2"></PARAM>'
+'<PARAM NAME="Caption" VALUE=""></PARAM>'
+'<PARAM NAME="SourceFileName"
VALUE="test.psr"></PARAM>'
+'<PARAM NAME="DataWindowObject"
VALUE="test.psr"></PARAM>'
+'<PARAM NAME="LogId" VALUE=""></PARAM>'
+'<PARAM NAME="LogPass" VALUE=""></PARAM>'
+'<PARAM NAME="dbParm" VALUE=""></PARAM>'
+'<PARAM NAME="SuppressEvents" VALUE="0"></PARAM>'
+'<PARAM NAME="VScrollBar" VALUE="0"></PARAM>'
+'<PARAM NAME="HScrollBar" VALUE="0"></PARAM>'
+'<PARAM NAME="HSplitScroll" VALUE="0"></PARAM>'
+'<PARAM NAME="LiveScroll" VALUE="0"></PARAM>'
+'</OBJECT>';
```

Properties and Param elements

How they work

The Web ActiveX and the Transaction Object control have several properties that specify connection information. The Web ActiveX also has properties that specify a DataWindow object or a PSR. You provide values for the properties with Param elements, which are enclosed in the Object element.

When you insert the controls in a JSP target, Param elements are automatically added to the Web page. You can provide values for the Param elements using the control property sheets. You can also write scripts that set or change property values.

The inserted Param elements are grouped in an order that corresponds to pages in the control property sheets. The first group of Param elements contains standard ActiveX properties. You can assign values for the standard ActiveX properties in the ActiveX OBJECT*n* Properties dialog box, where *n* is the order in which the object is placed on the page.

The Web ActiveX also inserts Param elements for custom properties. You set these custom properties from the Sybase DataWindow Web Control Properties dialog box. To open this dialog box, you can click the Control Properties button on the ActiveX page of the ActiveX OBJECT*n* Properties dialog box for the Web ActiveX control.

The Sybase DataWindow Web Control Properties dialog box has four tab pages where you can set custom properties: General, Scrolling, Transaction, and Trans User. The Transaction Object control has only two custom property pages: Transaction and Trans User.

Standard ActiveX properties

```
<PARAM VALUE=65536 NAME="_Version"></PARAM>
<PARAM VALUE=9280 NAME="_ExtentX"></PARAM>
<PARAM VALUE=5155 NAME="_ExtentY"></PARAM>
<PARAM VALUE=2 NAME="_StockProps"></PARAM>
<PARAM VALUE="" NAME=Caption></PARAM>
```

Properties on General page

```
<PARAM VALUE="javadwtest.pbl" NAME=SourceFileName>
</PARAM>
<PARAM VALUE="d_emp" NAME=DataWindowObject></PARAM>
<PARAM VALUE=0 NAME=SuppressEvents></PARAM>
```

Properties on Scrolling page

```
<PARAM VALUE=0 NAME=VScrollBar></PARAM>
<PARAM VALUE=0 NAME=HScrollBar></PARAM>
<PARAM VALUE=0 NAME=HSplitScroll></PARAM>
<PARAM VALUE=0 NAME=LiveScroll></PARAM>
```

Properties on Transaction page

```
<PARAM
VALUE="Driver='com.sybase.jdbc2.jdbc.SybDriver',
URL='jdbc:sybase:Tds:localhost:2638'" NAME=dbParm>
</PARAM>
```

Properties on Trans User page

```
<PARAM VALUE=dba NAME=LogId></PARAM>
<PARAM VALUE=sql NAME=LogPass></PARAM>
```

How to use the Web ActiveX in a JSP target

The PowerBuilder Setup program installs the Web ActiveX and the Transaction Object control in the system registry. You can find them in the System Tree on the Components tab.

These procedures provide a summary of the ways you author Web pages with ActiveX controls in a JSP target.

❖ **To find the controls in the System Tree:**

1 Select the Components tab.

2 Expand ActiveX Controls.

3 Expand Controls.

4 Look for the controls with these names:

- Sybase DataWindow Web Control 11.0
- Sybase DataWindow Transaction Object Control

These controls have Properties and Methods branches (the Web ActiveX also has an Events branch) that you can expand to view available properties and methods.

❖ **To add the controls to a Web page:**

- Drag the control name from the System Tree to the Web page
  or
  Select Insert>Component>ActiveX from the menu bar, select the Controls page, make sure that Controls is selected in the drop-down list box, and select the control (Sybase DataWindow Web Control or Sybase DataWindow Transaction Object Control) you want.

❖ **To set standard ActiveX properties for the control:**

1 *In the Page view*, right-click the control and select ActiveX OBJECT*n* Properties from the pop-up menu, where *n* is the order in which the object is placed on the page. When you specify a name for the control, the menu item uses that name instead of OBJECT*n*.
or
In Source view, right-click the Object tag and select Properties from the pop-up menu.

2 In the ActiveX Properties dialog box, specify the values you want. Some of the more common attributes to set include:

- Network Code Base—its value is the URL and version number of the CAB file that the browser downloads to install the ActiveX control

  For more about the CODEBASE attribute, see "HTML for inserting the controls on a Web page" on page 215.

- Width and Height—also set by dragging the control's resizing handles in Page view
- Name for forms and scripting

❖ **To set custom properties for the controls:**

- In the Page view, right-click the control and select Sybase DataWindow Web Control Properties (or Sybase DataWindow Transaction Object Control Properties) from the pop-up menu.

  For information on specifying the DataWindow object on the General tab page, see "Specifying a DataWindow object for the control" on page 222.

  For information about specifying connection information on the Transaction and Trans User tab pages, see "Making database connections" on page 225.

  The custom properties dialog box is not available from the Source view.

Where to set connection properties
You need to set connection properties only once. If you are using a separate Transaction Object control, you do not need to set properties on the Transaction and Trans User pages for the Web ActiveX. The same property pages are available for the Transaction Object control.

❖ **To use the System Tree when writing code for the controls:**

1 On the Page tab of the System Tree, expand *window* in the Client Side branch.

2 Under *window*, expand Properties and find the name of your control as one of the properties.

3 For the control, expand Properties, Events, or Methods.

4 Find the item you want to insert in the script editor and drag it to its position in the code.

A fully qualified expression is inserted, such as:

```
window.dw1.Describe(propertylist)
window.dw1.dbParm
```

Substitute your own variables and expressions for method parameters.

DataWindow objects for the Web ActiveX

The DataWindow Web control for ActiveX requires either of these:

- A DataWindow object stored in a PBL or PBD
- A PSR that was saved with data

Properties for the Web ActiveX identify the DataWindow object that you want to display in the control.

This section describes considerations for:

- Defining DataWindow objects
- Building libraries
- Identifying the DataWindow object or PSR by setting Web page properties

What the DataWindow object can include

You define DataWindow objects in PowerBuilder. The Web ActiveX supports all DataWindow presentation styles except RichText.

You can use all edit styles, including DropDownDataWindow. Properties can have conditional expressions (written in PowerScript) and computed fields can use any of the functions available to a standard DataWindow.

In the Web page, you can include scripts (written in JScript or other ECMAScript-compatible scripting languages) to manipulate DataWindow presentation and data.

Managing DataWindow objects in PowerBuilder libraries

Types of libraries

The Web ActiveX can use DataWindow objects that are stored in PBLs or PBDs.

When to use a PBD

The Web browser downloads the library specified for the Web ActiveX and stores it in a temporary cache. If you do not want your DataWindow object source code to be available to the user (who could copy it from the cache), convert a PBL to a PBD before deploying it.

Because the library will be downloaded, you should make it as small as possible—another good reason to convert a PBL to a PBD.

PowerBuilder version
PBLs must be migrated to Version 10 or later.

Grouping DataWindow objects into libraries

A PBL or PBD is downloaded in its entirety from the Web server; therefore, you should make sure your library includes only those objects needed on your Web pages. You can group DataWindow objects that are used on different Web pages in a single library; however, you should avoid forcing users to download objects used on pages they will not view.

When choosing how to group DataWindow objects into libraries, make the set of objects in the library correspond to the typical set of pages the user will view. Although a single download saves a lot of communications overhead, it is worthwhile for users only if they view the pages that use the objects.

Using other resources in the DataWindow object

A DataWindow object can use external resources such as bitmaps or cursors. You can use a resource file when you build a PBD to include these resources in the library. You can also store these resources on the Web server. A relative path in the DataWindow object can point to the file's location on the Web server. The browser retrieves the resource as needed.

A DataWindow object can use other DataWindow objects, such as drop-down DataWindows. Make sure these objects are included in the downloaded library.

Specifying a DataWindow object for the control

To identify the DataWindow object you want to display in the control, you specify values for two properties:

- SourceFileName
- DataWindowObject

You enter their values on the General page of the Sybase DataWindow Web Control Properties dialog box.

About SourceFileName

The value for SourceFileName is the name of the library that contains the DataWindow object for the control. It can be a URL or a file path. These examples illustrate some typical variations:

- Absolute URL:

```
http://www.domain.com/dwlibraries/financedws.pbd
```

- Relative URLs:

```
financedws.pbd
dwlibraries/financedws.pbd
```

- Absolute file path (can be useful while developing pages):

```
d:/web project/dwlibraries/financedws.pbd
```

In the Web ActiveX property sheet, when you use the Browse button to look for the library, you browse the file system, not URLs. After the full path is inserted in the field, you probably should edit it so that it is valid when your Web pages are deployed.

JSP targets When you are working in a PowerBuilder JSP target, it is a good idea to include both the library and the JSP page in the workspace. If you do, SourceFileName can be a relative path that is valid in both your design and deployment environments.

Changing SourceFileName during execution You can change the value of SourceFileName in a script. If you do, you also have to specify a value for DataWindowObject that is valid in the new library. You must also call SetTransObject again if you are using a separate transaction object.

About DataWindowObject

The value for DataWindowObject is the name of a DataWindow object that is in the library specified in SourceFileName.

If the library is accessible in the development environment (for example, it is part of the PowerBuilder workspace and you specify a relative URL), then the property sheet displays a drop-down list of the DataWindow objects contained in the library.

Displaying Powersoft reports
To display a PSR file instead of a DataWindow object, specify its URL as the value for the DataWindowObject and leave the SourceFileName blank.

Using the DataWindow Transaction Object control

What it does

The DataWindow Transaction Object control allows you to establish a database connection independent of the Web ActiveX. It is similar to the PowerBuilder Transaction object.

Internal transaction management or separate Transaction object

Both the Web ActiveX control and the Transaction Object control can establish a database connection. The one you use depends on your needs.

There are two main reasons to use the Transaction Object control:

- You can make one database connection for several Web ActiveX controls, saving the overhead of multiple connections.
- You can control transaction processing with Connect and Disconnect methods, equivalent to the SQL statements CONNECT and DISCONNECT. If the AutoCommit property is set to false, you can control when an update is committed or rolled back (by calling the Commit and Rollback methods).

If you have only one control and are simply retrieving data, you do not need either of these features. Instead of instantiating a separate control, you can set the connection properties of the Web ActiveX itself and allow it to connect and disconnect for each database access.

Status and error information

The Transaction Object control receives status information from the database. You can test the success or failure of a database operation and get status information with these methods, which are equivalent to PowerBuilder transaction object properties:

- GetSQLCode
- GetDBCode
- GetSQLErrText
- GetSQLNRows
- GetSQLReturnData

Hiding the Transaction Object control

The Transaction Object control has no visual aspect, but if it is in the BODY section of the Web page, it still takes up space. You can set its HEIGHT and WIDTH attributes to very small values or use stylesheet settings to make it invisible.

If you put it in the HEAD section, you will not have access to its property sheet in a PowerBuilder JSP target.

For information on setting properties for making a database connection, see "Making database connections" next.

Making database connections

The connection process

The Web ActiveX and Transaction Object controls make database connections using JDBC. Their Java classes interact with the Java classes of the database vendor's JDBC interface. The vendor's classes interact with the database.

JDBC driver from a database vendor

The classes for the JDBC database driver you plan to use must be available to the user's browser. If the user does not have them installed already, you can set up the Web page so that they are downloaded and installed just as you do for the Web ActiveX.

To have the JDBC driver classes installed automatically, you can:

1 Convert the database vendor's Java classes into a CAB file. You can use a Microsoft utility called CABARC to do the conversion. The classes can be in a ZIP archive or directory tree.

2 Add an Object element with a CODEBASE attribute for the CAB file to the Web page.

The browser downloads the CAB file and adds the classes to its internal class path. There is no change made to the CLASSPATH environment variable.

If you want to use the JDBC driver when you are defining DataWindow objects, you need to do some additional installation, such as putting the JDBC driver classes on the system class path.

For more information, see *Connecting to Your Database* or the online Help for PowerBuilder.

Connection properties

The connection information for the Transaction Object or Web ActiveX is set as Param elements enclosed in the Object element. Whether you use a separate Transaction object or the internal connection properties of the Web ActiveX, the connection properties are the same:

Table 8-1: Connection properties of the Transaction Object and Web ActiveX controls

Param name	Meaning	Typical value
LogID	The ID needed to log in to the database.	dba (default ID for ASA databases)
LogPass	The password needed to log in to the database.	sql (default password for ASA databases)
dbParm	A string specifying the Java classes for the driver and the URL for the database.	For the JConnect driver: Driver='com.sybase.jdbc.SybDriver', URL='jdbc:sybase:Tds:199.1.1.1:9999/mydatabase'

Param name	Meaning	Typical value
Lock	The isolation level of the connection.	Vendor-specific values
AutoCommit	Whether a commit occurs immediately after the database is updated.	False (default)

About dbParm

For JDBC drivers, the dbParm property specifies essential connection information. Its value is a string that contains at least two values. Those values identify the driver you want to use and the URL of the database, in a format understood by the driver.

The format is:

Driver='*JDBCclassname*',URL='*database_url* '

To find the class name for Driver and the format of the database URL, check the documentation from the DBMS vendor.

For examples of setting these properties in Param elements, see "Properties and Param elements" on page 217.

JConnect in the development environment

If you want to use JConnect when you define DataWindow objects, put the Java classes for JConnect on the class path.

Connecting and retrieving data

To connect and retrieve data, you must write a script. The script can belong to a Retrieve button, or you can have the retrieval occur automatically by putting the code in the window's onLoad script.

For example, to connect and retrieve data for a Web ActiveX named *dw_1*, using a DataWindow Transaction Object control named *trans_1*, your script would be similar to this:

```
trans_1.Connect( );
dw_1.SetTransObject( trans_1 );
dw_1.Retrieve( );
```

When you use the internal transaction properties, the Web ActiveX makes the connection automatically. The script can be simpler, like this:

```
dw_1.Retrieve( );
```

Coding for the Web ActiveX

You can write scripts in the Web page to manipulate the DataWindow data and presentation. The methods and events are similar to those available in a standard DataWindow, but the events have been renamed to match JavaScript naming conventions.

The properties, methods, and events—as well as the DataWindow object properties and expression functions—are documented in the *DataWindow Reference*.

You can see a list of the properties, methods, and events for the controls on the Components page of the System Tree or in another tool for examining ActiveX controls.

The Script editor lets you write scripts for all events of the Web ActiveX.

❖ **To write a script for a particular event in:**

1. In Page view, select a Web ActiveX that you have inserted in the Web page
   or
   In the Script editor, select the name of the DataWindow Web ActiveX in the leftmost drop-down list.
2. In the Script editor, select an event from the second drop-down list. The drop-down list shows the event name and the parameters whose values are available in the script.
3. In the third drop-down list, select JScript as the scripting language.
4. Write code for the event. To call methods or access properties without typing, drag them from the Page tab of the System Tree to the editor.

Datatypes for method arguments and return values

Primitive types

JScript supports three primitive datatypes:

- string
- number
- boolean

Method arguments and return values and event parameters are one of these basic types, or an object type.

DataWindow methods that deal with specific datatypes, such as GetItemDecimal, are not available for the Web ActiveX. Instead, you use the method that handles the more general data type, such as GetItemNumber.

Date datatypes

PowerBuilder has several date and time datatypes, but in JScript these all map to the Date object.

Enumerated datatypes

PowerBuilder enumerated datatypes have named values, but in JScript, each value is a number. The list of numbers (and their meaning) is documented in the *DataWindow Reference* for each enumerated data type.

Setting event return codes

The event return codes documented for DataWindow events are also valid for the Web ActiveX. However, JScript does not support return values for events. Instead, to specify a return code, you call the SetActionCode method as the last line in the event script.

For example, the return code of the onItemError event allows you to determine what happens when user-entered data fails a validation rule. By specifying a return code of 3, you cause the Web ActiveX to reject the data but allow focus to change. This statement would be the last line of the onItemError event script:

```
This.SetActionCode(3);
```

Deploying the Web ActiveX

CAB file for deployment

The PowerBuilder Setup program installs the *PSDWC110.CAB* file in the *Sybase\Shared\PowerBuilder* directory. This CAB file contains files and information the client Web browser (Internet Explorer) needs for installing the Web ActiveX and the Transaction Object control. The CAB file includes:

- An Open Software Distribution information file
- DLLs for the controls

The browser manages installation of the ActiveX controls using the information in the CAB file. It installs the controls in the system registry and maintains its Java class path. If your Web ActiveX control uses a PSR file, you need only to tell the browser how to find the CAB file.

❖ **To make the Web ActiveX and Transaction Object control available to users:**

1 Put the CAB file on your Web server.

If the Web server uses Internet Infomation Services (IIS) 6.0, you need to configure IIS to recognize the *.pbl*, *.pbd*, and *.psr* extensions as MIME types. See "Adding MIME types to IIS 6.0" on page 213.

2 Refer to the CAB file on the Web page in the CODEBASE attribute of the Object element.

In a JSP target, you can accomplish both goals by:

- Including the CAB file in your project and deploying it to the Web server along with your other project files.
- Specifying a value for Network Code Base in the ActiveX object property sheet. The value is the name and relative path for the CAB file, plus the version number. If the CAB file is part of your PowerBuilder JSP target, use the browse button to display the Choose URL dialog box. You can then select the CAB file on the Project page of this dialog box.

  For information about specifying the correct value for CODEBASE, see "HTML for inserting the controls on a Web page" on page 215.

If your Web page uses a JDBC connection, the Web ActiveX has additional deployment requirements:

- The Sun JRE 1.2 or later must be installed on the client. Users can download the latest version of the JRE from the Sun Java Web site at *http://www.java.com:80/en/download/windows_manual.jsp* (where *en* indicates the English language version).
- The path to the file *jvm.dll* (...*JRE**bin**client* for JRE 1.4 and ...*JRE**bin**classic* for JRE 1.2 or 1.3) must be added to each user's system PATH environment variable.
- The following files must be in a directory in the client's system PATH environment variable: *pbjvm110.dll*and *pbshr110.dll*.
- The *pbjdbc12110.jar* file, which contains class files required by the Web ActiveX, must be deployed to the client. You can deploy the JAR file by referencing it in the CODEBASE attribute of the Object element in your Web page.

- Java classes required by your database vendor's client layer must be available on the client. They can be added to a CAB file that is referenced in the CODEBASE attribute of the Object element in your Web page. For example, if you are using Sybase jConnect to connect to a database, the *jconn2.jar* file should be included in the CAB file. If the client layer is provided in a JAR file, it can be referenced directly in the CODEBASE attribute.

Deploying a new version

The version number in the value of the CODEBASE attribute determines whether the browser downloads and installs a new version of the ActiveX controls. The browser compares the version number in the CODEBASE value with the version of the controls that is installed in the system registry. If the version numbers do not match, the browser downloads the CAB file again and installs it.

❖ **To deploy the new CAB file when you get a new version of PowerBuilder:**

1 Find the new version number by checking the version number of any PowerBuilder DLL (as described in "HTML for inserting the controls on a Web page" on page 215).

2 Edit the Web pages that refer to the CAB file. Change the version number in the CODEBASE attribute to match the new number.

3 Replace the CAB file on the Web server with the new version.

Index

A

absolute positioning, Netscape browser 143
accent marks, Web DataWindow 139
action codes 41
aggregation functions, Web DataWindow 138
applications
  architectures 7
  using DataWindow objects in 13

B

bitmaps, dynamically adding and removing 69
buffers
  DataStore 85
  DataWindow 30, 44
Button controls, Web DataWindow 141

C

CAB file, Web control for ActiveX 228
CallBack paging 134
Case function, Web DataWindow 138
character set, foreign language 139
Clicked events in graphs 103
client control, Web DataWindow 114
client/server applications 7
client-side paging 134
CODEBASE HTML attribute, Web ActiveX 215
column status in DataWindow controls 44
COMMIT statement and SetTransObject 24
communication with databases 21
CONNECT statement and SetTransObject 24
connection caches, defining in EAServer 148
controls, supported in Web DataWindow 124
create capability for Modify 69
Create method 69
CrosstabDialog function 52
crosstabs
  modifying during execution 54
  users redefining during execution 52
  using in applications 52
  viewing underlying data 52
custom DataStore objects 82

D

data
  retrieving and updating 26
  saving in graphs 99
  sharing 85
  updating 27
  Web control for ActiveX 226
  Web DataWindow 124, 187
data sources
  external 29, 84
  types 6
database connections
  about 21
  Web control for ActiveX 225
  Web DataWindow 187
database errors 39
databases
  communicating with 21
  connecting automatically 22
  data source 6
  disconnecting automatically 22
  retrieving, presenting, and manipulating data 4, 26
  snapshot connections 22
  transaction management 24
  updating 27
DataModified status 44
DataObject property of DataWindow controls 19
DataStore objects
  about 4
  accessing data 84
  buffers 85

custom 82
importing data from external sources 84
methods 85
sharing data 85
DataWindow controls
about 3, 8, 11, 14
accessing a specified item 34
accessing the current text 32
action codes 41
and graphs 97
assigning transaction objects to 70
associating with objects during execution 19, 20
buffers 30, 44
column status 44
creating reports with 47
data management in 30
DataObject property 19
DBError event 40
displaying PSR files in 19, 20, 21
handling errors 39
importing data from external sources 29
ItemChanged event 33
ItemError event 34
methods 36
names 15
naming in code 16
placing in windows 15
processing entries 31
row status 44
updating, use of row/column status when 44
using crosstabs 52
using graph methods 101
DataWindow execution time errors 42
DataWindow objects
about 3
associating with controls 17, 19, 20
basic use of 13
creating dynamically 69
creating reports with 47
data sources 6
defining 5
designing for Web DataWindow 123
displaying data 21
dynamic use of 67
editing 18
generating HTML from 55
graphs in 93
HTML preview 144
names 15
overview 4
preparing to use 13
presentation styles 5
printing multiple on a page 50
properties of 37
PSR file for Web DataWindow 185
SRD file for Web DataWindow 185
Web DataWindow, set in script 185
Web DataWindow, set on server 196
DataWindow painter
about 13
editing DataWindow object 18
working in 13
DataWindow plug-in 4, 10
DataWindow technology 3, 7
DataWindowObject property, Web control for ActiveX 20, 222
DBError event 40
Delete buffer
DataStore 85
DataWindow 30
Describe method 37, 68, 70, 71
destroy capability for Modify 69
DISCONNECT statement and SetTransObject 24
display formats in Web DataWindow 125
distributed applications 7
dynamic DataWindow objects
about 67
adding elements 69
creating 69
modifying 68
providing query mode 72
specifying create syntax 70, 71

E

EAServer
database connection caches 148
HTML generator components 113
installing custom component 196
instantiating custom component 200

locked PBL 195
maintaining state for Web DataWindow 201
edit controls
in DataStore objects 85
in DataWindow controls 30, 32, 33
edit styles
overriding in query mode 76
Web DataWindow 124
EditChanged event 32
Error event 42
errors, following database retrieval or update 39
events
action codes 41
DBError 40
Error 42
ItemChanged 33
ItemError 34
Web control for ActiveX 227, 228
Web DataWindow client control 154
execution
accessing graphs 94
associating DataWindow objects with controls 19, 20
modifying DataWindow objects 68
Export/Import Template view icons 159
expressions, assigning DataWindow property values 68
External data source, importing data 29, 84
external functions in Web DataWindow 138

F

files as data source 6
Filter buffer 30, 85
fonts, using in reports 48
foreign language character set 139

G

Generate method, example 193
GenerateHTMLForm method 63
GetChild method 50
GetItemDate method 34
GetItemDateTime method 34
GetItemDecimal method 34
GetItemNumber method 34
GetItemString method 34
GetItemTime method 34
GetMessageText method 53
GetText method 32
graphics, adding to DataWindow objects 69
graphs
about 93
data properties 97
getting information about 97
internal representation 95
modifying data properties in DataWindow control 101
modifying display of data 100
modifying during execution 94
properties of 95
saving data 99
grAxis subobject of graphs 95
grDispAttr subobject of graphs 95

H

header section in XML template 165
Help, providing in dynamic DataWindow objects 77
HTML
appending to a control 140
generating forms 63
including in a control 140
saving DataWindow data as 55
HTML Preview 144

I

InfoMaker Report painter 5
instance pooling, EAServer 195
ItemChanged event 32, 33
ItemError event 32, 34
items in DataWindow controls 31

J

JavaScript caching, Web DataWindow 135
JSP targets
  class path 179
  custom tag libraries 181

L

language, character sets 139
libraries
  for DataWindow objects 12, 13
  locked by EAServer 195
  Web control for ActiveX 222
LibraryExport function 71

M

MessageText event 53
methods
  DataStore 85
  DataWindow 36
  graph 97
  JavaScript caching 135
  Web control for ActiveX, listed in System Tree 219
  Web DataWindow client control 155
  Web DataWindow server component 182
Modify method
  basic usage 37, 68
  using query mode 72
  using with graphs 95
  with crosstabs 54
multiple Web DataWindows on a page 155

N

names of DataWindow controls and DataWindow objects 15
nested reports
  creating during execution 51
  destroying during execution 51
  using in applications 50
Netscape browser, absolute positioning 143
New status 44
NewModified status 44
NotModified status 44

O

Object element, Web control for ActiveX 215
OneTrip method, Web DataWindow 200

P

PagingMethod property 133
painters 13
Param elements, Web control for ActiveX 217
PBD files *see* libraries 12
PBL files *see* libraries 12
Picture button in Web DataWindow 142
Picture controls in Web DataWindow 142
plug-in, DataWindow 10
point and click, in graphs 103
PowerBuilder libraries *see* libraries 12
Powersoft reports
  about 5, 13
  DataWindow plug-in 10
  displaying in DataWindow controls 19
  displaying in Web control for ActiveX 20
  no database connection needed 21
  Web application 10
presentation styles
  list 5
  supported in Web control for ActiveX 221
  supported in Web DataWindow 123
Primary buffer 30, 85
Print method 48
printing
  multiple DataWindow objects on a page 50
  reports 48
programs, using DataWindow objects in 13
properties
  DataWindow object 37
  retrieving current values of 68, 70, 71
PSR files *see* Powersoft reports

Q

query mode
clearing 75
forcing equality 76
providing to users 72
sorting in 76
quotation marks in self-link expressions 190

R

reports
see also Powersoft reports 5
creating with DataWindow objects 47
nested 50
printing 48
Retrieve method
handling errors 39
using 26
Web DataWindow example 188
RetrieveEx method, example 188
ROLLBACK statement and SetTransObject 24
rows
providing user-specified retrieval 72
status in DataWindow controls 44
runtime libraries *see* libraries 13

S

saving data in graphs 99
scripts, modifying graphs in 94
SELECT statements, modifying at execution time 75
selection criteria *see* query mode 72
self-link arguments, Web DataWindow 189
separator line in XML template 165
server component
accessing from server-side scripts 182
properties 197
Web DataWindow 179
server component, Web DataWindow 113
server-side validation for Web DataWindow 203
service classes for Web DataWindow 203
SetAction method, example 192
SetBrowser method, example 187
SetDWObject method, example 185
SetHTMLObjectName method, example 186
SetItem method 34
SetText method 32
SetTrans method
about 22
Web DataWindow example 187
SetTransObject method 24
SetWeight method, example 186
ShareData method 50
sorting in query mode 76
SourceFileName property, Web control for ActiveX 20, 222
SRD file for Web DataWindow 185
status of DataWindow rows or columns 44
SyntaxFromSQL method 71

T

text controls in DataWindow objects 69
text in DataWindow edit control 30
transaction objects
reassociating DataWindow controls with 70
Web control for ActiveX 224

U

Update method
handling errors 39
using 27
URLs, Web control for ActiveX 223
user events, for graphs in DataWindow controls 101
user objects, inherited from DataStore objects 83

V

validation rules in Web DataWindow 125
validation, server-side 203

W

Web applications and DataWindow technology 8, 109, 211
Web browsers
  DataWindow plug-in 10
  Web control for ActiveX 212
  Web DataWindow 143, 187
Web control for ActiveX
  about 9, 211
  CODEBASE HTML attribute 215
  database connections 225
  DataWindow objects 221
  DataWindowObject property 20, 222
  deployment 228
  event return codes 228
  libraries 222
  library versions 222
  Object element in HTML 215
  retrieving data 226
  setting DataWindow object during execution 223
  SourceFileName property 20, 222
  transaction objects 224
  version number of CAB 216
  Web browsers 212
  Web target 219
  writing scripts 227
Web DataWindow
  about 8, 109
  aggregation functions 138
  browsers 143
  browser-specific HTML 187
  client-side scripts 154
  custom server component 196
  data manipulation 141
  database connections 187
  DataWindow objects 123
  design-time control 152
  EAServer connection cache 147
  events for client control 154
  expressions 138
  foreign language text 139
  generating HTML 193
  how it works 111
  HTML generator components 113
  HTML version 143
  installing custom component 196
  instantiating custom component 200
  JavaScript caching 135
  JavaScript Generator wizard 135
  locked PBL 195
  maintaining state 201
  methods for client control 155
  multiple, on a page 155
  navigation 141
  object name 186
  OneTrip method 200
  page parameters 189, 192
  picture button 142
  process 112
  programming for server component 182
  PSR file 185
  reloading page 189
  retrieving data 188
  self-link information in scripts 189
  self-link properties 141
  server component and client control 113
  server component properties 197
  server component, about 179
  server-side scripts 182
  service classes 203
  setting DataWindow object during execution 185
  size of generated code 186
  SRD file 185
  types 110
  user actions 192
  where to install 177
Web DataWindow Container wizard 149
Web Target object model 152
Window painter, placing DataWindow controls 15
wizards
  Web DataWindow Container 149
  Web DataWindow JavaScript Generator 135

X

XHTML export template
  creating and saving 160
  Detail Start element 165
  editing 167
  exporting 174, 175
  saving 162

XML Web DataWindow
benefits 119
how it works 117
how to use 121
using 115
XMLClientSide paging 134