DataWindow® Programmers Guide

Appeon PowerBuilder®

2017

DOCUMENT ID: DC37775-01-1700-01

LAST REVISED: June 2017

Appeon Limited, 1/F, Shell Industrial Building, 12 Lee Chung Street, Chai Wan District, Hong Kong.

Contents

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® *Users Guide*.
- You have a basic familiarity with the PowerScript® language.

Related documents

For a description of all the books in the PowerBuilder documentation set, see the preface of *PowerBuilder Getting Started*.

Other sources of information

Use the Appeon Product Manuals web site to learn more about your product. The Appeon Product Manuals web site is accessible using a standard Web browser.

To access the Appeon Product Manuals web site, go to Product Manuals at https://www.appeon.com/developers/library/product-manuals-for-pb.

The installation guide in PDF format can be accessed from the PowerBuilder installation package. The release bulletin can be access from Online Help at https://www.appeon.com/support/documents/appeon_online_help/pb2017/release_bulletin_for_pb.

If you need help

All customers are entitled to standard technical support for reproducible software defects. You can open a standard support ticket at the Appeon support site: https://www.appeon.com/standardsupport/ (login required). If your organization has purchased a premium support contract for this product, then the designated authorized support contact(s) may seek assistance with your technical issue or question at the Appeon support site: https://support.appeon.com (login required).

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 *Users 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

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 invented for use in PowerBuilder to provide powerful data retrieval, manipulation, and update capabilities for client/server applications.

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.
Graph	Graphical presentation of data.
Crosstab	Data summary in a row-and-column format.
RichText	Paragraphs of text with embedded data columns.
OLE	An OLE object linked or embedded in the DataWindow and associated with the retrieved data.

For examples of the presentation styles, see the *PowerBuilder Users 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.

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 8.

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 Users Guide* and *Application Techniques*.

CHAPTER 2

Using DataWindow Objects

About this chapter

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

Contents

Before you begin

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

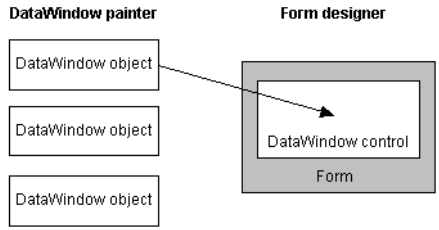
Putting a DataWindow object into a control

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 2, Using DataWindow Objects. For more information about drop-down DataWindows and composite DataWindows, see the *PowerBuilder Users 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

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

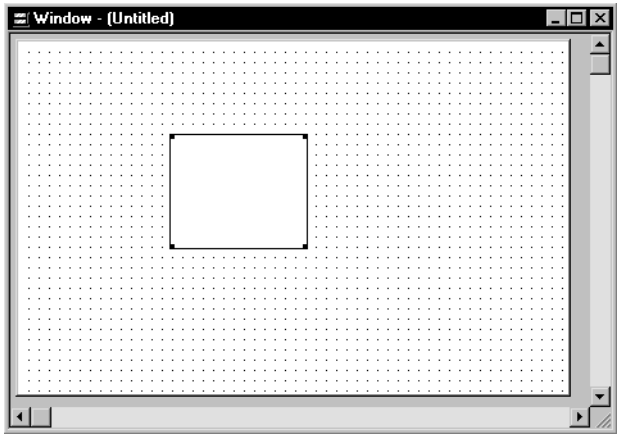
❖ **To place a DataWindow control in a window:**

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 Users 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:**

1. 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

In PowerBuilder, set the DataObject property to one of the DataWindow objects built into the application.

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 18.

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.

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"

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.

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*.

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.

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( )
```

For more information

For more information about database transaction processing:

- **PowerBuilder** See the chapter on using transaction objects in *Application Techniques*

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( )
```

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 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.

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.

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 37.

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;
```

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 33.

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

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

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.

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

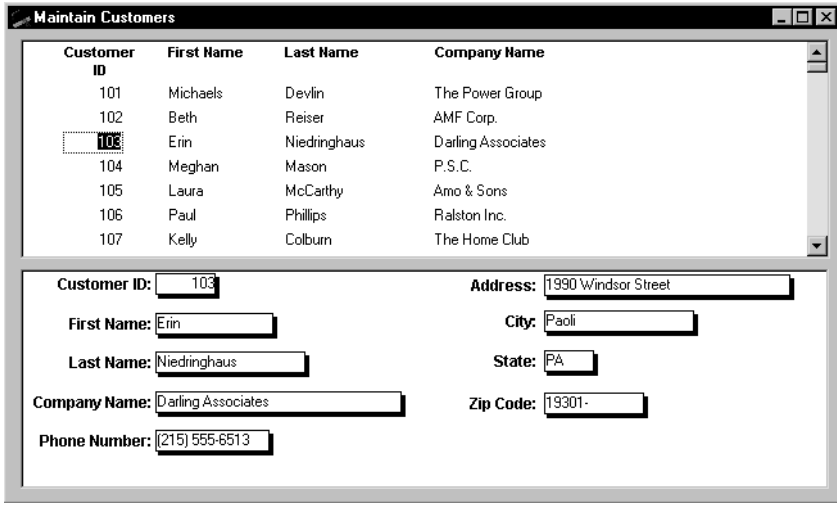
A DataWindow uses three buffers to store data:

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.

Table 2-2: Event names in PowerBuilder

Event	Description
EditChanged (not available on client control)	Occurs for each keystroke the user types in the edit control
ItemChanged	Occurs when a cell has been modified and loses focus
ItemError	Occurs when new data fails the validation rules for the column
ItemFocusChanged	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:

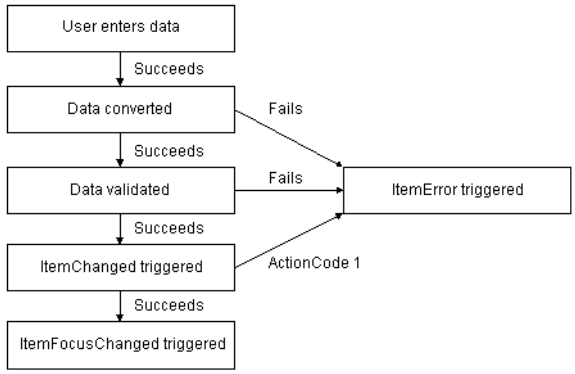
1 The DataWindow control converts the text into the correct datatype for the column. For example, if the user is in a numeric column, the DataWindow control converts the string that was entered into a number. If the data cannot be converted, the ItemError event is triggered.

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** Use a RETURN statement

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 28.

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:

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

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

- **PowerBuilder** SetItem

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.

In PowerBuilder, you construct data expressions using dot notation.

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

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.

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")
```

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 24)
- 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*

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:

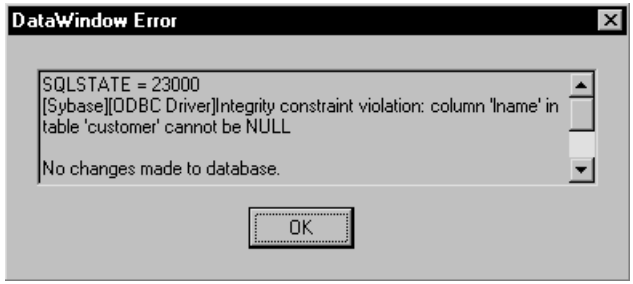
```
IF dw_emp.Update() > 0 THEN
```

```
        COMMIT USING EmpSQL;
ELSE
        ROLLBACK USING EmpSQL;
END IF
```

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.

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:

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	**Numeric value**	
New!	2	Rows
NewModified!	3	Rows
NotModified!	0	Rows and columns
DataModified!	1	Rows and columns

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

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 Users 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 Appeon 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 Users 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 12.

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 42.

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.

To learn about designing nested reports, see the *PowerBuilder Users 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 Users 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 Users 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 Users Guide*.

Letting users redefine the crosstab

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.

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 Users 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.)

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, richtext)	Column
Font	Column, Compute, Text
Format	Column, Compute
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.

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.

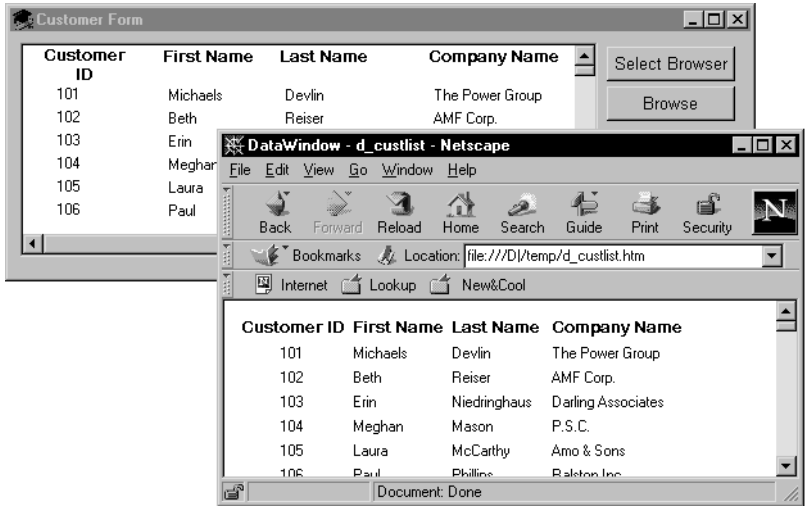
The key line of code gets the HTML from the DataWindow by referring to its HTMLTable property. In PowerBuilder, you can use the Describe method or a property expression.

PowerBuilder

```
ls_htmlstring = dw_1.Object.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.

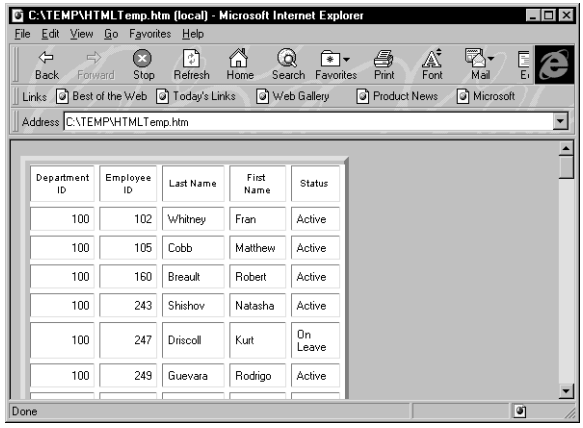
```
String ls_HTML, ls_FileName, ls_BrowserPath
Integer li_FileNumber, li_Bytes,
Integer li_RunResult, li_Result

// 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

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

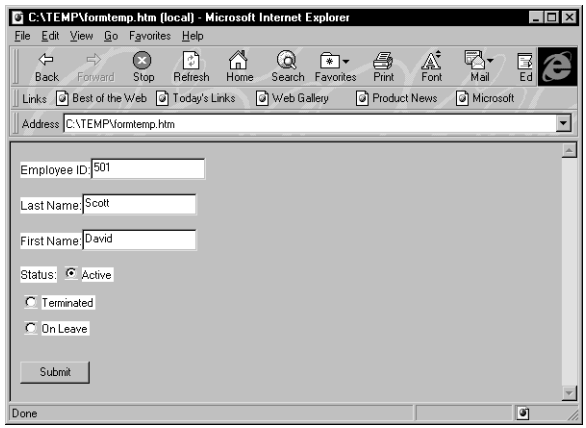
The GenerateHTMLForm method creates HTML form syntax for DataWindow objects. You can create an HTML form that displays a specified number of columns for a specified number of rows. Note the following:

- 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

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.

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 62.

- *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.

See Using DWSyntax for more information.

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 Users 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.

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.

In PowerBuilder, you can:

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

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.

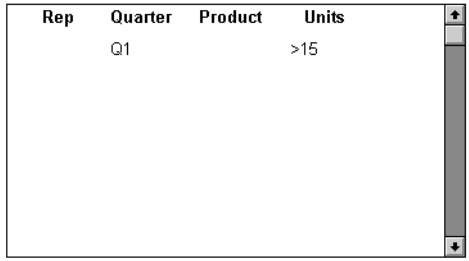
The following three figures show what happens when query mode is used.

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

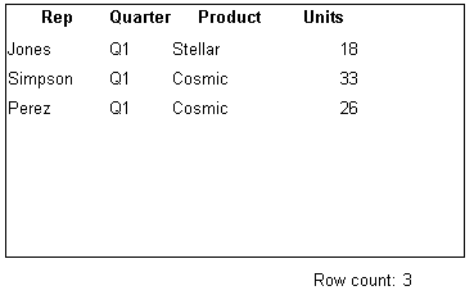
Rep	Quarter	Product	Units
Simpson	Q1	Stellar	12
Jones	Q1	Stellar	18
Perez	Q1	Stellar	15
Simpson	Q1	Cosmic	33
Jones	Q1	Cosmic	5
Perez	Q1	Cosmic	26
Simpson	Q1	Galactic	6

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:



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:



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 Users 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.

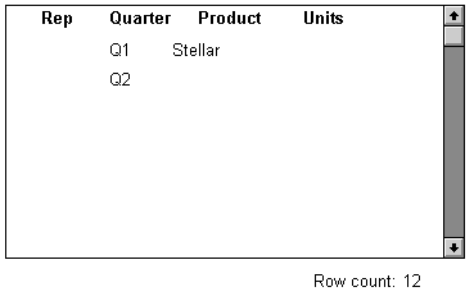
How the criteria affect the SELECT statement

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

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*.

Using DWSyntax

The DWSyntax tool, available on the Tool tab in the New dialog box, makes it easy to specify dot notation, Describe, Modify, and SyntaxFromSQL statements.

To access DWSyntax, select File>New and select the Tool tab. Select the type of statement you want to create from the Syntax menu:

- Describe

Select an object type from the Object dropdown listbox. In the Attributes listbox, select the property you want to describe. The bottom of the window displays Describe and dot notation statements.

- Modify
  - Attributes

    Select an object type and the property you want to modify. The bottom of the window displays Modify and dot notation statements.
  - Create

    Select the object type that you want to create. The bottom of the window displays a Modify statement.
  - Destroy

    Select the object type that you want to destroy. The bottom of the window displays a Modify statement.
- SyntaxFromSQL

  On each tab, select the properties you want to include in the arguments for the SyntaxFromSQL function. Notice that you can select multiple tabs and multiple properties per object for SyntaxFromSQL. When you have finished selecting properties, click Build Syntax to display the SyntaxFromSQL function at the bottom of the window.
- Tips on the syntax generated by DWSyntax

Describe

Reports the values of properties of a DataWindow object and objects within the DataWindow object. Each column and graphic object in the DataWindow has a set of properties. You specify one or more properties as a string and Describe returns the values of the properties.

Modify

Modifies a DataWindow object by applying specifications, specified as a list of instructions, that change the DataWindow object's definition. You can change appearance, behavior, and database information for the DataWindow object by changing the values of properties. You can add and remove objects from the DataWindow object by providing specifications for the objects.

Create

Creates a DataWindow object using DataWindow source code and puts that object in the specified DataWindow control. This “dynamic" DataWindow object does not become a permanent part of the application source library.

Destroy

Deletes a DataWindow object. This dynamic DataWindow object change does not become a permanent part of the application source library.

SyntaxFromSQL

Generates DataWindow source code based on a SQL SELECT statement and Style. A full presentation string has the format:

```
"Style(Type= value property=value ...)
DataWindow(property = value...)
    Column(property = value...)
    Group(groupby_col1 groupby_col2 ... property...)
    Text(property = value...)
    Title('titlestring')"
```

Tips on the syntax generated by DWSyntax

- Anything surrounded by <> indicates that a real value must be substituted (without surrounding <>). All other syntax is correct as is including single quotes.
- Internal to PowerBuilder, all DataWindow object properties are stored in strings. These can represent strings, numbers, or boolean (1/0, yes/no).

Where appropriate the compiler allows for the assigning of numbers or booleans and converts them to strings automatically. When these same property values are read they are returned as a string for the Describe syntax and as an Any variable for dot notation syntax.

Examples

The DataWindow readonly property is stored as 'yes' or 'no'.

Each of the following syntax statements sets the property to 'yes'.

```
dw_1.Modify("DataWindow.ReadOnly=Yes")
dw_1.Modify("DataWindow.ReadOnly=True")
dw_1.Object.DataWindow.ReadOnly = 'Yes'
dw_1.Object.DataWindow.ReadOnly = True
```

The result of `dw_1.Describe("DataWindow.ReadOnly")` is a string containing either 'yes' or 'no'.

The result of `dw_1.Object.DataWindow.ReadOnly` is an Any containing either 'yes' or 'no'.

The Column.Border property is stored as '0' through '6'.

Each of the following syntax statements sets the property to '5'.

```
dw_1.Modify("Column.Border = 5 ")
dw_1.Modify("Column.Border = '5' ")
dw_1.Object.Column.Border =  5
dw_1.Object.Column.Border = '5'
```

The result of `dw_1.Describe("Column.Border")` is always a string.

The result of `dw_1.Object.Column.Border` is an Any always containing a string.

CHAPTER 4

Using DataStore Objects

About this chapter

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

Contents

Before you begin

This chapter assumes you know how to build DataWindow objects in the DataWindow painter, as described in the *PowerBuilder Users 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.

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 Users 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 17.

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 80.
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.

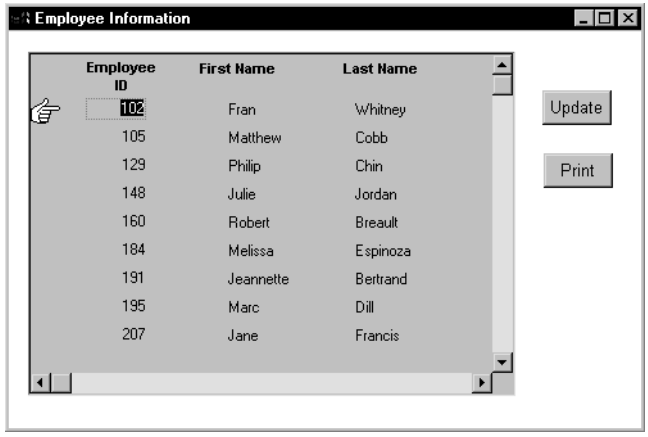
In most cases you do not need to turn off sharing, because the sharing of data is turned off automatically when a window is closed and any DataWindow controls (or DataStores) associated with the window are destroyed.

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

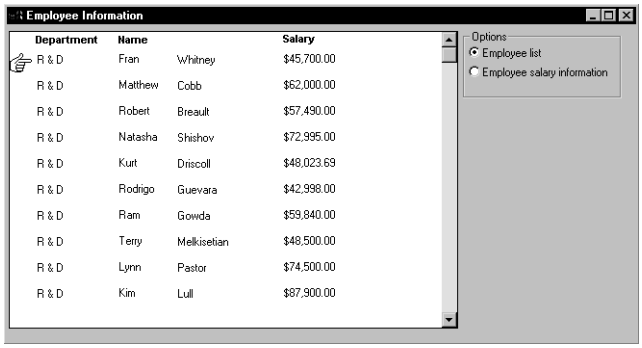
When the window closes, the DataStore gets destroyed.

This code is for the window's Close event:

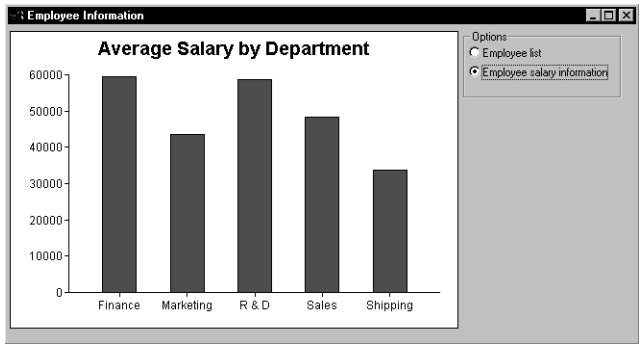
```
destroy ids_datastore
```

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

Using graphs

Supported environment

PowerBuilder 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.

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 Users 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. For 3D graphs, this includes the Render 3D property that uses transparency rather than overlays to enhance a graph's appearence and give it a more sophisticated look.
- **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

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'")
```

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 Users 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
GetData	The value of a data point, given its series and position (superseded by GetDataValue, which is more flexible)
GetDataLabelling	The display setting for the data label at a given data point in a DirectX 3D graph
GetDataPieExplode	The percentage at which a pie slice is exploded
GetDataStyle	The color, fill pattern, or other visual property of a specified data point
GetDataTransparency	The transparency percentage of a data point in a DirectX 3D graph
GetDataValue	The value of a data point, given its series and position
GetSeriesLabelling	The display setting for the series label for a given series in a DirectX 3D graph
GetSeriesStyle	The color, fill pattern, or other visual property of a specified series
GetSeriesTransparency	The transparency percentage of a series in a DirectX 3D graph
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

Saving graph data

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

Table 5-2: 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

Modifying colors, fill patterns, and other data

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

Table 5-3: PowerBuilder methods for modifying the appearance of data

Method	Action
ResetDataColors	Resets the color for a specific data point
SetDataLabelling	Specifies the display setting for a data label in a DirectX 3D graph
SetDataStyle	Sets the color, fill pattern, or other visual property for a specific data point
SetDataTransparency	Sets the transparency percentage for a data point in a DirectX 3D graph
SetSeriesLabelling	Specifies the display setting for a series label in a DirectX 3D graph
SetSeriesStyle	Sets the color, fill pattern, or other visual property for a series
SetSeriesTransparency	Sets the transparency percentage of a series in a DirectX 3D graph

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)

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.

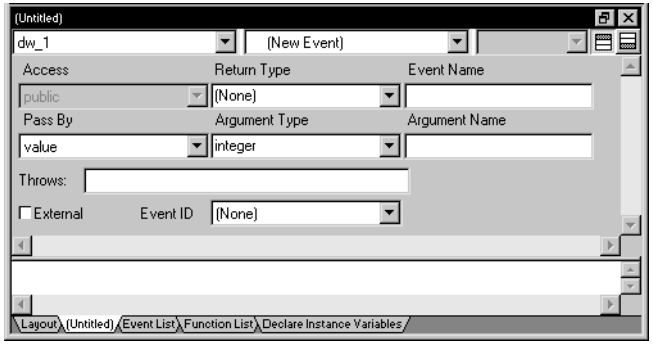
Next you create a user event for the DataWindow control that is triggered whenever a graph in the control is created or changed.

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.

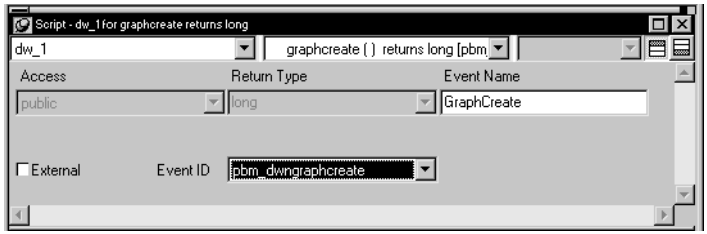
If the second drop-down list also changes to display an existing DataWindow event prototype, scroll to the top of the list to select New Event or select Insert>Event once again from the menu bar.



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)
```

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 Users 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

You should call ObjectAtPointer in the first statement of the event's code.

Using ObjectAtPointer

ObjectAtPointer works differently in PowerBuilder.

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

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
```

Index

E

F

G

H

I

L

M

N

P

Q

R

S

T

U

W