SYBASE®

User's Guide

PowerBuilder®

10.5

Contents

PART 3   **CODING FUNDAMENTALS**

CHAPTER 7   **Writing Scripts**

CHAPTER 8   **Working with User-Defined Functions**

About This Book

Audience

This book is for anyone who builds applications with PowerBuilder®. It assumes that:

- You are familiar with user interface guidelines. If not, consult a book that covers user interface conventions.

- You have a basic familiarity with SQL. If not, consult a book that describes SQL statements.

How to use this book

This book describes the PowerBuilder development environment. It shows you how to use PowerBuilder user interface tools to build the objects you need, including windows, menus, DataWindow®objects, and user-defined objects, to create client/server and multitier applications.

Related documents

Application Techniques presents information about programming techniques and building multitier applications.

The *DataWindow Programmer's Guide* explains how to use DataWindow objects in different environments and presents programming techniques related to DataWindows.

Working with Web and JSP Targets describes the tools provided with PowerBuilder for creating and deploying Web sites and JavaServer Pages.

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

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

- The SyBooks CD contains product manuals and is included with your software. The Eclipse-based SyBooks browser allows you to access the manuals in an easy-to-use, HTML-based format.

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

  Refer to the *SyBooks Installation Guide* on the Getting Started CD, or the *README.txt* file on the SyBooks CD for instructions on installing and starting SyBooks.

- The Sybase Product Manuals Web site is an online version of the SyBooks CD that you can access using a standard Web browser. In addition to product manuals, you will find links to EBFs/Maintenance, Technical Documents, Case Management, Solved Cases, newsgroups, and the Sybase Developer Network.

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

Sybase certifications on the Web

Technical documentation at the Sybase Web site is updated frequently.

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

1   Point your Web browser to Technical Documents at http://www.sybase.com/support/techdocs/.

2   Click Certification Report.

3   In the Certification Report filter select a product, platform, and timeframe and then click Go.

4   Click a Certification Report title to display the report.

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

1   Point your Web browser to Availability and Certification Reports at http://certification.sybase.com/.

2   Either select the product family and product under Search by Base Product; or select the platform and product under Search by Platform.

3   Select Search to display the availability and certification report for the selection.

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

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

1 Point your Web browser to Technical Documents at http://www.sybase.com/support/techdocs/.

2 Click MySybase and create a MySybase profile.

Sybase EBFs and software maintenance

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

1 Point your Web browser to the Sybase Support Page at http://www.sybase.com/support.

2 Select EBFs/Maintenance. If prompted, enter your MySybase user name and password.

3 Select a product.

4 Specify a time frame and click Go. A list of EBF/Maintenance releases is displayed.

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

5 Click the Info icon to display the EBF/Maintenance report, or click the product description to download the software.

Conventions

The formatting conventions used in this manual are:

Formatting example	Indicates
Retrieve and Update	When used in descriptive text, this font indicates:
	• Command, function, and method names
	• Keywords such as true, false, and null
	• Datatypes such as integer and char
	• Database column names such as emp_id and f_name
	• User-defined objects such as dw_emp or w_main

Formatting example	Indicates
variable or *file name*	When used in descriptive text and syntax descriptions, oblique font indicates: • Variables, such as *myCounter* • Parts of input text that must be substituted, such as *pblname*.pbd • File and path names
File>Save	Menu names and menu items are displayed in plain text. The greater than symbol (>) shows you how to navigate menu selections. For example, File>Save indicates "select Save from the File menu."
dw_1.Update()	Monospace font indicates: • Information that you enter in a dialog box or on a command line • Sample script fragments • Sample output fragments

If you need help

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

P A R T   1

The PowerBuilder Environment

This part describes the basics of using PowerBuilder: understanding and customizing the development environment, creating workspaces and targets, and using source control. The information in this part applies whether you are building PowerScript® or Web targets.

C H A P T E R   1   **Working with PowerBuilder**

About this chapter

This chapter describes the basics of working in the PowerBuilder development environment.

Contents

Before you begin

If you are new to PowerBuilder, doing the tutorial in *Getting Started* will help you become familiar with the development environment. The tutorial guides you through the process of building a PowerBuilder application.

About PowerBuilder

PowerBuilder is an object-centric graphical application development environment. Using PowerBuilder, you can easily develop many types of applications and components. PowerBuilder provides all the tools you need to build enterprise systems, such as order entry, accounting, and manufacturing systems.

Two-tier and multitier applications	PowerBuilder applications can be traditional graphical client/server two-tier applications that access server databases. A traditional client/server application is a collection of windows that contain controls that users can interact with. You can use standard controls—such as buttons, check boxes, drop-down lists, and edit controls—as well as special PowerBuilder controls that make your applications easy to develop and easy to use.
	You can also build multitier applications with PowerBuilder. A multitier application usually has a client application that requests services from a server application or component. For example, your client application could request services from a PowerBuilder component on an EAServer or COM+ server. The server component often requests services from a server database and/or other server components.
Web applications	PowerBuilder applications can also be Web based. You can create a new Web-based application for the Internet or Intranet, or adapt or extend an existing PowerBuilder application for the Web.

Concepts and terms

This section discusses some basic concepts and terms you need to be familiar with before you start using PowerBuilder to develop applications and components.

Workspaces and targets

In PowerBuilder, you work with one or more targets in a workspace. You can add as many targets to the workspace as you want, open and edit objects in multiple targets, and build and deploy multiple targets at once.

A PowerBuilder target can be one of two types:

* **PowerScript target** A client/server or multitier executable application or a server component. Most of this book is concerned with building PowerScript applications. See Chapter 5, "Working with PowerScript Targets."

- **Web target**   A Web target that can contain all the elements you need to build a Web site application—HTML files, scripts, images, downloaded components—or a JavaServer Pages (JSP) application. A Web target also contains settings for build options, database connections, and deployment. Web targets are described in detail in a separate book, *Working with Web and JSP Targets.*

For more information about creating a workspace and targets, see "Creating and opening workspaces" on page 17 and "Creating a target" on page 20.

PowerBuilder editions

The ability to create Web targets is available in the Enterprise edition of PowerBuilder. If you have the Professional or Desktop edition, or if you chose not to install Web target functionality, Web target options are not available in the user interface.

Objects

Your application is a collection of objects. For PowerScript targets, PowerBuilder provides many types of objects, including graphical objects such as windows, menus, and buttons, and nonvisual objects such as datastore, exception, and timing objects. In a Web target, the objects you work with include HTML pages, images, and style sheets.

As you work in your application, you create new objects and open existing objects to continue work on their development.

For more information about creating, opening, and editing objects, see "Working with objects" on page 131.

DataWindow objects

The applications you build are often centered around your organization's data. With PowerBuilder you can define DataWindow objects to retrieve, display, and manipulate data. For more information about DataWindow objects, see Chapter 18, "Defining DataWindow Objects."

PowerBuilder libraries

As you work in a PowerScript target, the objects you create are stored in one or more libraries (PBL files) associated with the application. When you run your application, PowerBuilder retrieves the objects from the library.

PowerBuilder provides a Library painter for managing your libraries. For information about creating a new library and working with libraries in the Library painter, see Chapter 6, "Working with Libraries."

Painters and editors

Some of the editors you use to edit objects are called painters. For example, you build a window in the Window painter. There you define the properties of the window, add controls such as buttons and labels, and code the window and its controls to work as your application requires.

PowerBuilder provides painters for windows, menus, DataWindow objects, visual and nonvisual user-defined objects, functions, structures, databases, data pipelines, and the application itself. For each of these object types, there is also a Source editor in which you can modify code directly. See "Working in painters" on page 114 and "Using the Source editor" on page 137.

There is also a file editor you can use to edit any file without leaving the development environment. See "Using the file editor" on page 32.

For Web targets, there are specialized editors for HTML and JSP files, frame sets, and style sheets. For more information, see *Working with Web and JSP Targets*.

Events and scripts

PowerScript applications are event-driven: users control the flow of the application by the actions they take. When a user clicks a button, chooses an item from a menu, or enters data into a text box, an event is triggered. You write scripts that specify the processing that should happen when the event is triggered.

For example, buttons have a Clicked event. You write a script for a button's Clicked event that specifies what happens when the user clicks the button. Similarly, edit controls have a Modified event, which is triggered each time the user changes a value in the control.

You write scripts using PowerScript, the PowerBuilder language, in a Script view in the painter for the object you are working on. Scripts consist of PowerScript functions, expressions, and statements that perform processing in response to an event. The script for a button's Clicked event might retrieve and display information from the database; the script for an edit control's Modified event might evaluate the data and perform processing based on the data.

Scripts can also trigger events. For example, the script for a button's Clicked event might open another window, which triggers the Open event in that window.

Functions

PowerScript provides a rich assortment of built-in functions you use to act upon the objects and controls in your application. There are functions to open a window, close a window, enable a button, retrieve data, update a database, and so on.

You can also build your own functions to define processing unique to your application.

Properties

All the objects and controls in a PowerScript target have properties, many of which you set as you develop your application. For example, you specify a label for a button by setting its text property. You can set these properties in painters or set them and modify them dynamically in scripts.

Source control

If you are working with other developers on a large application, you can make sure you are working with the latest version of a component or object by synchronizing the copy of the object you are working on with the last version of the object checked into a source control system. PowerBuilder provides a basic check in/check out utility as well as a standard application programming interface to more sophisticated source control systems. For more information, see Chapter 3, "Using Source Control."

PowerBuilder extensions

You can use PowerBuilder extension objects in a PowerScript target in the same way as you would built-in PowerBuilder objects, with one difference—you must add the PowerBuilder dynamic library that the object requires to the library list for the target. Some extensions are provided with PowerBuilder, but you can also obtain them from third parties or build your own.

For more information about the extensions provided with PowerBuilder, see the *PowerBuilder Extension Reference*. For how to build your own extensions, see the *PowerBuilder Native Interface Programmer's Guide and Reference*.

The PowerBuilder environment

When you start PowerBuilder for the first time, the Welcome to PowerBuilder dialog box lets you create a new workspace with or without targets:
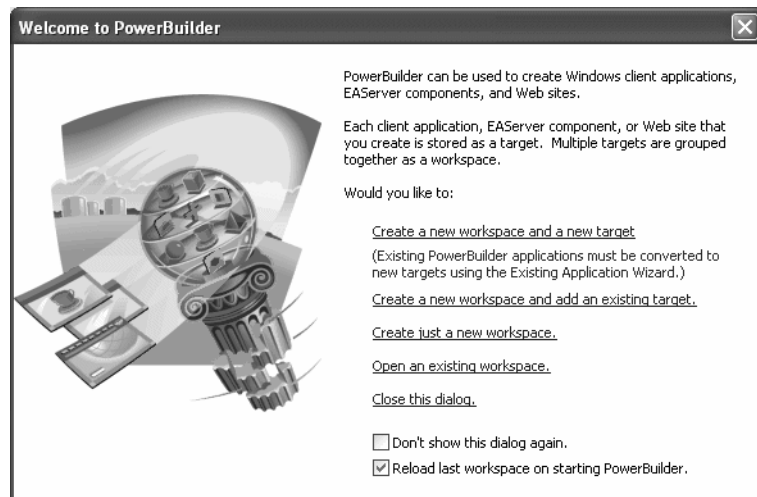
When PowerBuilder starts, it opens in a window that contains a menu bar and the PowerBar at the top and the System Tree and Clip window on the left. The remaining area will display the painters and editors you open when you start working with objects.
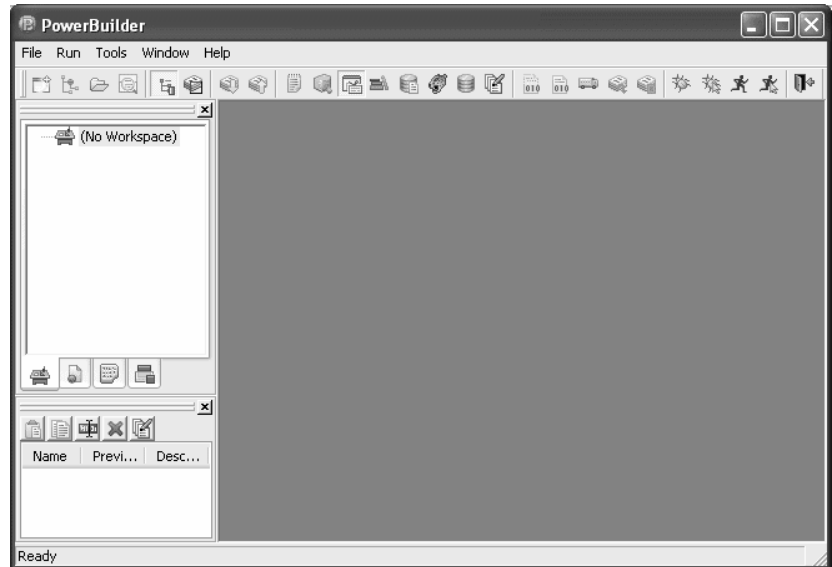
The System Tree

The System Tree provides an active resource of programming information you use while developing targets. It lets you not only get information, but also drag objects into painter views (such as the Script view, Layout view, and HTML editor) for immediate use.

This tab	Displays this content
Workspace	Workspace contents—including targets, folders, files, and libraries
Page	The page components of the active page open in the HTML editor
Language	HTML tags, client and server-side object models (objects, properties, methods, and events), and JavaScript, and VBScript syntax elements
Components	ActiveX controls, Plugins, Java Applets, JavaBeans, EAServer hosts, and JSP custom tag libraries available to your target

The System Tree displays by default when you start PowerBuilder for the first time. You can hide or display the System Tree using the System Tree button on the PowerBar or by selecting Window>System Tree.

Using the Workspace tab page

The Workspace tab page in the System Tree displays the current workspace and all its targets. PowerScript targets display the library list for the target and all the objects in each PBL. Web targets display the contents of the root folder of the Web target. The Workspace tab page in the System Tree works like a tree view in the Library painter, but you can keep it open all the time to serve as the control center of the development environment.

As in the Library painter, you can set the root of the Workspace page to your computer's root directory, the current selection, or any directory or library, as well as to the current workspace.

Working with targets

To see the pop-up menu that lets you perform operations on a target such as search, build, and migrate, you must set the root of the System Tree or the view in the Library painter to the current workspace.

The following illustration shows a workspace with two PowerScript targets. The first PowerScript target, orders, has a second library in its library search path.
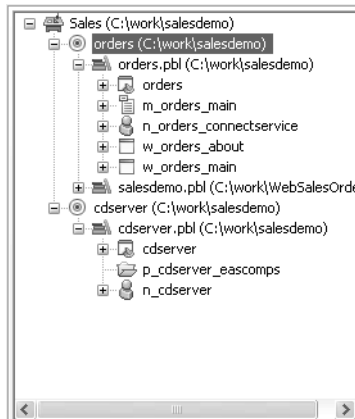
You can use the Workspace page as the hub of your PowerBuilder session. Pop-up menus let you build and deploy targets and open and edit any object. The following table lists the actions you can take on each item that displays on the Workspace page. You can also set properties for each item, choose which object types display in the tree view, change the root of the Workspace page, and reset the root to the current workspace.

Table 1-1: Action items for objects in the System Tree

Item	Menu action items
Workspace	New (opens New dialog box), Add Target, Open Workspace, Incremental Build, Full Build, Deploy, Debug, Run, Close, Show, Properties.
PowerScript target	New, Search, Incremental Build, Full Build, Migrate, Deploy, Debug, Run, Remove Target, Show, Properties.
Web or JSP target	New, New Folder, Import Files, Import Folder, Incremental Build, Full Build, Migrate Web, Deploy, Deploy to EP, Run, Remove Target, Show, Properties.
PBL	Delete, Remove Library, Search, Optimize, Print Directory, Build Runtime Library, Import, Import PB Extension, Show, Properties.
PBD	Remove Library, Show
PowerBuilder object	Edit, Edit Source, Inherit, Run/Preview, Copy, Move, Delete, Export, Regenerate, Search, Print, Properties.
	Edit Source is not available for project and proxy objects. Inherit and Run/Preview are available only for some object types. Source control items are available only if source control information is associated with the target.
Functions and events	Properties.
	The Properties dialog box shows the prototype of the function or event and its "signature." The signature is a string that represents the argument types, return types, and passing style. You use this string when you write a PBNI extension that calls the function or event. For more information, see the *PBNI Programmer's Guide and Reference*.
Web target object	Edit, Rename, Delete, Build, Properties.

Using the Page tab page

The Page tab page in the System Tree displays the Microsoft Internet Explorer object model and hierarchy for the page currently displayed in the HTML editor. This page is used only with Web targets.

The Page tab lists properties, methods, and events for the following types of objects on your page:

•   Predefined HTML and JavaScript objects that are part of the IE object model

•   User-defined objects for which you have set a name or ID attribute, such as forms and scripts

If you are developing a Web target
For more information on using the System tree when you are developing a Web target, see *Working with Web and JSP Targets*. The Page and Language tabs are relevant *only* when you are developing a Web target.

Using the Language tab page

The Language tab page in the System Tree lists the language elements available to Web targets:

•   The HTML elements and attributes defined in HTML Version 3.2, as well as extensions to HTML defined by Microsoft and Netscape, organized alphabetically and by category

•   Syntax and object models for the client- and server-side scripting languages used by Web and JSP targets
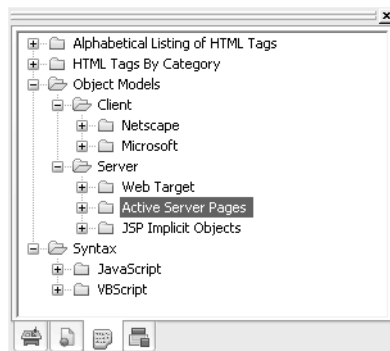
Using the Components tab page

The Components tab page in the System Tree displays the ActiveX controls, plugins, Java applets, and JavaBeans installed on your system as well as the EAServer components and JSP custom tag libraries accessible from your system.
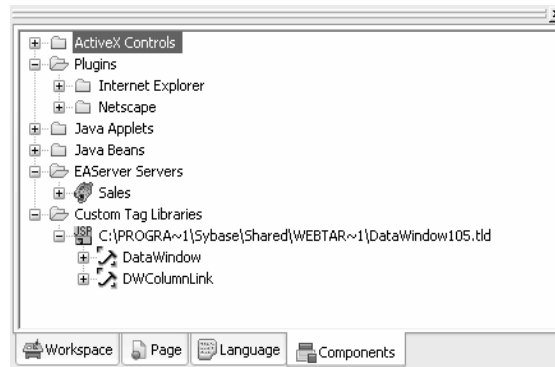
The PowerBar

Like the System Tree, the PowerBar provides a main control point for building PowerBuilder applications. From the PowerBar you can create new objects and applications, open existing objects, and debug and run the current application.

While you are getting used to using PowerBuilder, you can display a label on each button in a toolbar to remind you of its purpose. To do so, right-click a toolbar button and select Show Text from the pop-up menu.

Table 1-2 lists the buttons from left to right on the PowerBar.

Table 1-2: PowerBar buttons and their uses

This PowerBar button	Lets you do this
New	Create new objects.
Inherit	Create new windows, user objects, and menus by inheriting from an existing object.
Open	Open existing objects.
Run/Preview	Run windows or preview DataWindows.

This PowerBar button	Lets you do this
System Tree	Work in the System Tree window, which can serve as the hub of your development session. For more information see "The System Tree" on page 9.
Output Window	Examine the output of a variety of operations (migration, builds, deployment, project execution, object saves, and searches). See "The Output window" on page 16.
Next Error, Previous Error	Navigate through the Output window.
To-Do List	Keep track of development tasks you need to do for the current application and use links to get you quickly to the place where you complete the tasks.
Browser	View information about system objects and objects in your application, such as their properties, events, functions, and global variables, and copy, export, or print the information.
Clip Window	Store objects or code you use frequently. You can drag or copy items to the Clip window to be saved and then drag or copy these items to the appropriate painter view when you want to use them. See "The Clip window" on page 15.
Library	Manage your libraries using the Library painter.
DB Profile	Define and use named sets of parameters to connect to a particular database.
EAServer Profile	Define the connection parameters for a particular server. You can then use this predefined profile whenever you need to connect to EAServer.
Database	Maintain databases and database tables, control user access to databases, and manipulate data in databases using the Database painter.
Edit	Edit text files (such as source, resource, and initialization files) in the file editor.
Incremental Build Workspace	Update all the targets and objects in the workspace that have changed since the last build.
Full Build Workspace	Update all the targets and objects in the workspace.
Deploy Workspace	Deploy all the targets in the workspace.
Skip, Stop	Interrupt a build, deploy, or search operation. When a series of operations is in progress, such as a full deploy of the workspace, the Skip button lets you jump to the next operation. The Stop button cancels all operations.

This PowerBar button	Lets you do this
Debug	Debug the last target you ran or debugged. You can set breakpoints and watch expressions, step through your code, examine and change variables during execution, and view the call stack and objects in memory.
Select & Debug	Select a target and open the Debugger.
Run	Run the last target you ran or debugged just as your users would run it.
Select & Run	Select a target and run it.
Exit	Close PowerBuilder.

Customizing the PowerBar

You can customize the PowerBar. For example, you can choose whether to move the PowerBar around, add buttons for operations you perform frequently, and display text in the buttons.

For more information, see "Using toolbars" on page 46.

About PowerTips

In the PowerBar, when you leave the mouse pointer over a button for a second or two, PowerBuilder displays a brief description of the button, called a PowerTip. PowerTips display in PowerBuilder wherever there are toolbar buttons.

The Clip window

You can store code fragments you use frequently in the Clip window. You copy text to the Clip window to save it and then drag or copy this text to the appropriate Script view or editor when you want to use it.

Using the Clip window

The Clip window displays a list of named clips, a preview of the information contained in the clip, and a description. It provides buttons to move Clip window contents to the clipboard, copy clipboard contents to the Clip window, rename a clip, delete a clip, and modify the clip's description. Clips you save in one workspace are available in all your workspaces; you might want to use a naming convention that reflects this.

For example, you might use standard error-checking code when you use the ConnectToServer function to connect to EAServer. To copy it to the clipboard, highlight the code in a Script view and select Copy from the pop-up menu. In the Clip window, click the Paste icon, and name the clip. The Clip Description dialog box opens so that you can enter a description. To change the description later, select the clip's name and click the Modify button.

You can drag a clip from the Clip window to any script in which you want to connect to EAServer. You can also use the Copy icon to copy the clip to the clipboard.

You can hide or display the Clip window using the Clip Window button on the PowerBar or by selecting Window>Clip.
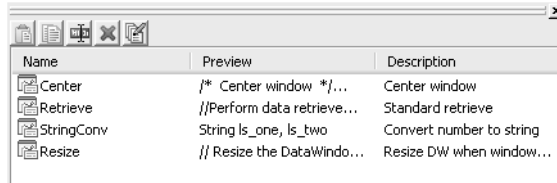
The Output window

The output of a variety of operations (migration, builds, deployment, project execution, object saves, and searches) displays in the Output window. You control operations in the window using the Skip, Stop, Next Error, and Previous Error buttons or menu options.

You can hide or display the Output window by using the Output button on the PowerBar or selecting Window>Output.

Using the Output window

When appropriate, lines in the Output window provide links that invoke the correct painter when you double-click on that line. The pop-up menu also provides the options Edit and Edit Source to open an object in a painter or the Source editor.
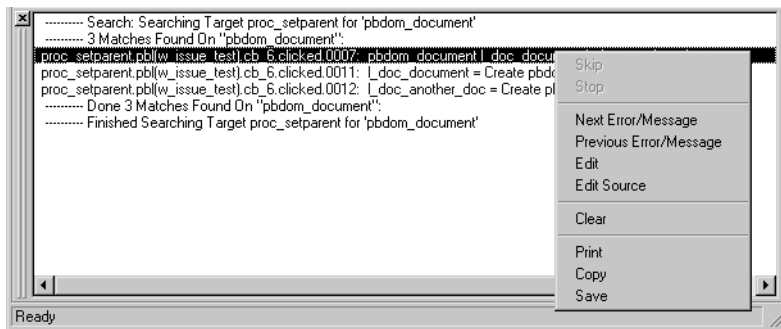
Creating and opening workspaces

Before you can begin any development in PowerBuilder, you need to create or open a workspace.

Creating a workspace

❖ **To create a new workspace:**

1   Do one of the following:

   •   Click the New button in the PowerBar.

   •   Select File>New from the menu bar.

   •   In the Workspace tab of the System Tree, right-click the workspace name and select New from the pop-up menu.

   The New dialog box opens.

2   On the Workspace tab, select Workspace.

   The New Workspace dialog box displays.

3   Enter a name for the workspace (*.pbw*) you want to create and click Save.

   The workspace is created and the name of the new workspace displays in the PowerBuilder title bar.

Opening a workspace

The next time you start PowerBuilder, it opens without opening a workspace. You can change this behavior by modifying options on the Workspaces tab of the System Options dialog box or on the Welcome to PowerBuilder screen. For example, you can have PowerBuilder open not only the workspace you used most recently, but also the objects and scripts you worked on last. See "Starting PowerBuilder with an open workspace" on page 37.

When PowerBuilder opens with an open workspace, it displays the name of the current workspace in the title bar. The current workspace is also displayed in the Workspace tab page in the System Tree. Although you can create multiple workspaces, you can have only one workspace open at a time. You can change workspaces at any time.

> ❖ **To change workspaces:**
>
> 1 Do one of the following:
>
> • Select File>Open Workspace from the menu bar.
>
> • In the Workspace tab of the System Tree, right-click on the workspace name and select Open Workspace from the pop-up menu.
>
> The Open Workspace dialog box displays.
>
> 2 From the list, select the workspace you want to open.
>
> The workspace is changed and the name of the new workspace displays in the PowerBuilder title bar.
>
> ❖ **To change the workspace to a recent workspace:**
>
> • Select File>Recent Workspaces from the menu bar and select the workspace.
>
> The workspace list includes the eight most recently accessed workspaces. You can include up to 36 workspaces on the list by selecting Tools>System Options and modifying the number of items.

Using wizards

After you have created a workspace, you can add new or existing targets to it. The first step in building a new PowerBuilder target is to use a Target wizard to create the new target and name it.

About wizards

Wizards simplify the initial creation of applications and components. Using your specifications, wizards can create multiple objects and in some cases automatically generate complex code that you can modify as needed. The first page in most wizards explains what the wizard builds. If you need help with the information you need to give the wizard, click the Help [?] button in the upper right corner of the window and then click the field you need help with, or click the field and press F1.

You start wizards from the New dialog box, but not all the icons in the New dialog box represent wizards. On the Project tab page, there are two versions of some icons: one that starts a wizard, and one that takes you straight to the Project painter.

Many wizards generate To-Do List entries to guide you through the rest of the development of the application, object, or component. See "Using the To-Do List" on page 30.

Related wizard types

Some of the wizards on the Target, PB Object, and Project tabs are related. For example, if you want to develop and deploy EAServer components, you can use the following:

- **EAServer Component Target wizard**   To create a new target, a new custom class user object, and the project object needed to generate the EAServer component and deploy the component to EAServer

- **EAServer Component Object wizard**   To create a new custom class user object in an existing target and create the project object

- **EAServer Component Project wizard**   To create a project object that will generate an EAServer component from one or more existing custom class user objects

The EAServer Component Target and Object wizards can also launch the Connection Object wizard, which builds a standard nonvisual user object, inherited from the Connection object, with properties preset for connecting to EAServer.

Creating a target

When you create a target, you are prompted for the name and location of a Target (*.pbt*) file and one or more other objects. Target files are text files that contain information about the target.

❖ **To create a new target:**

1 Do one of the following:

- Click the New button in the PowerBar.

- Select File>New from the menu bar.

- In the Workspace tab of the System Tree, highlight the workspace name and select New from the pop-up menu.

The New dialog box opens.
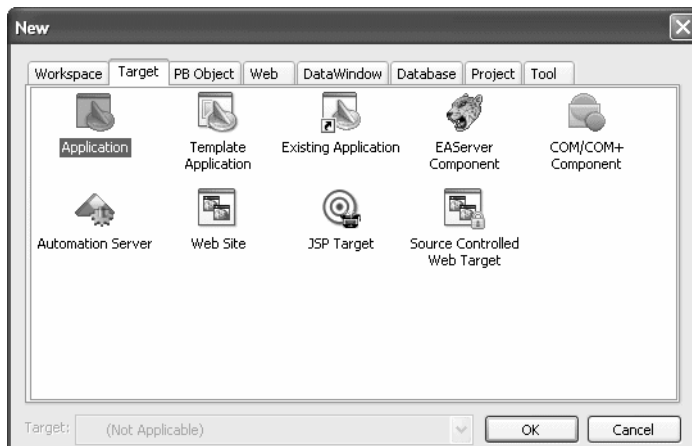
2 On the Target tab page, select one of the Target wizards.

For more information about each type of Target wizard, see the sections following these instructions.

3 Follow the instructions in the wizard, providing the information the wizard needs.

In most wizards, you can review your choices on the summary page that displays when you have finished entering information. This is a summary page from the Template Application wizard:
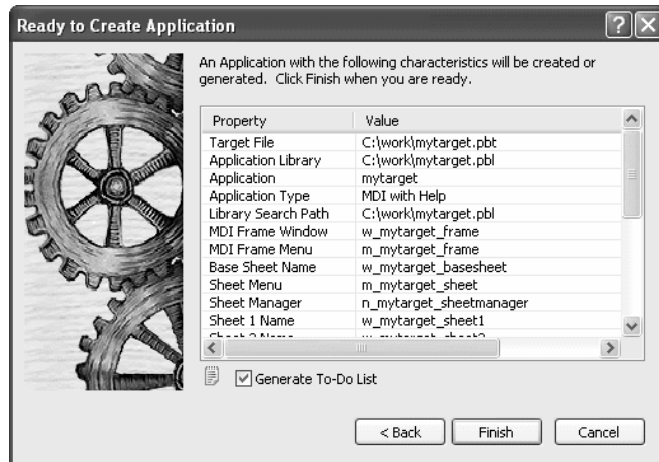
Be sure the Generate To-Do List check box is checked if you want the wizard to add items to the To-Do List to guide and facilitate your development work.

4   When you are satisfied with your choices in the wizard, click Finish.

The objects are created in the target you specified. If you specified that items were to be added to the To-Do List, you can see the items by clicking the To-Do List button in the PowerBar.

As you develop the application, you can use linked items on the To-Do list to open an object in the specific painter and view where you need to work. See "Using the To-Do List" on page 30.

Target types

This section describes each of the PowerScript application and component targets and Web targets you can build.

PowerScript application targets

There are three wizards for creating PowerScript application targets:

* Application Target wizard

* Template Application Target wizard

* Existing Application Target wizard

Application Target wizard

You use the Application Target wizard to create a new PowerScript-based Application object and the library containing it. You must create any other objects you need from scratch.

Template Application Target wizard

You use the Template Application Target wizard to create a PowerScript-based application, the library containing it, and a set of basic objects and scripts. If the application requires a connection to EAServer or a SQL database, the wizard automatically creates a Connection object.

In the Template Application wizard, you can choose one of two application types: MDI Application and SDI Application.

MDI Application   The wizard automatically generates the shell and scripts for a basic Multiple Document Interface (MDI) application that includes these objects:

    Application object
    Frame window
    Frame menu
    Base sheet window
    Sheet menu
    Sheet menu service object
    Sheet windows
    About window
    Toolbar window
    Connection service object (if database connection is needed)
    Project object (optional; can build later using a Project wizard)

You can run the MDI application immediately by clicking the Run button on the PowerBar. You can open sheets, display an About box, and select items from menus. The To-Do List can help you use the application as a starting point for continuing development of an MDI application.

SDI Application   In the Template Application wizard, you can also choose to create an SDI application. An SDI application has only one main window with a menu and an about window. If the application requires a connection to EAServer or a SQL database, the wizard automatically creates a Connection object.

For information about building MDI and SDI applications, see *Application Techniques*.

Existing Application Target wizard

You use the Existing Application Target wizard to add a target to your workspace that uses an application you built in an earlier version of PowerBuilder. After you complete the wizard, the Migrate Application dialog box opens so you can migrate the application to this version.

Before you migrate   Always make a backup copy of all the PBLs used in an application before you migrate it to a new version of PowerBuilder.

You can use the Migration Assistant to check for obsolete syntax in your application before you migrate; then you can make changes in the earlier version of PowerBuilder and avoid some migration errors. The Migration Assistant is particularly useful if you are migrating from PowerBuilder 6 or earlier. Open the Migration Assistant from the Tool tab of the New dialog box, and press F1 if you need Help in the wizard.

You should also check the release notes for the version of PowerBuilder that you are using to find out if there are any migration issues that might affect you.

For more information about migrating targets, see "Migrating targets" on page 159.

For information about building PowerScript applications, see the rest of this book and *Application Techniques*.

PowerScript component targets

EAServer Component
Target wizard

You use the EAServer Component Target wizard to create a new target containing a custom class user object (to which you later add methods and properties) with the characteristics required by an EAServer component object that you can deploy to EAServer. If you want to create an EAServer component in an existing target, use the wizard on the PB Object page.

Access to the EAServer component from a PowerBuilder client application is gained through a Proxy object. You create a Proxy object using the EAServer Proxy wizard on the Project tab page.

COM/COM+
Component Target
wizard

The COM/COM+ Component Target wizard creates a new target containing a custom class user object (to which you later add methods and properties) with the characteristics needed to be a component object that you can deploy to a Microsoft COM+ server. If you want to create a COM or COM+ component in an existing target, use the wizard on the PB Object page.

Automation Server
Target wizard

The Automation Server Target wizard creates a new custom class user object (to which you later add methods and properties) with the characteristics needed to be an automation server. You can access methods in the user object from clients built with OLE-enabled applications such as PowerBuilder, Visual Basic, and Visual C++.

For information about building components, see *Application Techniques*.

Web and JSP targets

JSP Target wizard

The JSP Target wizard creates a target in which you can build a Web application and helps you set deployment and configuration options for deploying the target to an application server.

Web Site Target
wizard

The Web Site Target wizard creates a target for organizing the files for a Web site that can be deployed to ASP or to the file system.

Source Controlled
Web Target wizard

The Source Controlled Web Target wizard opens a Web target that has been checked into a source control system.

For more information about creating Web applications, see *Working with Web and JSP Targets*.

Managing workspaces

Adding an existing target to a workspace

Although you can have only one workspace open at a time, you can add as many targets to the workspace as you want and open and edit objects in multiple targets.

❖ **To add an existing target to a workspace:**

1   Right-click on the workspace displayed in the System Tree and select Add Target from the pop-up menu.

The Add Target to Workspace dialog box displays.

2   Navigate to the directory containing the target you want to add and select the target (*.pbt*) file.

3   Click Open.

The target is added to your current workspace.

Removing a target from a workspace

When you remove a target from the workspace, the *.pbt* file is not deleted.

❖ **To remove a target from a workspace:**

•   Right-click on the target displayed in the System Tree and select Remove Target from the pop-up menu.

Specifying workspace properties

You specify workspace properties in the Properties of Workspace dialog box.

❖ **To specify workspace properties:**

1   In the Workspace tab of the System Tree, select Properties from the pop-up menu for the workspace.

2   Select the Targets, Deploy Preview, or Source Control tab page.

3   Specify the properties as described in the following sections.

Specifying target order	You can specify the targets and the order in which those targets are built or deployed on the Targets tab page. All the targets identified with the workspace are listed. Check the targets you want to include in the workspace build or deploy. Use the arrows to change a target's position in the target order list.
Previewing deployment	You can verify the targets and the order in which those targets are built or deployed on the Deploy Preview tab page. To make changes, you need to use:

- The Targets page in the Properties of Workspace dialog box to change which targets are selected for deployment

- The Deploy page on the properties dialog box for the target to specify which projects in a PowerScript target are built or which deployment configuration is used for a Web target

Specifying source control properties	You can specify which source control system, if any, is used for this workspace, as well as other source control properties. For more information, see Chapter 3, "Using Source Control."

Building workspaces

You can build and deploy workspaces while you are working in PowerBuilder, and from a command line.

In the development environment

In the development environment, you can specify how you want the targets in your workspace to be built and deployed. Then you can build individual targets or all the targets in the workspace. Table 1-3 summarizes where you set up build and deploy options, and how you start builds.

Table 1-3: Building and deploying targets and workspaces

To do this	Do this
Set deploy options for a PowerScript target	Select Properties from the pop-up menu for the target and select the Deploy tab. Check the box next to a project to build it when you select Deploy from the target's pop-up menu. Use the arrows to set the order in which projects are built.
	Set options for each project in the target in the Project painter.

To do this	Do this
Set build and deploy options for a Web target	Set up and activate deployment controllers on the Deploy page of the target's properties dialog box.
Set build and deploy options for the workspace	Select Properties from the pop-up menu for the workspace and select the order in which targets should be built. You can check which projects and deploy configurations are currently selected on the Deploy Preview page.
Build, migrate, or deploy a selected PowerScript target	Select Incremental Build, Full Build, Migrate, or Deploy from the pop-up menu for the target. Deploy builds the projects in the target in the order listed on the Deploy page of the target's properties dialog box.
Build or deploy a selected Web target	Select Incremental Build or Full Build from the pop-up menu for the target to check HTML files for broken links and populate a build directory with their data, and copy other files to the build directory. Select Deploy to deploy the target using the deployment controllers marked as active on the Deploy page of the target's properties dialog box.
Build or deploy all the targets in the workspace	Select Incremental Build, Full Build, or Deploy from the pop-up menu for the workspace, from the Run menu, or from the PowerBar.

From a command line

When you deploy or build a workspace from a command line, PowerBuilder starts, completes the build, and exits as soon as the operation is completed. To retain a log file for the session, you can send the contents of the Output window to a file. Table 1-4 shows command-line options for building and deploying targets and workspaces.

Table 1-4: Command-line options for building and deploying

Option	Description
/workspace *workspacepath*	Open the workspace *workspacepath*
/target *targetpath*	Open the target *targetpath*
/deploy	Deploy the workspace and exit
/fullbuild	Fully build the workspace and exit
/incrementalbuild	Incrementally build the workspace and exit
/output *outputpath*	Log the contents of the Output window to *outputpath*

As with other command-line options, you need only use the initial letter or letters of the option name as long as the option is uniquely identified. The deploy, fullbuild, and incrementalbuild options can be used only with the workspace option. You need to create projects and specify build and deploy options for the workspace in PowerBuilder before you start a build from the command line. Deploy builds the projects in the target in the order listed on the Deploy page of the target's properties dialog box.

Example

This example assumes that the location of the PowerBuilder executable file is in your system path. It opens the workspace called CDShop, builds and deploys the targets in the workspace according to your specifications in the workspace and target properties, records the content of the Output window in the file *D:\tmp\cdshop.out*, and exits PowerBuilder:

```
pb105 /w D:\CDShop\CDShop.pbw /d /out D:\tmp\cdshop.out
```

There are additional command-line options you can use to start PowerBuilder. See "Using command line arguments" on page 38.

Working with tools

PowerBuilder provides a variety of tools to help you with your development work. There are several ways to open tools:

• Click a button in the PowerBar for the tool you want

• Select the tool from the Tools menu

• Open the New dialog box and select the tool you want on the Tool tab page

Table 1-5 lists the tools available in the PowerBar. Some of these tools are also listed on the Tools menu.

Table 1-5: Tools available in the PowerBar

Tool	What you use the tool for
To-Do List	Keep track of development tasks you need to do for the current target and create links to get you quickly to the place where you need to complete the tasks. For information, see "Using the To-Do List" on page 30.
Browser	View information about system objects and objects in your PowerScript target, such as properties, events, functions, and global variables, and copy, export, or print the information. For information, see "Browsing the class hierarchy" on page 308.

Tool	What you use the tool for
Library painter	Manage libraries, create a new library, build dynamic libraries, and use source control.
Database profiles	Define and use named sets of parameters to connect to a particular database. For information, see *Connecting to Your Database*.
EAServer profiles	Define and use named sets of parameters to connect to a particular EAServer host. For information, see *Connecting to Your Database*.
Database painter	For information, see Chapter 16, "Managing the Database."
File Editor	Edit text files such as source, resource, and initialization files. For information, see "Using the file editor" on page 32.
Debugger	Set breakpoints and watch expressions, step through your application, examine and change variables during execution, and view the call stack and objects in memory. For information, see Chapter 32, "Debugging and Running Applications."

Table 1-6 lists the tools you can launch from the Tool tab page in the New dialog box. You can also launch the Library painter and File Editor from this dialog box.

Table 1-6: Tools available in the New dialog box

Tool	What you use the tool for
Migration Assistant	Scans PowerBuilder libraries and highlights usage of obsolete functions and events. For information, see the Migration Assistant online Help.
DataWindow Syntax	Helps construct the syntax required by Modify, Describe, and SyntaxFromSQL functions. For information, see DataWindow Syntax online Help.
Profiling Class View, Profiling Routine View, and Profiling Trace View	Use trace information to create a profile of your application. For information, see Chapter 33, "Tracing and Profiling Applications."
Web DataWindow JavaScript Generator	Generate a JavaScript file that contains DataWindow methods you want to associate with a specific DataWindow object. For information, see the *DataWindow Programmer's Guide*.

Using the To-Do List

The To-Do List displays a list of development tasks you need to do. You can create tasks for any target in the workspace or for the workspace itself. A drop-down list at the top of the To-Do List lets you choose which tasks to display. To open the To-Do List, click the To-Do List button in the PowerBar or select Tools>To-Do List from the menu bar.

To-Do List entries   The entries on the To-Do list are created:

- Automatically by most PowerBuilder wizards to guide you through the continued development of objects of different types that you will need to build the application or component specified by the wizard

- At any time by you when you are working in a painter and want a link to a task that you want to remember to complete

Some To-Do List entries created by wizards are hot-linked to get you quickly to the painter (and the specific object you need) or to a wizard. You can also create an entry yourself that links to the PowerBuilder painter where you are working so you can return to the object or script (event/function and line) you were working on when you made the entry. When you move the pointer over entries on the To-Do list, the pointer changes to a hand when it is over a linked entry.

For example, if you generate an MDI application with the Template Application wizard, one of the linked entries on the To-Do List reminds you to register new sheets with the sheet manager service, which is a nonvisual user object created by the wizard. Double-clicking this entry automatically opens the Window painter and the Script view where you register new sheets.
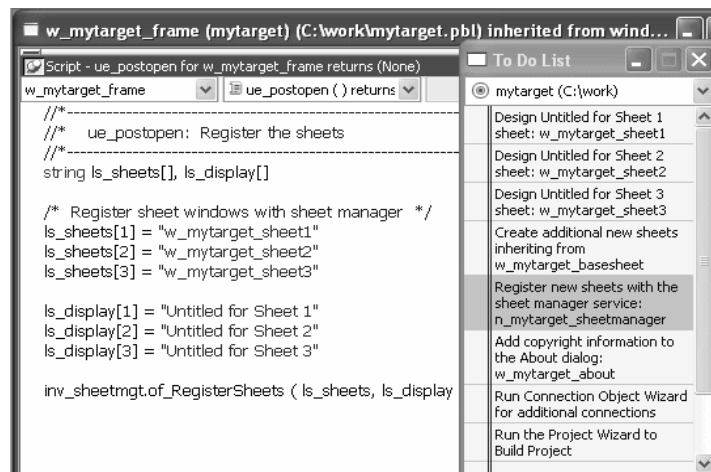
Exporting and
importing lists

You can export or import a To-Do List by selecting Export or Import from the
pop-up menu. Doing this is useful if you want to move from one computer to
another or you need to work with To-Do Lists as part of some other system
such as a project management system.

Linked entries
If you import a list from another workspace or target, or from a previous
version of PowerBuilder, linked entries will display in the list but the links will
not be active.

Working with entries
on the To-Do List

Table 1-7 tells you how to work with entries on the To-Do List.

Table 1-7: Using the To-Do List

To	Do this
See linked entries	Move the pointer over the entries. A hand displays when the entry you are over is linked.
Use a linked entry to get to a painter or wizard	Double-click the linked entry or select it and then select Go To Link from the pop-up menu.
Add an entry with no link	Select Add from the pop-up menu.
Add a linked entry to a painter that edits objects	With the painter open, select Add Linked from the pop-up menu.
Add an entry for a specific target	If the To-Do List is open, select the target from the drop-down list at the top of the To-Do List and add the entry. If the To-Do List is closed, select a target in the System Tree, open the To-Do List, and add the entry.
Add an entry for the workspace	Select Current Workspace from the drop-down list at the top of the To-Do List and add the entry.
Change the list that displays	Select a specific target or Current Workspace from the drop-down list at the top of the To-Do List. To display tasks for all targets and the workspace, select All Items.
Change an entry's position on the list	Drag the entry to the position you want.
Edit or delete an entry	Select Edit or Delete from the pop-up menu.
Delete checked entries or all entries	Select Delete Checked or Delete All from the pop-up menu.
Check or uncheck an entry	Click in the margin to the left of the entry or select an entry and then select Check/Uncheck from the pop-up menu.

To	Do this
Export a To-Do List	Select Export from the pop-up menu, name the To-Do List text file, and click Save.
Import a To-Do List	Select Import from the pop-up menu, navigate to an exported To-Do List text file, and click Open.

Using the file editor

One of the tools on the PowerBar and Tools menu is a text editor that is always available. Using the editor, you can view and modify text files (such as initialization files and tab-separated files with data) without leaving PowerBuilder. Among the features the file editor provides are find and replace, undo, importing and exporting text files, and dragging and dropping text.

Setting file editing properties

The file editor has font properties and an indentation property that you can change to make files easier to read. If you do not change any properties, files have black text on a white background and a tab stop setting of 3 for indentation. Select Design>Options from the menu bar to change the tab stop and font settings.

Editor properties apply elsewhere
When you set properties for the file editor, the settings also apply to the Function painter, the Script view, the Source editor, the Interactive SQL view in the Database painter, and the Debug window.

Dragging and dropping text

To move text, simply select it, drag it to its new location, and drop it. To copy text, press the Ctrl key while you drag and drop the text.

Using online Help

PowerBuilder has two kinds of online Help: HTML Help and Windows Help.

About HTML Help

HTML Help contains several PowerBuilder books: this *User's Guide*, *Application Techniques*, the *DataWindow Programmer's Guide*, *Connecting to Your Database*, and, for PowerBuilder Enterprise users, *Working with Web and JSP Targets*.

Context-sensitive Help and reference information for Web targets is also in HTML format in a separate HTML Help file that is launched when you select Help in a Web target. This context-sensitive and reference Help is installed only if you install the Web targets feature.

About Windows Help

Windows Help contains a list of new features in this release, context-sensitive Help, all the reference information in *Objects and Controls*, the *DataWindow Reference*, and the *PowerScript Reference*, and Help for tools such as the Migration Assistant and the Profiling and DWSyntax tools.

Some Windows Help topics provide links to related topics in the HTML Help.

Accessing Help

Table 1-8 lists the ways you can access Help.

Table 1-8: Accessing online Help

Approach	What it does
Use the Help menu on the menu bar	Displays the Help contents, the What's New in PowerBuilder Help, or Help for the current painter.
In a wizard, click the Help button [?] in the upper right corner of the window	The pointer displays with a question mark so you can get context-sensitive Help. Point and click in a field you need Help on.
In the Properties view in a painter, select Help from the pop-up menu on any tab page	Displays a Help topic from which you can get Help on the properties, events, and functions for the object or control whose properties are displaying in the Properties view.
Add a Help button to the PowerBar and use it	Displays the Help contents.
Press F1	Displays the Help contents.
Press Shift+F1 in the Script view or Function painter	Displays context-sensitive Help about the function, event, or keyword under the cursor.
Select Help from the pop-up menu in the Browser	Displays Help for the Browser or for the selected object, control, or function.
Click the Help button in a dialog box	Displays information about that dialog box.
Click the link to HTML Help icon in a Windows Help topic	Opens the HTML Help at the linked topic.

Sybooks CD and Web site

PowerBuilder books are also provided on the Sybooks CD and the Sybase Product Manuals Web site. For more information, see "Other sources of information" on page xxiii.

Building an application

This section describes the basic steps you follow when building a traditional client/server application. After completing step 1, you can work in any order. That is, you can define the objects used in your application in any order, as you need them.

❖ **To build a traditional client/server application:**

1 Create the application (using a New wizard) and specify the library list for the application.

When you use a Start wizard, you create the Application object, which is the entry point into the application. The Application object contains the name of the application and specifies the application-level scripts.

See Chapter 5, "Working with PowerScript Targets," and Part 3, "Coding Fundamentals."

2 Create windows.

Place controls in the window and build scripts that specify the processing that will occur when events are triggered.

See Chapter 11, "Working with Windows."

3 Create menus.

Menus in your windows can include a menu bar, drop-down menus, cascading menus, and pop-up menus. You define the menu items and write scripts that execute when the items are selected.

See Chapter 14, "Working with Menus and Toolbars."

4 Create user objects.

If you want to be able to reuse components that are placed in windows, define them as user objects and save them in a library. Later, when you build a window, you can simply place the user object on the window instead of having to redefine the components.

See Chapter 15, "Working with User Objects."

5   Create functions, structures, and events.

To support your scripts, you define functions to perform processing unique to your application and structures to hold related pieces of data. You can also define your own user events.

See Chapter 8, "Working with User-Defined Functions," Chapter 9, "Working with User Events," and Chapter 10, "Working with Structures."

6   Create DataWindow objects.

Use these objects to retrieve data from the database, format and validate data, analyze data through graphs and crosstabs, and update the database.

See Part 6, "Working with DataWindows."

7   Test and debug your application.

You can run your application at any time. If you discover problems, you can debug your application by setting breakpoints, stepping through your code, and looking at variable values during execution. You can also create a trace file when you run your application and use PowerBuilder's profiling tools to analyze the application's performance and logical flow.

See Chapter 32, "Debugging and Running Applications," and Chapter 33, "Tracing and Profiling Applications."

8   Prepare an executable.

When your application is complete, you prepare an executable version to distribute to your users.

See Chapter 34, "Creating Executables and Components."

Using other books

This book tells you how to use PowerBuilder painters and tools.

For programming techniques for building applications and server clients and components in PowerBuilder, see *Application Techniques*.

For programming techniques related to DataWindows, including using the Web DataWindow, see the *DataWindow Programmer's Guide*.

For information about Web targets, see *Working with Web and JSP Targets*.

CHAPTER 2 **Customizing PowerBuilder**

About this chapter   This chapter describes how you can customize the PowerBuilder development environment to suit your needs and get the most out of PowerBuilder's productivity features.

Contents

Topic	Page
Starting PowerBuilder with an open workspace	37
Changing default layouts	40
Using toolbars	46
Customizing keyboard shortcuts	54
Changing fonts	55
Defining colors	56
How the PowerBuilder environment is managed	57

Starting PowerBuilder with an open workspace

When you start PowerBuilder, you might want to resume work on an existing project. You can have PowerBuilder open the workspace that you used last, and even open the painters you had open, with the last Script view you touched open at the code you were working on.

Using options in the development environment

There are three options on the Workspaces page of the System Options dialog box that you can use to determine what displays when you start PowerBuilder.

❖ **To open the System Options dialog box:**

• Select Tools>System Options from the menu bar.

Opening just the workspace   If you want PowerBuilder to open the last workspace you used at startup, check Reopen Workspace at Startup.

Opening the workspace, painters, and scripts	If you want PowerBuilder to open the last workspace you used *and* the painters and editors you were using, check Reopen Workspace on Startup and Reload Painters When Opening Workspace. When you open PowerBuilder, any painters and editors that were open when you closed PowerBuilder are reloaded. If you edited a script before closing PowerBuilder, the Script view is scrolled to show the last line you edited.
Opening with no workspace	If you want PowerBuilder to open without loading a workspace, clear Reopen Workspace on Startup. If you want the painters and editors that were open when you last used a workspace to be reloaded when you reopen it, clear Reopen Workspace on Startup and check Reload Painters When Opening Workspace.
Displaying the Welcome dialog box	If you want to see the Welcome to PowerBuilder dialog box when you start PowerBuilder, check Show Start Dialog at Startup with no Workspace and clear Reopen Workspace at Startup. The Welcome to PowerBuilder dialog box is shown in "The PowerBuilder environment" on page 8.

Using a workspace file

Double-click a workspace file in Windows Explorer. Workspaces have a *.pbw* extension. PowerBuilder starts with the workspace open.

Using command line arguments

You can start PowerBuilder from a command line (or the Windows Run dialog box) and optionally open a workspace, target, and/or painter. These are the painters and tools you can open:

Application painter
Database painter
Data Pipeline painter
DataWindow painter
Debugger
File Editor
Function painter
Library painter
Menu painter
Query painter
Structure painter
User Object painter
Window painter

The syntax is:

> *directory*\pb105.exe {/workspace *workspacepath*} {/target *targetpath*}
> {/painter *paintername*} {/output *outputpath*}

where *directory* is the fully qualified name of the directory containing
PowerBuilder.

You can also add one or more of the following options to the command line
after /painter *paintername* to open a specific object or create a new one:

> {/library *libraryname*} {/object *objectname*} {/inherit objectname} {/new}
> {/run} {/runonly} {/argument *arguments*}

The syntax statements show the long form of option names. You need only use
the initial letter or letters of the option name as long as the option is uniquely
identified, as shown in Table 2-1.

Table 2-1: Command-line options for opening PowerBuilder

Option	Description
/W *workspacepath*	Opens the workspace *workspacepath*. The default is the most recently used workspace if you have selected the Reopen Workspace on Startup check box in the System Options dialog box. If you have not selected this check box, you must specify the /W option before specifying any other options.
/T *targetpath*	Opens the target *targetpath*.
/P *paintername*	Opens the painter *paintername*. The default is the window that displays when you begin a new PowerBuilder session.
	The painter name must uniquely identify the painter. You do not have to enter the entire name. For example, you can enter q to open the Query painter and datab to open the Database painter. If you enter the full name, omit any spaces in the name (enter UserObject and DataPipeline, for example).
	The painter name is not case sensitive. To open the file editor, you could set *paintername* to FI or fileeditor.
	Except for the /W, /T, and /L switches, other switches must follow /P *paintername* on the command line, as shown in the examples after the table.
/OU *outputpath*	Logs the contents of the Output window to *outputpath*.
/L *libraryname*	Identifies the library that contains the object you want to open.
/O *objectname*	Identifies the object, such as a DataWindow object or window, you want to open.
/I *objectname*	Identifies the object you want to inherit from.
/N	Creates a new DataWindow object.
/R	Runs the DataWindow object specified with /O and allows designing.

Option	Description
/RO	Runs the DataWindow object specified with /O but does not allow designing.
/A *arguments*	Provides arguments for the specified DataWindow object.

Examples

The following examples assume that the location of the PowerBuilder executable file is in your system path.

This example starts a PowerBuilder session by opening the Window painter in the Client PBL in the Math workspace. The output of the session is sent to a file called *math.log*. The workspace file, the PBL, and the log file are all in the current directory:

```
pb105 /w Math.pbw /l Client.pbl /p window /out math.log
```

Enter this command to start PowerBuilder and open the DataWindow object called d_emp_report in the workspace *Emp.pbw*:

```
pb105 /w D:\pbws\Emp.pbw /P dataw /O d_emp_report
```

Building from the command line
You can also build and deploy a workspace from the command line. For more information, see "Building workspaces" on page 26.

Changing default layouts

You can change the layout of the PowerBuilder main window in several ways. This section describes:

- Showing or hiding the System Tree, Output, and Clip windows and changing their locations

- Showing or hiding views in painters and changing their locations

You can also show or hide toolbars, change their locations, and add custom buttons. See "Using toolbars" on page 46.

Arranging the System Tree, Output, and Clip windows

Hiding windows

The System Tree, Output, and Clip windows can all be hidden at any time by clicking their buttons on the PowerBar.

Moving windows

You can dock the System Tree, Output, and Clip windows at the top, bottom, left, or right of the PowerBuilder main window by dragging the double bar at the top or side of the windows.

Using the full width or height of the main window

Windows docked at the top or bottom of the main window occupy the full width of the frame. You can change this default by clearing the Horizontal Dock Windows Dominate check box on the General page System Options dialog box. The following screen shows the Clip and Output windows docked at the bottom of the window. The Horizontal Dock Windows Dominate check box has been cleared so that the System Tree occupies the full height of the window:
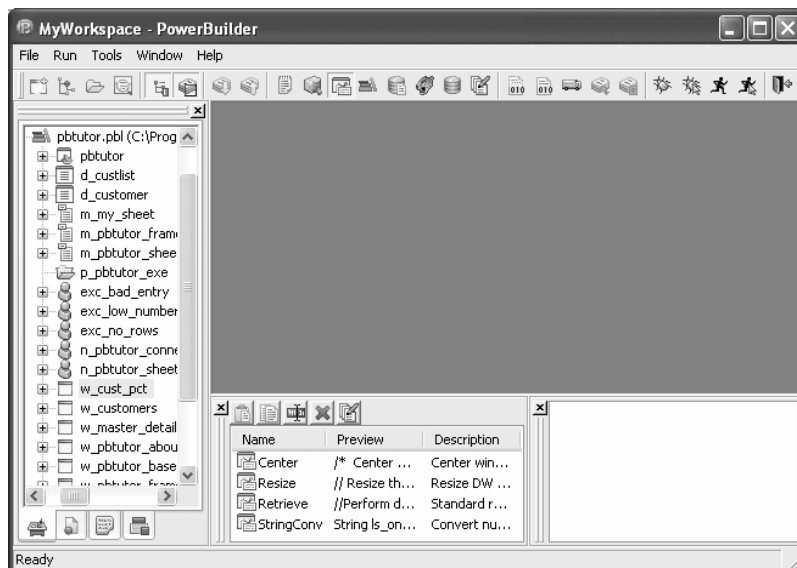
Using views in painters

Most of the PowerBuilder painters have views. Each view provides a specific way of viewing or modifying the object you are creating or a specific kind of information related to that object. Having multiple views available in a painter window means you can work on more than one task at a time. In the Window painter, for example, you can select a control in the Layout view to modify its properties, and double-click the control to edit its scripts.

Views are displayed in panes in the painter window. Some views are stacked in a single pane. At the bottom of the pane there is a tab for each view in the stack. Clicking the tab for a view pops that view to the top of the stack.

Each painter has a default layout, but you can display the views you choose in as many panes as you want to and save the layouts you like to work with. For some painters, all available views are included in the default layout; for others, only a few views are included.

Each pane has:

- A title bar you can display temporarily or permanently

- A handle in the top-left corner you can use to drag the pane to a new location

- Splitter bars between the pane and each adjacent pane

Displaying the title bar

For most views, a title bar does not permanently display at the top of a pane because it is often unnecessary, but you can display a title bar for any pane either temporarily or permanently.

❖ **To display a title bar:**

1 Place the pointer on the splitter bar at the top of the pane.

 The title bar displays.

2 To display the title bar permanently, click the pushpin at the left of the title bar or select Pinned from its pop-up menu.

 Click the pushpin again or select Pinned again on the pop-up menu to hide the title bar.

After you display a title bar either temporarily or permanently, you can use the title bar's pop-up menu.

❖ **To maximize a pane to fill the workspace:**

- Select Maximize from the title bar's pop-up menu or click the Maximize button on the title bar

❖ **To restore a pane to its original size:**

- Select Restore from the title bar's pop-up menu or click the Restore button on the title bar

Moving and resizing panes and views

You can move a pane or a view to any location in the painter window. You might find it takes a while to get used to moving panes and views around, but if you do not like a layout, you can always revert to the default layout and start again. To restore the default layout, select View>Layouts>Default.

To move a pane, select and drag the title bar of the view that is at the top of the stack. If the pane contains stacked views, *all* views in the stack move together. To move one of the views out of the stack, drag the tab for the view you want to move.

❖ **To move a pane:**

1   Place the pointer anywhere on the title bar of the view at the top of the stack, hold down the left mouse button, and start moving the pane.

A gray outline appears around the pane:

2   Drag the outline to the new location.

The outline changes size as you drag it. When the pointer is over the middle of a pane, the outline fills the pane. As you drag the pointer toward any border, the outline becomes a narrow rectangle adjacent to that border. When the pointer is over a splitter bar between two panes, rows, or columns, the outline straddles the splitter bar:
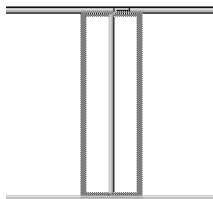
When you move the pointer to a corner
When you move the pointer to a corner, you will find that you have many places where you can drop the outline. To see your options, move the pointer around in all directions in the corner and see where the outline displays as you move it.

3   Release the mouse button to drop the outline in the new location:

To move a pane here	Drop the outline here
Between two panes	On the splitter bar between the panes
Between a border and a pane	At the side of the pane nearest the border
Into a new row	On the splitter bar between two rows or at the top or bottom of the painter window
Into a new column	On the splitter bar between two columns or at the left or right edge of the painter window
Onto a stack of panes	On the middle of the pane (if the pane was not already tabbed, tabs are created)

❖   **To move a view in a stacked pane:**

•   Place the pointer anywhere on the view's tab, hold down the left mouse button, and start moving the view.

You can now move the view as in the previous procedure. If you want to rearrange the views in a pane, you can drag the view to the left or right within the same pane.

❖   **To resize a pane:**

•   Drag the splitter bars between panes.

Floating and docking views

Panes are docked by default within a painter window, but some tasks may be easier if you float a pane. A floating pane can be moved outside the painter's window or even outside the PowerBuilder window.

When you open another painter
If you have a floating pane in a painter and then open another painter, the floating pane temporarily disappears. It reappears when the original painter is selected.

❖   **To float a view in its own pane:**

•   Select Float from the title bar's pop-up menu.

❖   **To float a view in a stacked pane:**

•   Select Float from the tab's pop-up menu.

❖ **To dock a floating view:**

• Select Dock from the title bar's pop-up menu.

Adding and removing views

You may want to add additional views to the painter window. You can open only one instance of some views, but you can open as many instances as you need of others, such as the Script view. If there are some views you rarely use, you can move them into a stacked pane or remove them. When removing a view in a stacked pane, make sure you remove the view and not the pane.

❖ **To add a new view to the painter window:**

1   Select View from the menu bar and then select the view you want to add.

    The view displays in a new pane in a new row.

2   Move the pane where you want it.

    For how to move panes, see "Moving and resizing panes and views" on page 43.

❖ **To remove a view in its own pane from the painter window:**

1   If the view's title bar is not displayed, display it by placing the pointer on the splitter bar at the top of the pane.

2   Click the Close button on the title bar.

❖ **To remove a view in a stacked pane from the painter window:**

• Select the tab for the view and select Close from its pop-up menu.

❖ **To remove a stacked pane from the painter window:**

1   If the title bar of the top view in the stack is not displayed, display it by placing the pointer on the splitter bar at the top of the pane.

2   Click the Close button on the title bar.

Saving a layout

When you have rearranged panes in the painter window, PowerBuilder saves the layout in the registry. The next time you open the painter window, your last layout displays. You can also save customized layouts so that you can switch from one to another for different kinds of activities.

❖ **To save customized layouts for a painter window:**

1 Select View>Layouts>Manage from the menu bar.

2 Click the New Layout button (second from the left at the top of the dialog box).
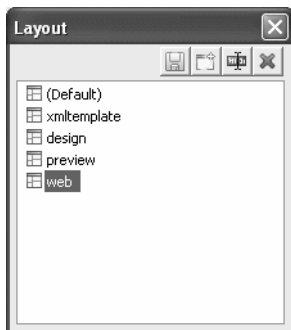
3 Type an appropriate name in the text box and click OK.

You can restore the default layout at any time by selecting Views>Layout>Default.

Using toolbars

Toolbars provide buttons for the most common tasks in PowerBuilder. You can move (dock) toolbars, customize them, and create your own.

Toolbar basics

PowerBuilder uses three toolbars: the PowerBar, PainterBar, and StyleBar:

This toolbar	Has buttons for	And (unless hidden) displays
PowerBar	Opening painters and tools	Always
PainterBar	Performing tasks in the current painter	In each painter or editor; some painters have more than one PainterBar
StyleBar	Changing properties of text, such as font and alignment	In appropriate painters

Drop-down toolbars

To reduce the size of toolbars, some toolbar buttons have a down arrow on the right that you can click to display a drop-down toolbar containing related buttons.

For example, the down arrow next to the Text button in the DataWindow painter displays the Controls drop-down toolbar, which has a button for each control you can place on a DataWindow object.
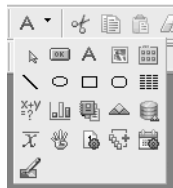
Default button replaced
The button you select from a drop-down toolbar replaces the default button on the main toolbar. For example, if you select the Picture button from the Controls drop-down toolbar, it replaces the Text button in the PainterBar.

Controlling the display of toolbars

You can control:

- Whether to display individual toolbars and where
- Whether to display text on the buttons
- Whether to display PowerTips

Choosing to display text and PowerTips affects all toolbars.

❖ **To control a toolbar using the pop-up menu:**

1 Position the pointer on a toolbar and display the pop-up menu.

2 Click the items you want.

A check mark means the item is currently selected.

❖ **To control a toolbar using the Toolbars dialog box:**

1 Select Tools>Toolbars from the menu bar.

   The Toolbars dialog box displays.

2 Click the toolbar you want to work with (the current toolbar is highlighted) and the options you want.

PowerBuilder saves your toolbar preferences in the registry and the PowerBuilder initialization file.

Moving toolbars using the mouse

You can use the mouse to move a toolbar.

❖ **To move a toolbar with the mouse:**

1 Position the pointer on the grab bar at the left of the toolbar or on any vertical line separating groups of buttons.

2 Press and hold the left mouse button.

3 Drag the toolbar and drop it where you want it.

   As you move the mouse, an outlined box shows how the toolbar will display when you drop it. You can line it up along any frame edge or float it in the middle of the frame.

Docking toolbars

When you first start PowerBuilder, all the toolbars display one above another at the top left of the workspace. When you move a toolbar, you can dock (position) it:

• At the top or bottom of the workspace, at any point from the left edge to the right edge

• At the left or right of the workspace, at any point from the top edge to the bottom edge

• To the left or right of, or above or below, another toolbar

Customizing toolbars

You can customize toolbars with PowerBuilder buttons and with buttons that invoke other applications, such as a clock or text processor.

Adding, moving, and deleting buttons

You can add, move, and delete buttons in any toolbar.

❖ **To add a button to a toolbar:**

1   Position the pointer on the toolbar and display the pop-up menu.

2   Select Customize.

The Customize dialog box displays.
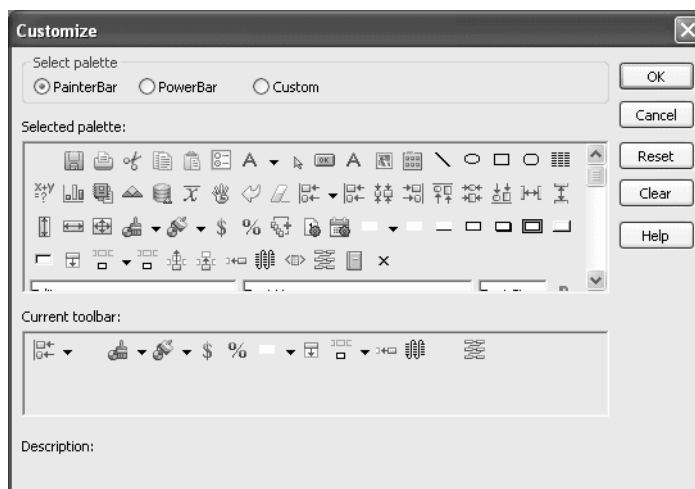
3   Click the palette of buttons you want to use in the Select Palette group box.

4   Choose a button from the Selected Palette box and drag it to the position you want in the Current Toolbar box.

If you choose a button from the Custom palette, another dialog box displays so you can define the button.

For more information, see "Adding a custom button" on page 51.

Seeing what is available in the PowerBar
PowerBuilder provides several buttons that do not display by default in the PowerBar, but which you can add. To see what is available, scroll the list of buttons and select one. PowerBuilder lists the description for the selected button.

❖ **To move a button on a toolbar:**

1   Position the pointer on the toolbar, display the pop-up menu, and select
    Customize.

2   In the Current toolbar box, select the button and drag it to its new position.

❖ **To delete a button from a toolbar:**

1   Position the pointer on the toolbar, display the pop-up menu, and select
    Customize.

2   In the Current toolbar box, select the button and drag it outside the Current
    toolbar box.

Resetting a toolbar   You can restore the original setup of buttons on a toolbar at any time.

❖ **To reset a toolbar:**

1   Position the pointer on the toolbar, display the pop-up menu, and select
    Customize.

2   Click the Reset button, then Yes to confirm, then OK.

Clearing or deleting a   Whenever you want, you can remove all buttons from a toolbar. If you do not
toolbar   add new buttons to the empty toolbar, the toolbar is deleted. You can delete
                             both built-in toolbars and toolbars you have created.

To recreate a toolbar
If you delete one of PowerBuilder's built-in toolbars, you can recreate it easily.
For example, to recreate the PowerBar, display the pop-up menu, select New,
and then select PowerBar1 in the New Toolbar dialog box.

For information about creating new toolbars and about the meaning of
PowerBar1, see "Creating new toolbars" on page 53.

❖ **To clear or delete a toolbar:**

1   Position the pointer on the toolbar, display the pop-up menu, and select
    Customize.

2   Click the Clear button, then Yes to confirm.

    The Current toolbar box in the Customize dialog box is emptied.

3   If you want to add new buttons, select them.

4   Click OK to save the toolbar if you added new buttons, or delete the
    toolbar if you did not.

Adding a custom
button

You can add a custom button to a toolbar. A custom button can:

• Invoke a PowerBuilder menu item

• Run an executable (application) outside PowerBuilder

• Run a query or preview a DataWindow object

• Place a user object in a window or in a custom user object

• Assign a display format or create a computed field in a DataWindow
 object

❖ **To add a custom button:**

1 Position the pointer on the toolbar, display the pop-up menu, and select
 Customize.

2 Select Custom in the Select Palette group box.

 The custom buttons display in the Selected Palette box.

3 Select a custom button and drag it to where you want it in the Current
 Toolbar box.

 The Toolbar Item Command dialog box displays. Different buttons display
 in the dialog box depending on which toolbar you are customizing:
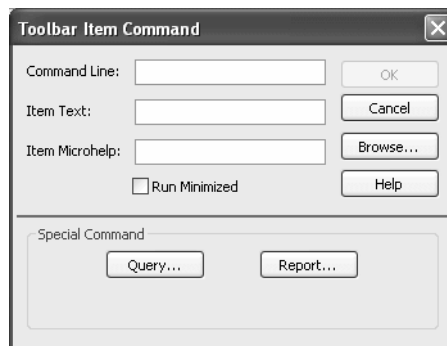
4 Fill in the Command Line box using Table 2-2 on page 52.

5 In the Item Text box, specify the text associated with the button in two
 parts separated by a comma: the text that displays on the button and text
 for the button's PowerTip:

 ButtonText, PowerTip

 For example:

 ```
 Save, Save File
 ```

If you specify only one piece of text, it is used for both the button text and the PowerTip.

6   In the Item MicroHelp box, specify the text to appear as MicroHelp when the pointer is on the button.

Table 2-2: Defining custom buttons

Button action	Toolbar Item Command dialog box entry

Invoke a PowerBuilder menu item	Type
	
	    `@MenuBarItem.MenuItem`
	
	in the Command Line box. For example, to make the button mimic the Open item on the File menu, type
	
	    `@File.Open`
	
	You can also use a number to refer to a menu item. The first item in a drop-down or cascading menu is 1, the second item is 2, and so on. Separator lines in the menu count as items. This example creates a button that pastes a FOR...NEXT statement into a script:
	
	    `@Edit.Paste Special.Statement.6`

Run an executable file outside PowerBuilder	Type the name of the executable file in the Command Line box. Specify the full path name if the executable is not in the current search path. To search for the file name, click the Browse button.
Run a query	Click the Query button and select the query from the displayed list.
Preview a DataWindow object	Click the Report button and select a DataWindow object from the displayed list. You can then modify the command-line arguments in the Command Line box.
Select a user object for placement in a window or custom user object	(Window and User Object painters only) Click the UserObject button and select the user object from the displayed list.
Assign a display format to a column in a DataWindow object	(DataWindow painter only) Click the Format button to display the Display Formats dialog box. Select a data type, then choose an existing display format from the list or define your own in the Format box. For more about specifying display formats, see Chapter 22, "Displaying and Validating Data."
Create a computed field in a DataWindow object	(DataWindow painter only) Click the Function button to display the Function for Toolbar dialog box. Select the function from the list.

Modifying a custom button

❖ **To modify a custom button:**

1 Position the pointer on the toolbar, display the pop-up menu, and select Customize.

2 Double-click the button in the Current toolbar box.

The Toolbar Item Command dialog box displays.

3 Make your changes, as described in "Adding a custom button" on page 51.

Creating new toolbars

PowerBuilder has built-in toolbars. When you start PowerBuilder, you see what is called the PowerBar. In each painter, you also see one or more PainterBars. But PowerBar and PainterBar are actually types of toolbars you can create to make it easier to work in PowerBuilder.

PowerBars and PainterBars

A PowerBar is a toolbar that always displays in PowerBuilder, unless you hide it. A PainterBar is a toolbar that always displays in the specific painter for which it was defined, unless you hide it:

For this toolbar type	The default is named	And you can have up to
PowerBar	PowerBar1	Four PowerBars
PainterBar	PainterBar1 PainterBar2 and so on	Eight PainterBars in each painter

Where you create them

You can create a new PowerBar anywhere in PowerBuilder, but to create a new PainterBar, you must be in the workspace of the painter for which you want to define the PainterBar.

❖ **To create a new toolbar:**

1 Position the pointer on any toolbar, display the pop-up menu, and select New.

The New Toolbar dialog box displays.

About the StyleBar
In painters that do not have a StyleBar, StyleBar is on the list in the New Toolbar dialog box. You can define a toolbar with the name StyleBar, but you can add only painter-specific buttons, not style buttons, to it.

---

2   Select a PowerBar name or a PainterBar name and click OK.

The Customize dialog box displays with the Current toolbar box empty.

3   One at a time, drag the toolbar buttons you want from the Selected palette box to the Current toolbar box and then click OK.

Customizing keyboard shortcuts

You can associate your own keyboard shortcuts with PowerBuilder menu items. For example, if you have used another debugger, you may be accustomed to using specific function keys or key combinations to step into and over functions. You can change the default keyboard shortcuts to associate actions in PowerBuilder's Debugger with the keystrokes you are used to.

Tip
Creating keyboard shortcuts means you can use the keyboard instead of the mouse in many common situations, including changing workspaces, objects, or connections. To do this, create shortcuts for the File>Recent menu items.

❖   **To associate a keyboard shortcut with a menu item:**

1   Select Tools>Keyboard Shortcuts from the menu bar.

The keyboard shortcuts for the current menu bar display.

2   Select a menu item with no shortcut or a menu item with a default shortcut that you want to change and then put the cursor in the Press Keys For Shortcut box.

3   Press the keys you want to use for the shortcut.

The new shortcut displays in the text box. If you type a shortcut that is already being used, a message notifies you so you can type a different shortcut or change the existing shortcut.

Press keys for shortcut:

| Ctrl+S | Remove |

Currently assigned to: Save
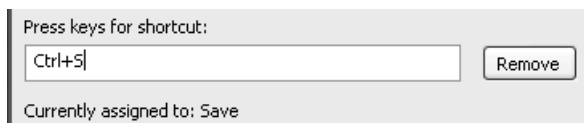
❖ **To remove a keyboard shortcut associated with a menu item:**

1   Select Tools>Keyboard Shortcuts from the menu bar.

2   Select the menu item with the shortcut you want to remove.

3   Click Remove.

You can reset keyboard shortcuts to the default shortcuts globally or for the current painter only.

❖ **To reset keyboard shortcuts to the default:**

•   Click the Reset button and respond to the prompt.

Changing fonts

Table 2-3 summarizes the various ways you can change the fonts used in PowerBuilder.

Table 2-3: Changing the fonts used in PowerBuilder

Object, painter, or tool	How to change fonts
A table's data, headings, and labels	In the Database painter, display the Properties view for the table, and change the font properties on the Data, Heading, and Label Font tabs.
Objects in the User Object, Window, and DataWindow painters	Select objects and then modify settings in the StyleBar, or, in the Properties view for one or more objects, change the font properties on Font tab page.
Application, Menu, and Library painters, System Tree, Output window, Browser, and MicroHelp	Select Tools>System Options from the menu bar and change the font properties on the Editor Font and Printer font tab pages.
Function painter, Script view, Interactive SQL view in the Database painter, Source editor, file editor, and Debug window (changes made for one of these apply to all)	Select Design>Options from the menu bar and change the font properties on the System Font and Printer Font tab pages of the dialog box that displays. In the Debug window, select Debug>Options.

Use the Printer font tab to set fonts specifically for printing. If you need to print multilanguage characters, make sure you use a font that is installed on your printer. Changes you make in the Tools>System Options dialog box and from the Design>Options menu selection are used the next time you open PowerBuilder.

Defining colors

You can define custom colors to use in most painters and in objects you create.

❖ **To define custom colors:**

1 In a painter that uses custom colors, select Design>Custom Colors from the menu bar.
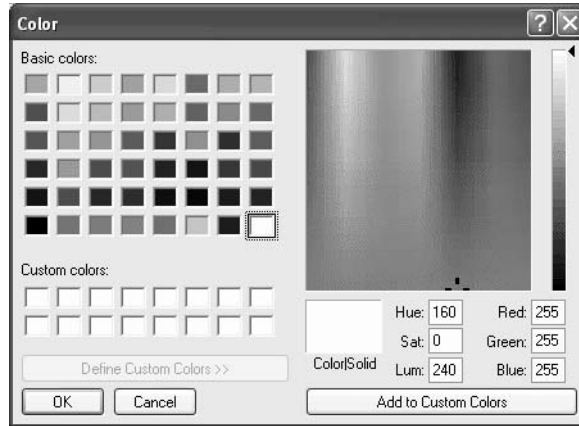
2 Define your custom colors:

Area of the Color dialog box	What you do
Basic colors	Click the basic color closest to the color you want to define to move the pointer in the color matrix and slider on the right.
Custom colors palette	Modify an existing color—click a custom color, then modify the color matrix and slider. Define a new color—click an empty box, define the color, and click Add to Custom Colors.
Color matrix	Click in the color matrix to pick a color.
Color slider	Move the slider on the right to adjust the color's attributes.
Add to Custom Colors button	After you have designed the color, click this button to add the custom color to the Custom colors palette on the left.

How the PowerBuilder environment is managed

PowerBuilder configuration information is stored in both the PowerBuilder initialization file (*PB.INI*) and the registry. When you start PowerBuilder, it looks in the registry and *PB.INI* to set up your environment.

About the registry

Some PowerBuilder features require the use of the *PB.INI* file, but many features use the registry for getting and storing configuration information. Normally, you should not need to access or modify items in the registry directly.

Information related to your preferences (such as the applications you have created, the way you have arranged your views in the painters, and the shortcut keys you have defined for PowerBuilder menu items) is stored in *HKEY_CURRENT_USER/Software/Sybase/PowerBuilder/10.5*.

Installation-related information is stored in *HKEY_LOCAL_MACHINE/Software/Sybase/PowerBuilder/10.5*.

About the initialization file

PB.INI is a text file that contains variables that specify your PowerBuilder preferences. These preferences include information such as the last workspace you used and your startup preferences. When you perform certain actions in PowerBuilder, PowerBuilder writes your preferences to *PB.INI* automatically.

Format of INI files   *PB.INI* uses the Windows INI file format. It has three types of elements:

- Section names, which are enclosed in square brackets

- Keywords, which are the names of preference settings

- Values, which are numeric or text strings, assigned as the value of the associated keyword

A variable can be listed with no value specified, in which case the default is used.

Some sections are always present by default, but others are created only when you specify different preferences. If you specify preferences for another painter or tool, PowerBuilder creates a new section for it at the end of the file.

Specifying preferences

Normally, you do not need to edit *PB.INI.* You can specify all your preferences by taking an action, such as resizing a window or opening a new application, or by selecting Design>Options from one of the painters. If a variable does not appear by default in the options sheet for the painter, you can use a text editor to modify the variable in the appropriate section of *PB.INI.*

Editing the initialization file
Do not use a text editor to edit *PB.INI* or any preferences file accessed by Profile functions while PowerBuilder or your application is running. PowerBuilder caches the contents of initialization files in memory and overwrites your edited *PB.INI* when it exits, ignoring changes.

Where the initialization file is kept

PB.INI is installed in the same directory as the PowerBuilder executable file.

Telling PowerBuilder where your initialization file is

You can keep *PB.INI* in another location and tell PowerBuilder where to find it by specifying the location in the System Options dialog box. You may want to do this if you use more than one version of PowerBuilder or if you are running PowerBuilder over a network.

❖ **To record your initialization path:**

1   Select Tools>System Options from the menu bar.

2   On the General tab page, enter the path of your initialization file in the Initialization Path text box.

PowerBuilder records the path in the Windows registry.

How PowerBuilder finds the initialization file

PowerBuilder looks in the Windows Registry for a path to the initialization file, and then looks for the file in the directory where PowerBuilder is installed. If PowerBuilder cannot find *PB.INI* using the path in the Registry, it clears the path value.

If the initialization file is missing

If PowerBuilder does not find *PB.INI* when it starts up, it recreates it. However, if you want to retain any preferences you have set, such as database profiles, keep a backup copy of *PB.INI*. The recreated file has the default preferences.

CHAPTER 3   **Using Source Control**

About this chapter   PowerBuilder provides a direct connection to external SCC-compliant
source control systems as well as to the PowerBuilder native (PBNative)
utility. This chapter describes how to work with source control.

Contents

Topic	Page
About source control systems	59
Using a source control system with PowerBuilder	66
Source control operations in PowerBuilder	76
Modifying source-controlled targets and objects	90
Migrating existing projects under source control	93

About source control systems

This section provides an overview of source control systems and describes
the PowerBuilder interface (API) to such systems.

What source control
systems do

Source control systems (version control systems) track and store the
evolutionary history of software components. They are particularly useful
if you are working with other developers on a large application, in that
they can prevent multiple developers from modifying the same
component at the same time. You can make sure you are working with the
latest version of a component or object by synchronizing the copy of the
object you are working on with the last version of the object checked into
the source control system.

Why use a source control
system

Most source control systems provide disaster recovery protection and
functions to help manage complex development processes. With a source
control system, you can track the development history of objects in your
PowerBuilder workspace, maintain archives, and restore previous
revisions of objects if necessary.

Source control
interfaces

You work with a source control system through a source control interface.
PowerBuilder supports source control interfaces based on the Microsoft
Common Source Code Control Interface Specification, Version 0.99.0823. You
can use the PowerBuilder SCC API with any source control system that
implements features defined in the Microsoft specification.

PowerBuilder institutes source control at the object level. This gives you a finer
grain of control than if you copied your PBLs directly to source control outside
of the PowerBuilder SCC API.

No other interfaces

PowerBuilder does not support working with source control systems through
proprietary interfaces provided by source control vendors. To work with source
control systems from your PowerBuilder workspace, you must use the
PowerBuilder SCC API. PowerBuilder also uses this API to connect to the
PowerBuilder Native check in/check out utility that installs with the product.

Using your source control manager

The PowerBuilder SCC API works with your source control system to perform
certain source control operations and functions described in the next section.
Other source control operations must be performed directly from the source
control management tool. After you have defined a source control connection
profile for your PowerBuilder workspace, you can open your source control
manager from the Library painter.

❖ **To start your source control manager from PowerBuilder**

• Select Entry>Source Control>Run *Source Control Manager* from the
Library painter menu bar.

The menu item name varies depending on the source control system you
selected in the source control connection profile for your current
workspace. There is no manager tool for the PBNative check in/check out
utility.

For information on configuring a source control connection profile, see
"Setting up a connection profile" on page 67.

Which tool to use

The following table shows which source control functions you should perform from your source control manager and which you can perform from PowerBuilder:

Table 3-1: Where to perform source control operations

Tool or interface	Use for this source control functionality
Source control manager	Setting up a project* Assigning access permissions Retrieving earlier revisions of objects* Assigning revision labels* Running reports* Editing the PBG file for a source-controlled target*
PowerBuilder SCC API	Setting up a connection profile Viewing the status of source-controlled objects Adding objects to source control Checking objects out from source control Checking objects in to source control Clearing the checked-out status of objects Synchronizing objects with the source control server Refreshing the status of objects Comparing local objects with source control versions Displaying the source control version history Removing objects from source control

* You can perform these source control operations from PowerBuilder for some source control systems.

Using PBNative

PowerBuilder provides minimal in-the-box source control through the PBNative check in/check out utility. PBNative allows you to lock the current version of PowerBuilder objects and prevents others from checking out these objects while you are working on them. It provides minimal versioning functionality, and does not allow you to add comments or labels to objects that you add or check in to the PBNative project directory.

Connecting to PBNative

You connect to PBNative from PowerBuilder in the same way you connect to all other source control systems: through the PowerBuilder SCC API. You use the same menu items to add, check out, check in, or get the latest version of objects on the source control server. However, any menu item that calls a source control management tool is unavailable when you select PBNative as your source control system.

Because there is no separate management tool for PBNative, if you need to edit project PBG files that get out of sync, you can open them directly on the server without checking them out of source control.

For more information about PBG files, see "Editing the PBG file for a source-controlled target" on page 92.

PRP files

PBNative creates files with an extra PRP extension for every object registered in the server storage location. If an object with the same file name (minus the additional extension) has been checked out, a PRP file provides the user name of the person who has placed a lock on the object. PRP files are created on the server, not in the local path.

PowerBuilder also adds a version number to the PRP file for an object in the PBNative archive directory when you register that object with PBNative source control. PowerBuilder increments the version number when you check in a new revision. The version number is visible in the Show History dialog box that you open from the pop-up menu for the object, or in the Library painter when you display the object version numbers.

For more information on the Show History dialog box, see "Displaying the source control version history" on page 89. For information on displaying the version number in the Library painter, see "Controlling columns that display in the List view" on page 146.

Using Show Differences functionality with PBNative
PBNative has an option that allows you to see differences between an object on the server and an object on the local machine using a 32-bit visual difference utility that you must install separately. For information on setting up a visual difference utility for use with PBNative, see "Comparing local objects with source control versions" on page 86.

Constraints of a multi-user environment

Any object added or checked into source control should be usable by all developers who have access permissions to that object in source control. This requires that the local paths for objects on different machines be the same in relation to the local root directory where the PowerBuilder workspace resides.

Project manager's
tasks

Before developers can start work on PowerBuilder objects in a workspace
under source control, a project manager usually performs the following tasks:

- Sets up source control projects (and archive databases)

- Assigns each developer permission to access the new project

- Sets up the directory structure for all targets in a project

   Ideally, the project manager should create a subdirectory for each target.
   Whatever directory structure is used, it should be copied to all machines
   used to check out source-controlled objects.

- Distributes the initial set of PBLs and target (PBT) files to all developers
   working on the project or provides a network location from which these
   files and their directory structure can be copied.

PowerScript targets require that all PBLs listed in a target library list be present
on the local machine. For source control purposes, all PBLs in a target should
be in the same local root path, although they could be saved in separate
subdirectories. PBWs and PBLs are not stored in source control unless they are
added from outside the PowerBuilder SCC API. They cannot be checked into
or out of source control using the PowerBuilder SCC API.

If you are sharing PBLs in multiple targets, you can include the shared PBLs
in a workspace and in targets of their own, and create a separate source control
project for the shared objects. After adding (registering) the shared PBL
objects to this project, you can copy the shared targets to other workspaces, but
the shared targets should not be registered with the corresponding projects for
these other workspaces. In this case, the icons indicating source control status
for the shared objects should be different depending on which workspace is the
current workspace.

For small projects, instead of requiring the project manager to distribute PBLs
and target files, developers can create targets in their local workspaces having
the same name as targets under source control. After creating a source control
connection profile for the workspace, a developer can get the latest version of
all objects in the workspace targets from the associated project on the source
control server, overwriting any target and object files in the local root path.

Unfortunately, this does not work well for large PowerScript projects with
multiple PBLs and complicated inheritance schemes, but it can be used for
Web targets. A better way for developers to add source-controlled Web targets
to their workspaces is to use the Source Controlled Web Target wizard on the
Target page of the New dialog box.

Ongoing maintenance tasks of a project manager typically include:

• Distributing any target (PBT) files and PBLs that are added to the workspace during the course of development, or maintaining them on a network directory in an appropriate hierarchical file structure

• Making sure the PBL mapping files (PBGs) do not get out of sync

For information about the PBG files, see "Editing the PBG file for a source-controlled target" on page 92.

Connections from each development machine to the source control project can be defined on the workspace after the initial setup tasks are performed.

Developers' tasks

Each user can define a local root directory in a workspace connection profile. Although the local root directory can be anywhere on a local machine, the directory structure below the root directory must be the same on all machines that are used to connect to the source control repository. Only relative path names are used to describe the location of objects in the workspace below the root directory level.

After copying the directory structure for source-controlled PowerScript targets to the local root path, developers can add these targets to their local workspaces. The target objects can be synchronized in PowerBuilder, although for certain complex targets, it might be better to do the initial synchronization through the source control client tool or on a nightly build machine before adding the targets to PowerBuilder. (Otherwise, the target PBLs may need to be manually rebuilt and regenerated.)

Developers can use the Source Controlled Web Target wizard to add Web targets to their workspaces. The wizard creates subdirectories in the local root path and populates the subdirectories with the latest version of the Web target source files from the source control server.

For more information about getting the latest version of objects in source control, see "Synchronizing objects with the source control server" on page 84.

Extension to the SCC API

Status determination by version number

PowerBuilder provides third-party SCC providers with an extension to the SCC API that allows them to enhance the integration of their products with PowerBuilder. Typically, calls to the SccDiff method are required to determine if an object is out of sync with the SCC repository. (This is not required for Perforce and ClearCase.)

However, SCC providers can implement SccQueryInfoEx as a primary file comparison method instead of SccDiff. The SccQueryInfoEx method returns the most recent version number for each object requested. This allows PowerBuilder to compare the version number associated with the object in the PBL with the version number of the tip revision in the SCC repository in order to determine whether an object is in sync.

Since SccQueryInfoEx is a much simpler request than SccDiff, the performance of the PowerBuilder IDE improves noticeably when this feature is implemented by the SCC provider. For these providers, the SccDiff call is used as a backup strategy only when a version number is not returned on an object in the repository. Also for these providers, the version number for registered files can be displayed in the Library painter.

For more information on viewing the version number, see "Controlling columns that display in the List view" on page 146.

Once the new API method is implemented in an SCC provider DLL and exported, PowerBuilder automatically begins to use the SCCQueryInfoEx call with that provider. The SccQueryInfoEx method is currently used by PBNative.

Overriding the version number

For source control systems that support the SccQueryInfoEx method, you can manually override the version number of local files, but only for PowerScript objects, and only when you are connected to source control.

This can be useful with source control systems that allow you to check out a version of an object that is not the tip revision. However, the source control system alone decides the version number of the tip revision when you check a file back into source control. It is the version returned by the source control system that gets added to the PBC file for the workspace and to the PBLs in the local directory.

For more information about the PBC file, see "Working in offline mode" on page 72.

You change the local version number for a source-controlled PowerScript object in its Properties dialog box, which you access from the object's pop-up menu in the System Tree or the Library painter. If the source control system for the workspace supports the SccQueryInfoEx method and you are connected to source control, the Properties dialog box for a source-controlled PowerScript object (other than a PBT) has an editable SCC Version text box. The SCC Version text box is grayed if the source control system does not support the SccQueryInfoEx method or if you are not connected to source control.

Local change only
The version number that you manually enter for an object is discarded on check-in. Only the source control provider decides what number the tip revision is assigned.

Using a source control system with PowerBuilder

PowerBuilder provides a direct connection to external SCC-compliant source control systems. It no longer requires you to register PowerBuilder objects in a separate work PBL before you can check them into or out of the source control system.

For information on migrating PowerBuilder applications and objects previously checked into source control through a registered PBL, see "Migrating existing projects under source control" on page 93.

Before you can perform any source control operations from PowerBuilder, you must set up a source control connection profile for your PowerBuilder workspace, either from the System Tree or from the Library painter. Even if you use the PBNative check in/check out utility, you must access source-controlled objects through an SCC interface that you define in the Workspace Properties dialog box.

The source control connection profile assigns a PowerBuilder workspace to a source control project. Setting up a source control project is usually the job of a project manager or administrator. See "Project manager's tasks" on page 63.

Creating a new source control project

Although you can create a project in certain source control systems directly from PowerBuilder, it is usually best to create the project from the administrative tool for your source control system before you create the connection profile in PowerBuilder.

Setting up a connection profile

In PowerBuilder you can set up a source control connection profile at the workspace level only. Local and advanced connection options can be defined differently on each machine for PowerBuilder workspaces that contain the same targets.

Local connection options

Local connection options allow you to create a trace log to record all source control activity for your current workspace session. You can overwrite or preserve an existing log file for each session.

You can also make sure a comment is included for every file checked into source control from your local machine. If you select this connection option, the OK button on the Check In dialog box is disabled until you type a comment for all the objects you are checking in.

The following table lists the connection options you can use for each local connection profile:

Table 3-2: Source control properties for a PowerBuilder workspace

Select this option	To do this
Log All Source Management Activity (not selected by default)	Enable trace logging. By default the log file name is *PBSCC105.LOG*, which is saved in your workspace directory, but you can select a different path and file name.
Append To Log File (default selection when logging is enabled)	Append source control activity information to named log file when logging is enabled.
Overwrite Log File (not selected by default)	Overwrite the named log file with source control activity of the current session when logging is enabled.
Require Comments On Check In (not selected by default; not available for PBNative source control)	Disable the OK button on the Check In dialog box until you type a comment.
This Project Requires That I Sometimes Work Offline (not selected by default)	Disable automatic connection to source control when you open the workspace.

Select this option	To do this
Delete PowerBuilder Generated Object Files (not selected by default)	Remove object files (such as SRDs) from the local directory after they are checked into source control. This may increase the time it takes for PowerBuilder to refresh source control status, but it minimizes the drive space used by temporary files. You cannot select this option for the Perforce, ClearCase, or Continuus source control systems.
Perform Diff On Status Update	Permit display of out-of-sync icons for local objects that are different from objects on the source control server. Selecting this also increases the time it takes to refresh source control status.You cannot select this option for Perforce.
Suppress prompts to overwrite read-only files	Avoid message boxes warning that read-only files exist on your local project directory.
Status Refresh Rate (5 minutes by default)	Specifies the minimum time elapsed before PowerBuilder automatically requests information from the source control server to determine if objects are out of sync. Valid values are between 1 and 59 minutes. Status refresh rate is ignored when you are working offline.

Advanced connection options

Advanced connection options depend on the source control system you are using to store your workspace objects. Different options exist for different source control systems.

Applicability of advanced options
Some advanced options might not be implemented or might be rendered inoperable by the PowerBuilder SCC API interface. For example, if an advanced option allows you to make local files writable after an Undo Check Out operation, PowerBuilder still creates read-only files when reverting an object to the current version in source control. (PowerBuilder might even delete these files if you selected the Delete PowerBuilder Generated Object Files option.)

❖ **To set up a connection profile:**

1 Right-click the Workspace object in the System Tree (or in the Tree view of the Library painter) and select Properties from the pop-up menu.

2 Select the Source Control tab from the Workspace Properties dialog box.

3   Select the system you want to use from the Source Control System
    drop-down list.

    Only source control systems that are defined in your registry
    (*HKEY_LOCAL_MACHINE\SOFTWARE\SourceCodeControlProvider\
    InstalledSCCProviders*) appear in the drop-down list.

4   Type in your user name for the source control system.

    Some source control systems use a login name from your registry rather
    than the user name that you enter here. For these systems (such as Perforce
    or PVCS), you can leave this field blank.

5   Click the ellipsis button next to the Project text box.

    A dialog box from your source control system displays. Typically it allows
    you to select or create a source control project.

    The dialog box displayed for PBNative is shown below:
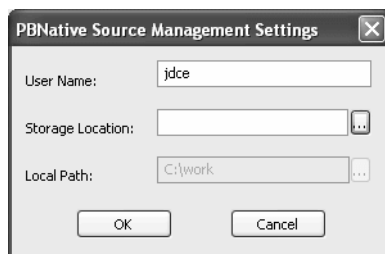
6   Fill in the information required by your source control system and click
    OK.

    The Project field on the Source Control page of the Workspace Properties
    dialog box is typically populated with the project name from the source
    control system you selected. However, some source control systems (such
    as Perforce or Vertical Sky) do not return a project name. For these
    systems, you can leave this field blank.

7   Type or select a path for the local root directory.

    All the files that you check into and out of source control must reside in
    this path or in a subdirectory of this path.

8   (Option) Select the options you want for your local workspace connection
    to the source control server.

9   (Option) Click the Advanced button and make any changes you want to have apply to advanced options defined for your source control system.

The Advanced button might be grayed if you are not first connected to a source control server. If Advanced options are not supported for your source control system, you see only a splash screen for the system you selected and an OK button that you can click to return to the Workspace Properties dialog box.

10   Click Apply or click OK.

Viewing the status of source-controlled objects

After a PowerBuilder workspace is assigned to a source control project through a connection profile, icons in the PowerBuilder System Tree display the source control status of all objects in the workspace. The same icons are also displayed for objects in the Library painter if the workspace to which they belong is the current workspace for PowerBuilder.

Source control icons   The icons and their meanings are described in Table 3-3 and Table 3-4.

Table 3-3: Source control status icons in PowerBuilder

Icon	Source control status of object displaying icon
+	The object resides only locally and is not under source control.
.	The object is under source control and is not checked out by anyone. The object on the local machine is in sync with the object on the server unless the icon for indeterminate status also appears next to the same object.
✓	The object is checked out by the current user.
✕	The object is checked out by another user.
?	The current status of an object under source control has not been determined. You are likely to see this icon only if the Perform Diff On Status Update check box is not selected and if diffs are not performed for your source control system based on version number. This icon can appear only in conjunction with the icon for a registered object (green dot icon) or for an object checked out by another user (red x icon).
↻	The object on the local machine is registered to source control, but is out of sync with the object on the server. This icon can also appear with the icon for an object checked out by another user. The Perform Diff On Status Update check box must be selected for this icon to display.

Compound icons with a red check mark can display only if you add a *PB.INI* setting to permit multiple user checkouts. These icons are described in the following table:

Table 3-4: Source control status icons with multiple checkouts enabled

Icon	Source control status of object displaying icon
	The object is under source control and is checked out nonexclusively by another user. PowerBuilder allows a concurrent checkout by the current user.
	The object is checked out by both the current user and another user.
	The object is checked out nonexclusively by another user and the version in the current user's local path is out of sync.

For more information on allowing multiple user checkouts, see "Checking objects out from source control" on page 79.

Pop-up menus

Pop-up menus for each object in the workspace change dynamically to reflect the source control status of the object. For example, if the object is included in a source-controlled workspace but has not been registered to source control, the Add To Source Control menu item is visible and enabled in the object's pop-up menu. However, other source control menu items such as Check In and Show Differences are not visible until the object is added to source control.

Library painter Entry menu

Additional status functionality is available from the Entry menu of the Library painter. Depending on the source control system you are using, you can see the owner of an object and the name of the user who has the object checked out. For most source control systems, you can see the list of revisions, including any branch revisions, as well as version labels for each revision.

Library painter selections

When a painter is open, menu commands apply to the current object or objects in the painter, not the current object in the System Tree. This can get confusing with the Library painter in particular, since Library painter views list objects only (much like the System Tree), and do not provide a more detailed visual interface for viewing current selections, as other painters do.

❖ **To view the status of source-controlled objects**

1 In a Library painter view, select the object (or objects) whose status you want to determine.

2 Select Entry>Source Control>*Source Control Manager* Properties.

A dialog box from your source control system displays. Typically it indicates if the selected file is checked in, or the name of the user who has the file checked out. It should also display the version number of the selected object.

Displaying the version number in the Library painter
You can display the version number of all files registered in source control directly in the Library painter. You add a Version Number column to the Library painter List view by making sure the SCC Version Number option is selected in the Options dialog box for the Library painter.

For more information, see "Controlling columns that display in the List view" on page 146.

Working in offline mode

Viewing status information offline

You can work offline and still see status information from the last time you were connected to source control. However, you cannot perform any source control operations while you are offline, and you cannot save changes to source-controlled objects that you did not check out from source control before you went offline.

To be able to work offline, you should select the option on the Source Control page of the Workspace Properties dialog box that indicates you sometimes work offline. If you select this option, a dialog box displays each time you open the workspace. The dialog box prompts you to select whether you want to work online or offline.

For more information about setting source control options for your workspace, see "Setting up a connection profile" on page 67.

About the PBC file

If you opt to work offline, PowerBuilder looks for (and imports) a PBC file in the local root directory. The PBC file is a text file that contains status information from the last time a workspace was connected to source control. PowerBuilder creates a PBC file only from a workspace that is connected to source control. Status information is added to the PBC file from expanded object nodes (in the System Tree or in a Library painter view) at the time you exit the workspace.

If a PBC file already exists for a workspace that is connected to source control, PowerBuilder merges status information from the current workspace session to status information already contained in the PBC file. Newer status information for an object replaces older status information for the same object, but older status information is not overwritten for objects in nodes that were not expanded during a subsequent workspace session.

Backing up the PBC file

You can back up the PBC file with current checkout and version information by selecting the Backup SCC Status Cache menu item from the Library painter Entry>Source Control menu, or from the pop-up menu on the current workspace item in the System Tree. The Library painter menu item is only enabled when the current workspace file is selected.

The Backup SCC Status Cache operation copies the entire contents of the refresh status cache to the PBC file in the local project path whether the status cache is dirty or valid. To assure a valid status cache, you can perform a Refresh Status operation on the entire workspace before backing up the SCC status cache.

For information about refreshing the status cache, see "Refreshing the status of objects" on page 85.

Fine-tuning performance for batched source control requests

PowerBuilder uses an array of object file names that it passes to a source control system in each of its SCC API requests. The SCC specification does not mention an upper limit to the number of files that can be passed in each request, but the default implementation in PowerBuilder limits SCC server requests to batches of 25 objects.

A PB.INI file setting allows you to override the 25-file limit on file names sent to the source control server in a batched request. You can make this change in the Library section of the PB.INI file by adding the following instruction:

```
SccMaxArraySize=nn
```

where *nn* is the number of files you want PowerBuilder to include in its SCC API batch calls. Like other settings in the *PB.INI* file, the SccMaxArraySize parameter is not case sensitive.

Configuring Java VM initialization

When you connect to a source control system, PowerBuilder instantiates a Java VM by default. For certain SCC programs, such as Borland's StarTeam or Serena's TrackerLink, the Java VM instantiated by PowerBuilder conflicts with the Java VM instantiated by the SCC program. To prevent Java VM conflicts, you must add the following section and parameter setting to the PB.INI file:

```
[JavaVM]
CreateJavaVM=0
```

By adding this section and parameter setting to the PB.INI file, you prevent PowerBuilder from instantiating a Java VM when it connects to a source control system.

Files available for source control

The following schema shows a directory structure for files in the local PowerBuilder workspace and on the source control server. Directories and files in the local root path that can be copied to the source control server from PowerBuilder are displayed in bold print. Asterisks indicate a variable name for a file and italic print indicates a variable name for a file or folder.

Figure 3-1: Directory structure in local path and source control server
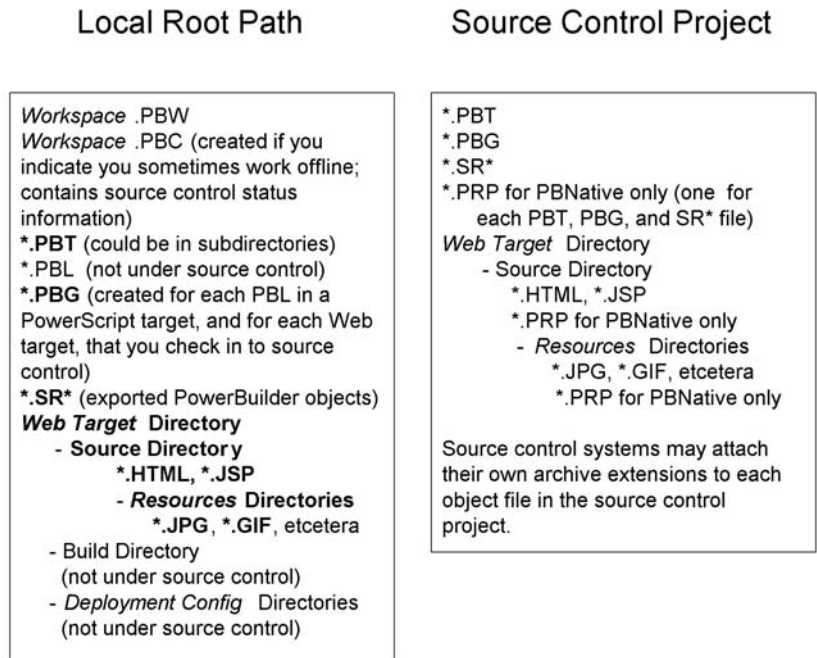
Typically, the source control server files are stored in a database but preserve the file system structure. Files in a Web target *Build* directory and in any deployment configuration directories can be regenerated automatically by building and deploying the files in the *Source* directory.

Temporary files in local root path

When you add or check in a PowerScript object to source control, PowerBuilder first exports the object as a temporary file (**.SR**) to your local target directory. For some source control systems, you might choose to delete temporary files from the local root path.

Source control operations in PowerBuilder

The following source control operations are described in this section:

- Adding objects to source control

- Checking objects out from source control

- Checking objects in to source control

- Clearing the checked-out status of objects

- Synchronizing objects with the source control server

- Refreshing the status of objects

- Comparing local objects with source control versions

- Displaying the source control version history

- Removing objects from source control

Source control operations on workspace and PBL files are performed on the objects contained in the current workspace or in target PBLs, not on the actual PBW and PBL files. The PBW and PBL files cannot be added to source control through the PowerBuilder interface. Source control operations are not enabled for target PBD files or for any of the objects in target PBD files.

Adding objects to source control

You add an object to your source control project by selecting the Add To Source Control menu item from the object's pop-up menu in the System Tree or in the Library painter. You can also select an object in a Library painter view and then select Entry>Source Control>Add To Source Control from the Library painter menu bar.

What happens when you add objects to source control

When you add an object to source control, the icon in front of the object changes from a plus sign to a green dot, indicating that the object on the local machine is in sync with the object on the server.

PowerBuilder creates read-only object files in the local root directory for each PowerBuilder object that you add to source control. These files can be automatically deleted if you selected the Delete PowerBuilder Generated Object Files option as a source control connection property (although you cannot do this for certain SCC systems such as Perforce or ClearCase).

Read-only attributes are also assigned to any Web target files that you add to source control. Read-only attributes are not changed by PowerBuilder if you later remove a workspace containing these files from source control.

Adding multiple objects to source control

If the object you select is a PowerBuilder workspace, a dialog box displays listing all the objects for that workspace that are not currently under source control (although the workspace PBW and target PBLs are not included in the list). If the object you select is a PowerBuilder target, and at least one of the objects in that target has not been registered with the current source control project, PowerBuilder displays a dialog box that prompts you to:

- Select multiple files contained in the target

- Register the target file only

If you select the multiple files radio button, another dialog box displays with a list of objects to add to source control. A check box next to each object lets you select which objects you want to add to source control. By default, check boxes are selected for all objects that are not in your source control project. They are not selected for any object already under source control.
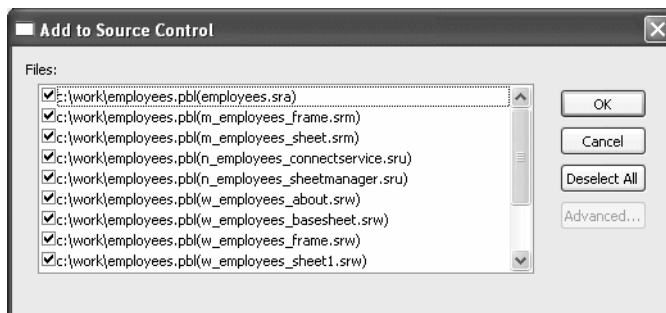
You can resize all source control dialog boxes listing multiple files by placing a cursor over the edge of a dialog box until a two-headed arrow displays, then dragging the edge in the direction of one of the arrow heads.

Selecting multiple files from a PBL
If you select Add To Source Control for a target PBL, you immediately see the list of multiple files from that PBL in the Add To Source Control dialog box. There is no need for an intervening dialog box as there is for a target or workspace, since you cannot register a PBL file to source control from the PowerBuilder UI—you can only register the objects contained in that PBL.

You can also select multiple objects to add to source control from the List view of the Library painter (without selecting a workspace, target, or PBL).

The Add To Source Control menu item is disabled for all objects that are registered in source control except workspaces and targets. If you select the Add To Source Control menu item for a workspace or target in which all the objects are already registered to source control, PowerBuilder displays the Add To Source Control dialog box with an empty list of files. You cannot add objects to your source control project that are already registered with that project.

Creating a mapping file for target PBLs

When you add a target or an object (in a target that is not under source control) to source control, PowerBuilder creates a PBG file. A PBG file maps objects in a target to a particular PBL in a PowerScript target. One PBG file is created per PBL, so there can be multiple PBG files per PowerScript target. PBG files are also used to list target directories and subdirectories for Web targets, but there is only one PBG file per Web target.

If a PBG file already exists for a target PBL containing the object you are adding to source control, PowerBuilder checks the PBG file out of source control and adds the name of the object to the names of objects already listed in the PBG file. It then checks the PBG file back into source control.

The PBG files are used by PowerBuilder to make sure that objects are distributed to the correct PBLs and targets when you check the objects out (or get the latest versions of the objects) from source control.

If your source control system requires comments on registration and check-in, you get separate message boxes for the PBG file and the objects that you are adding to source control. If your source control system gives you the option of adding the same comments to all the objects you are registering, you can still get additional message boxes for PBG files, since PBG files are checked in separately.

Because it is possible for PBG files to get out of sync, it is important that the project manager monitor these files to make sure they map all objects to the correct PBLs and contain references to all objects in the source control project. However, you cannot explicitly check in or check out PBG files through the PowerBuilder SCC API.

For more information on modifying PBG files, see "Editing the PBG file for a source-controlled target" on page 92.

Checking objects out from source control

Enabling multiple user checkout

Checking out an object from a source control system usually prevents other users from checking in modified versions of the same object. By default, PowerBuilder assumes that any object checked out by a user is exclusively reserved for that user until the object is checked back in.

Some source control systems, such as Serena Version Manager (formerly Merant PVCS) and MKS Source Integrity, permit multiple user checkouts. In these systems, you can allow shared checkouts of the same object. PowerBuilder can recognize shared checkouts, but only if you add the following instruction to the Library section of the PB.INI file:

```
[Library]
SccMultiCheckout=1
```

After you add this PB.INI setting, PowerBuilder shows a red check mark as part of a compound icon to indicate that an object is checked out to another user in a shared (nonexclusive) mode. PowerBuilder allows you to check out an object with this compound icon, even though another user has already checked the object out.

Multiple user checkouts
If you configure the PB.INI file to allow multiple user checkouts, you must make sure that you have a way to manage updates to the same object by multiple users. The merge functionality is not supported by the SCC API, so either you must configure merge operations through the Advanced Check Out dialog box of the source control system, or you must merge the changes using the source control management system instead of the PowerBuilder user interface.

What happens

When you check out an object, PowerBuilder:

• Locks the object in the archive so that no one else can modify it

• Copies the object to the directory for the target to which it belongs

• For a PowerScript object, compiles the object and regenerates it in the target PBL to which it is mapped

• Displays a check mark icon next to the object in your System Tree and in your Library painter to show that the object has been checked out

Checking out multiple objects

If you select the Check Out menu item for a PowerBuilder target that is not already checked out, and at least one of the objects in that target is available for checkout, PowerBuilder displays a dialog box that prompts you to:

• Select multiple files contained in the target

• Check out the target file only

If you select the multiple file option, or if the target file is already checked out, the Check Out dialog box displays the list of objects from that target that are available for checkout. A check box next to each object in the list lets you choose which objects you want to check out. By default, check boxes are selected for all objects that are not currently checked out of source control.

The Deselect All button in the Check Out dialog box lets you clear all the check boxes with a single click. When none of the objects in the list is selected, the button text becomes Select All, and you can click the button to select all the objects in the list.

You can also select multiple objects (without selecting a target) in the List view of the Library painter. The PowerBuilder SCC API does not let you check out an object that you or someone else has already checked out or that is not yet registered with source control. If you use multiple object selection to select an object that is already checked out, PowerBuilder does not include this object in the list view of the Check Out dialog box.

Creating a source control branch

If your source control system supports branching and its SCC API lets you check out a version of an object that is not the most recent version in source control, you can select the version you want in the Advanced Check Out dialog box (that you access by clicking the Advanced button in the Check Out dialog box). When you select an earlier version, PowerBuilder displays a message box telling you it will create a branch when you check the object back in. You can click Yes to continue checking out the object or No to leave the object unlocked in the source control project. If this is part of a multiple object checkout, you can select Yes To All or No To All.

If you want just a read-only copy of the latest version of an object
Instead of checking out an object and locking it in the source control system, you can choose to get the latest version of the object with a read-only attribute. See "Synchronizing objects with the source control server" on page 84.

❖   **To check out an object from source control:**

1   Right-click the object in the System Tree or in a Library painter view and select Check Out from the pop-up menu
    or
    Select the object in a Library painter view and select Entry>Source Control>Check Out from the Library painter menu.

    The Check Out dialog box displays the name of the object you selected. For PowerScript objects, the object listing includes the name of the PBL that contains the selected object.

    If you selected multiple objects, the Check Out dialog box displays the list of objects available for checkout. You can also display a list of available objects when you select a target file for checkout. A check mark next to an object in the list marks the object as assigned for checkout.

2   Make sure that the check box is selected next to the object you want to check out, and click OK.

Checking objects in to source control

When you finish working with an object that you checked out, you must check it back in so other developers can use it, or you must clear the object's checked-out status. You cannot check in objects that you have not checked out.

If you do not want to use the checked-out version
Instead of checking an entry back in, you can choose not to use the checked-out version by clearing the checked-out status of the entry. See "Clearing the checked-out status of objects" next.

Checking in multiple objects

If you select the Check In menu item for a workspace, PowerBuilder lists all the objects in the workspace that are available for check-in. If you select the Check In menu item for a PowerBuilder target that is currently checked out to you, and at least one of the objects in that target is also checked out to you, PowerBuilder displays a dialog box that prompts you to:

•   Select multiple files contained in the target

•   Check in the target file only

If you select the multiple file option, or if the target file is not currently checked out to you, the Check In dialog box displays the list of objects from that target that are available for you to check in. A check box next to each object in the list lets you choose which objects you want to check in. By default, check boxes are selected for all objects that you currently have checked out of source control.

The Deselect All button in the Check In dialog box lets you clear all the check boxes with a single click. When none of the objects in the list is selected, the button text becomes Select All, and you can click the button to select all the objects in the list.

You can also select multiple objects (without selecting a workspace or target) in the List view of the Library painter. The PowerBuilder SCC API does not let you check in an object that you have not checked out of source control. If you use multiple object selection to select an object that is not checked out to you, PowerBuilder does not include this object in the list view of the Check In dialog box.

❖ **To check in objects to source control:**

1   Right-click the object in the System Tree or in a Library painter view and select Check In from the pop-up menu
or
Select the object in a Library painter view and select Entry>Source Control>Check In from the Library painter menu.

The Check In dialog box displays the name of the object you selected. If you selected multiple objects or a workspace, the Check In dialog box displays the list of objects available for check-in. You can also display a list of available objects when you select a target file. A check mark next to an object in the list marks the object as assigned for check-in.

2   Make sure the check box is selected next to the object you want to check in and click OK.

Clearing the checked-out status of objects

Sometimes you need to clear (revert) the checked-out status of an object without checking it back into source control. This is usually the case if you modify the object but then decide not to use the changes you have made. When you undo a checkout on an object, PowerBuilder replaces your local copy with the latest version of the object on the source control server. For PowerScript targets, it compiles and regenerates the object in its target PBL.

Clearing the status of
multiple objects

If you select the Undo Check Out menu item for a PowerBuilder target that is
checked out to you, and at least one of the objects in that target is also checked
out to you, PowerBuilder displays a dialog box that prompts you to:

- Select multiple files contained in the target

- Undo the checked-out status for the target file only

If you select the multiple file option, or if the target file is not currently checked
out to you, the Undo Check Out dialog box displays the list of objects from that
target that are locked by you in source control. A check box next to each object
in the list lets you choose the objects for which you want to undo the
checked-out status. By default, check boxes are selected for all objects that are
currently checked out to you from source control.

You can also select multiple objects (without selecting a target) in the List view
of the Library painter. The PowerBuilder SCC API does not let you undo the
checked-out status of an object that you have not checked out of source control.
If you use multiple object selection to select an object that is not checked out
to you, PowerBuilder does not include this object in the list view of the Undo
Check Out dialog box.

❖ **To clear the checked-out status of entries:**

1 Right-click the object in the System Tree or in a Library painter view and
   select Undo Check Out from the pop-up menu
   or
   Select the object in a Library painter view and select Entry>Source
   Control>Undo Check Out from the Library painter menu.

   The Undo Check Out dialog box displays the name of the object you
   selected. If you selected multiple objects, the Undo Check Out dialog box
   displays the list of objects in the selection that are currently checked out to
   you. You can also display a list of objects that are checked out to you when
   you select a target file.

2 Make sure that the check box is selected next to the object whose
   checked-out status you want to clear, and click OK.

Synchronizing objects with the source control server

You can synchronize local copies of PowerBuilder objects with the latest versions of these objects in source control without checking them out from the source control system. The objects copied to your local machine are read-only. The newly copied PowerScript objects are then compiled into their target PBLs.

If there are exported PowerScript files in your local path that are not marked read-only, and you did not select the Suppress Prompts To Overwrite Read-Only Files option, your source control system may prompt you before attempting to overwrite these files during synchronization. If you are synchronizing multiple objects at the same time, you can select:

• Yes To All, to overwrite all files in your selection

• No To All, to cancel the synchronization for all objects in the selection that have writable files in the local path

Synchronizing an object does not lock that object on the source control server. After you synchronize local objects to the latest version of these objects in source control, other developers can continue to perform source control operations on these objects.

If you want only to check whether the status of the objects has changed on the source control server, you can use the Refresh Status menu item from the Library painter Entry menu or System Tree pop-up menus. The Refresh Status command runs on a background thread. If you do not use the Refresh Status feature before getting the latest versions of workspace or target objects, then PowerBuilder has to obtain status and out-of-sync information from the SCC provider in real time during a GetLatestVersion call.

For more information, see "Refreshing the status of objects" on page 85.

❖ **To synchronize a local object with the latest source control version:**

1   Right-click the object in the System Tree or in a Library painter view and select Get Latest Version from the pop-up menu.
    or
    Select the object in a Library painter view and select Entry>Source Control>Get Latest Version from the Library painter menu.

The Get Latest Version dialog box displays the name of the object you selected. If you selected multiple objects in the Library painter List view, the Get Latest Version dialog box lists all the objects in your selection. If you selected a workspace, the Get Latest Version dialog box lists all the objects referenced in the PBG files belonging to your workspace. You can also display a list of available objects (from the PBG files for a target) when you select the Get Latest Version menu item for a target file.

A check mark next to an object in the list assigns the object for synchronization. By default only objects that are currently out of sync are selected in this list. You can use the Select All button to select all the objects for synchronization. If all objects are selected, the button text becomes Deselect All. Its function also changes, allowing you to clear all the selections with a single click.

2   Make sure that the check box is selected next to the object for which you want to get the latest version, and click OK.

Refreshing the status of objects

PowerBuilder uses the source control connection defined for a workspace to check periodically on the status of all objects in the workspace. You can set the status refresh rate for a workspace on the Source Control page of the Workspace Properties dialog box. You can also select the Perform Diff on Status Update option to detect any differences between objects in your local directories and objects on the source control server.

For more information about source control options you can set on your workspace, see "Setting up a connection profile" on page 67.

PowerBuilder stores status information in memory, but it does not automatically update the source control status of an object until a System Tree or Library painter node containing that object has been expanded and the time since the last status update for that object exceeds the status refresh rate.

Status information can still get out of sync if multiple users access the same source control project simultaneously and you do not refresh the view of your System Tree or Library painter. By using the Refresh Status menu item, you can force a status update for objects in your workspace without waiting for the refresh rate to expire, and without having to open and close tree view nodes containing these objects.

The Refresh Status feature runs in the background on a secondary thread. This allows you to continue working in PowerBuilder while the operation proceeds. When the Refresh Status command is executed, your SCC status cache is populated with fresh status values. This allows subsequent operations like a target-wide synchronization (through a GetLatestVersion call) to run much faster.

❖ **To refresh the status of objects:**

1 Right-click the object in the System Tree or in a Library painter view and select Refresh Status from the pop-up menu
   or
   Select the object in a Library painter view and select Entry>Source Control>Refresh Status from the Library painter menu.

   If the object you selected is not a workspace, target, or PBL file, the object status is refreshed and any change is made visible by a change in the source control icon next to the object. If you selected an object in a Library painter view, the status of this object in the System Tree is also updated.

   For information about the meaning of source control icons in PowerBuilder, see "Viewing the status of source-controlled objects" on page 70.

2 If the object you selected in step 1 is a workspace or target file, select a radio button to indicate whether you want to refresh the status of the selected file only or of multiple files in the workspace or target.

3 If the object you selected in step 1 is a PBL, or if you selected the multiple files option in step 2, make sure that the check box is selected next to the object or objects whose status you want to refresh, and click OK.

   Status is refreshed for every object selected in the Refresh Status dialog box. Any change in status is made visible by a change in the source control icon next to the objects (in the selected workspace, target, or PBL) that are refreshed.

Comparing local objects with source control versions

The PowerBuilder SCC API lets you compare an object in your local directory with a version of the object in the source control archive (or project). By default, the comparison is made with the latest version in the archive, although most source control systems let you compare your local object to any version in the archive. Using this feature, you can determine what changes have been made to an object since it was last checked into source control.

Setting up PBNative for object comparisons

PBNative does not have its own visual difference utility, but it does allow you to select one that you have already installed. You must use only a 32-bit visual difference utility for the object comparisons. You can select any or all of the following options when you set up the utility to work with a PBNative repository:

Table 3-5: Object comparison options for use with PBNative

Option	Select this if
Enclose file names in double quotes	Your visual difference utility does not handle spaces in file names.
Refer to local PBL entry as argument #1	You do not want the visual difference utility to use the repository object as the first file in a file comparison.
Generate short (8.3) file names	Your visual difference utility does not handle long file names.
Generate an extra space prior to file arguments	Your visual difference utility requires an extra space between files that are listed as arguments when you open the utility from a command line. This option was added for backward compatibility only, as an extra space was automatically added by PowerBuilder 8.

❖ **To set up PBNative for object comparisons**

1   Right-click the Workspace object in the System Tree and click the Source Control tab in the Workspace Properties dialog box.

PBNative should be your selection for the source control system, and you must have a project and local root directory configured. If you are connected already to source control, you can skip the next step.

2   Click Connect.

The Connect button is disabled if you are already connected to source control.

3   Click Advanced.

The PBNative Options dialog box displays.

4   Type the path to a visual difference utility followed by the argument string required by your utility to perform a diff (comparison) on two objects.

Typically, you would add two *%s* parameter markers to indicate where PowerBuilder should perform automatic file name substitution. The following figure shows a setting used to call the Microsoft *WinDiff* utility:
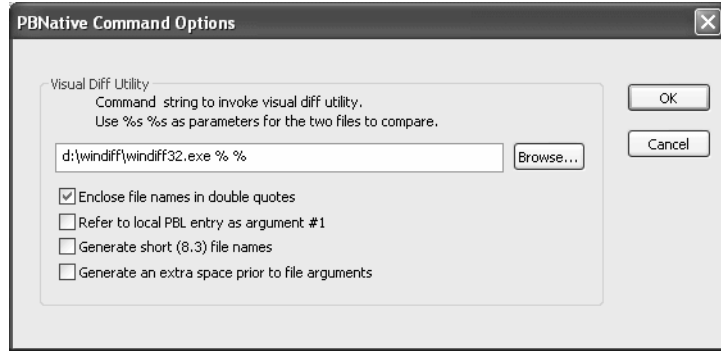
5 (Optional) Select any or all of the check box options in the PBNative Command Options dialog box for your object comparisons.

6 Click OK twice.

You are now set to use your visual difference utility to compare objects on the local machine and the server.

Using Show Differences to compare objects

You can select Show Differences from a pop-up menu or from the Library painter menu bar. If the object you want to compare has not been added to the source control project defined for your workspace, the Show Differences menu item is not available.

❖ **To compare a local object with the latest source control version:**

1 Right-click the object in the System Tree or in a Library painter view and select Show Differences from the pop-up menu
or
Select the object in a Library painter view and select Entry>Source Control>Show Differences from the Library painter menu bar.

A dialog box from your source control system displays.

PBNative connections
Skip the next step if you are using a visual difference utility with PBNative. The difference utility displays the files directly or indicates that there are no differences between the files.

---

2   Select the source control comparison options you want and click OK.

Some source control systems support additional comparison functions. You may need to run the source control manager for these functions. See your source control system documentation for more information.

Displaying the source control version history

For some source control systems, the PowerBuilder SCC API lets you show the version control history of an object in source control. Using this feature, you can determine what changes have been made to an object since it was first checked into source control.

The Show History menu item is not visible if the object for which you want to display a version history has not been added to the source control project defined for your workspace. It is grayed out if your source control system does not support this functionality through the PowerBuilder SCC API.

❖   **To display the source control version history:**

1   Right-click the object in the System Tree or in a Library painter view and select Show History from the pop-up menu
    or
    Select the object in a Library painter view and select Entry>Source Control>Show History from the Library painter menu bar.

A dialog box from your source control system displays.

2   Select the source control options you want and click OK.

Some source control systems support additional tracing and reporting functions for objects in their archives. You may need to run the source control manager for these functions. See your source control system documentation for more information.

Removing objects from source control

The PowerBuilder SCC API lets you remove objects from source control, although for some source control systems, you may have to use the source control manager to delete the archives for the objects you remove. You cannot remove an object that is currently checked out from source control.

You cannot delete a source-controlled object from a local PowerBuilder workspace before that object has been removed from source control. There is no requirement, however, that the source control archive be deleted before you delete the object from its PowerBuilder workspace.

❖ **To remove objects from source control:**

1   Select the object in a Library painter view and select Entry>Source Control>Remove From Source Control from the Library painter menu.

The Remove From Source Control dialog box displays the name of the object you selected.

If you selected multiple objects or a workspace, the Remove From Source Control dialog box displays the list of objects in your selection that are not currently checked out from source control. You can also display a list of available objects when you select the Remove From Source Control menu item for a target file. A check mark next to an object in the list marks the object as assigned for removal from source control.

2   Make sure that the check box is selected next to the object you want to remove, and click OK.

Modifying source-controlled targets and objects

Objects in targets under source control must be managed differently than the same objects in targets that are not under source control.

Effects of source control on object management

You must check out a target file from source control before you can modify its properties. If objects in a source-controlled target are not themselves registered in source control, you can add them to or delete them from the local target without checking out the target. However, you must remove a source-controlled object from the source control system before you can delete the same object from the local copy of the target (whether or not the target itself is under source control).

Although you can add objects to a source-controlled target without checking out the target from source control, you cannot add existing libraries to the library list of a source-controlled PowerScript target unless the target is checked out.

For information on removing an object from source control, see "Removing objects from source control" on page 89.

Copy and move operations on source-controlled objects

You cannot copy a source-controlled object to a destination PBL in the same directory as the source PBL. Generally when you work with source control, objects with the same name should not exist in more than one PBL in the same directory.

Moving an object that is not under source control to a destination PBL having a source-controlled object with the same name is permitted only when the second object is checked out of source control.

You cannot move an object from a source PBL if the object is under source control, even when the object has been checked out. The right way to move an object under source control is described below.

❖ **To move an object under source control from one PBL to another:**

1   Export the object from the first PBL.

2   Remove the object from source control.

See "Removing objects from source control" on page 89.

3   Delete the object from the first PBL.

4   Import the object into the second PBL.

5   Register the object in source control once again.

Editing the PBG file for a source-controlled target

PowerBuilder creates and uses PBG files to determine if any objects present on a source control server are missing from local PowerScript targets. Up-to-date PBG files insure that the latest objects in source control are available to all developers on a project, and that the objects are associated with a named PBL file.

PBG files and Web targets
PowerBuilder also creates PBG files for Web targets, but these list only the directory structure for the targets, not the individual files. Since Web Target files are flat files, you can check them out (or get the latest versions) directly from source control without having to compile them into PBLs.
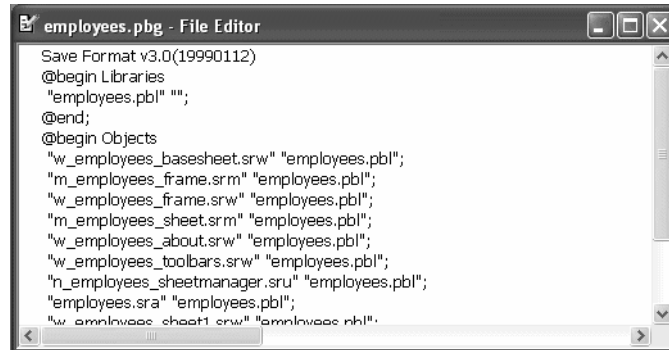
If you do not have all the files on your local machine—and your source control system supports the SCCPopulateList call—you might still be able to check out or get the latest versions of these files through the PowerBuilder SCC API. In this case, you might still need to force a status refresh in order to view the new objects in the System Tree or the Library painter.

Ideally, PBG files are not necessary. If the source control system exposes the latest additions of objects in a project through its SCC interface, PowerBuilder can obtain the list of all objects added to a project since the last status refresh. However, many source control systems do not support this, so PowerBuilder uses the PBG files to make sure it has an up-to-date list of objects under source control.

PBG files are registered and checked in to source control separately from all other objects in PowerBuilder. They are automatically updated to include new objects that are added to source control, but they can easily get out of sync when multiple users simultaneously register objects to (or delete objects from) the same source control project. For example, it is possible to add an object to source control successfully yet have the check-in of the PBG file fail because it is locked by another user.

You cannot see the PBG files in the System Tree or Library painter unless you set the root for these views to the file system. To edit PBG files manually, you should check them out of source control using the source control manager and open them in a text editor. (If you are using PBNative, you can edit PBG files directly in the server storage location, without checking them out of source control.)

You can manually add objects to the PBG file for a PowerBuilder library by including a new line for each object after the `@begin Objects` line. The following is an example of the contents of a PBG file for a PBL that is saved to a subdirectory (*target1*) of the workspace associated with the source control project:

Migrating existing projects under source control

Migrating from
PowerBuilder 8

There are different strategies for migrating existing source control projects from earlier versions of PowerBuilder. To migrate a target from PowerBuilder 8 or later, you can check the target out from source control, then add the target to a workspace in the new version of PowerBuilder. When the Migrate Application dialog box prompts you to migrate the libraries in the application library list, click OK to begin the migration.

If you change the directory hierarchy for target libraries in the new version of PowerBuilder, you should use the Existing Application target wizard to create a new target instead of adding and migrating a PowerBuilder 8 or later target. If you keep the old target file (PBT) in the new target path, you must give the new target a different name or the wizard will not be able to create a new PBT.

For information on using the Existing Application target wizard, see "Using the Existing Application target wizard" on page 94.

When you open a PowerBuilder 8 or later workspace in the current version of PowerBuilder, a dialog box prompts you to migrate the workspace targets. If you select the No Prompting check box and click OK, the target libraries are migrated without additional prompting, and the Migrate Application dialog box never displays. You can then add the migrated target objects to source control from the new version of PowerBuilder.

For more information about migration, see "Migrating targets" on page 159.

Uninstalling PowerBuilder 8
If you uninstall PowerBuilder 8 from a machine where you have already installed a later version of PowerBuilder, the uninstall deletes the PBNative registry entry. Subsequently, if you want to use PBNative source control with PowerBuilder 10, you must reregister *PBNAT105.DLL.* Attempting to use PBNative before reregistering the DLL produces an error message that points out the problem and the solution. You can reregister the DLL by opening a DOS command box, switching to the *Sybase\Shared\PowerBuilder* directory, and entering the command REGSVR32 PBNAT105.DLL.

The uninstall for PowerBuilder 9 and later does not remove the DLL or the registry entry for PBNative source control.

Migrating from
PowerBuilder 7 or
earlier

Migrating an application from PowerBuilder 7 or earlier requires a different approach, since workspaces and targets were introduced with PowerBuilder 8. You need to create a new workspace and appropriate targets for any PowerBuilder 7 (or older) objects that you are migrating.

The strategies available to you or the project manager are:

• Using the Existing Application target wizard

• Importing source control files to a new library

To use the first strategy, you must keep a copy of the old version of PowerBuilder—at least until you have finished migrating all your source-controlled PBLs.

Using the Existing Application target wizard

Source control in early
PowerBuilder versions

Because workspaces and targets were not available in PowerBuilder prior to version 8, you must use the Existing Application wizard to create targets for applications that you built with PowerBuilder 7 or earlier PowerBuilder versions. A source control project in PowerBuilder 7 (or earlier PowerBuilder versions) was associated with a single application.

Beginning with PowerBuilder 8, source control is associated with a workspace that can have multiple targets and applications.

If you keep a copy of your old version of PowerBuilder, you can check out your application object and all other objects from source control to a work PBL. By checking out the objects in the older version of PowerBuilder, you make sure that no one else makes changes to the objects before you migrate them to the current version of PowerBuilder.

Deciding on a directory hierarchy

You should decide on a directory hierarchy before you migrate. PowerBuilder 7 and earlier versions required you to keep all source-controlled files in a single directory. Beginning with PowerBuilder 8, you can create subdirectories to contain each PBL in your library list. Although this is not required, it is useful in that it keeps objects from different PBLs separated in source control subprojects.

You must also decide whether to add a new target to an existing PowerBuilder workspace or to a new workspace that you create specifically for the target. You can then use the Existing Application target wizard to create a new target from the local copies of your registered PBLs (making sure to select all the supporting PBLs for your application on the Set Library Search Path page of the wizard). When you run the wizard, PowerBuilder prompts you to migrate the PBLs you select.

After you have run the wizard and migrated all the source-controlled PBLs, you can define the source control connection profile for the workspace to point to the old source control project if you want to maintain it, or to a new source control project if you do not. You can then check in or add the migrated objects to source control and delete the work PBL containing the older versions of the objects. You do not need a separate work PBL in PowerBuilder 9 or later.

❖ **To migrate a source control project using the Existing Application wizard:**

1   From your old version of PowerBuilder, check out your objects to a work PBL.

2   Decide on a new file hierarchy for the libraries in your application library list.

You can keep all the libraries in the same directory if you want, but it can be advantageous to create separate subdirectories for each library in the list. If you plan to share libraries among different targets, you should structure the directories so that the common libraries are in the local root path of every target that uses them.

3   Create a new workspace in the new version of PowerBuilder, or open an existing workspace in the new version of PowerBuilder.

4   Create a new target using the Existing Application wizard.
In the wizard, point to the PBL with an Application object and add all the helper PBLs to the library search path.

PowerBuilder prompts you to migrate the PBLs.

5   Click Yes to migrate each library in the path.

6   Create a source connection profile for the workspace that points to the old source control project or to a new project.

7   Check in the migrated objects to source control if you are using the old source control project, or add the migrated objects to source control if you are using a new source control project.

8   Delete the work PBL whenever you want.

Importing source control files to a new library

You can use your source control manager to check out all the old PowerBuilder objects to a named directory or folder. If you plan to use the same project to store your migrated objects, you must make sure that the manager locks the files you check out of the source control archive.

You can create a new target using the Application target wizard in a new or existing workspace. The Application wizard lets you select or name a new PBL file to associate with the target it creates. You can use the Target property sheet to list any additional PBLs you want to associate with the target.

You can then import the files that you checked out of source control, distributing them as needed to the libraries you associated with the new target. After importing the files, you can migrate the target by right-clicking it in the System Tree and selecting Migrate from the target pop-up menu. You should also do a full build of the target. After you have migrated and built the target, you can define the connection profile for the workspace to point to the old source control project if you want to maintain it, or to a new source control project if you do not.

CHAPTER 4   **PowerDesigner Integration**

About this chapter   PowerBuilder provides a plug-in interface that you can use to link a
                            PowerBuilder target to a PowerDesigner object-oriented model. This
                            chapter describes how to enable the plug-in and use it to generate or
                            reverse-engineer PowerBuilder targets and objects.

Contents

About PowerDesigner and the PowerDesigner plug-in

About PowerDesigner   PowerDesigner is a data modeling tool that supports several types of
                          models and many programming languages. The following modules are
                          available for data modeling in PowerDesigner:

- Conceptual Data Model (CDM) to model the overall logical structure
  of a data application, independent of any software or data storage
  structure considerations

- Physical Data Model (PDM) to model the overall physical structure
  of a database, taking into account DBMS software or data storage
  structure considerations

- Object Oriented Model (OOM) to model a software system using the
  Unified Modeling Language (UML) notation, which can then be
  generated as objects in a supported object-oriented language such as
  Java or PowerScript

- Business Process Model (BPM) to model the means by which one or more processes are accomplished in operating business practices

- Free Model (FEM) to create any kind of chart diagram, in a context-free environment

About the PowerDesigner plug-in

PowerBuilder 10 includes a plug-in that provides the class diagram functionality of the PowerDesigner OOM directly inside the PowerBuilder design environment.

When the PowerDesigner plug-in is enabled, you can link a PowerBuilder target with a UML class diagram through a process called reverse-engineering. You can make modifications and adjustments to the class diagram and immediately generate those changes back to the objects in the PowerBuilder target library.

With the plug-in, you can also create a UML class diagram from scratch (or open an existing OOM file) and generate a PowerBuilder target using PowerDesigner menus in the PowerBuilder design environment. The plug-in facilitates synchronization between a class diagram and the PowerBuilder target objects that it represents.

Requirement for enabling plug-in
You must install the PowerDesigner 10.1, 11.0, 11.1, or 12.0 Object Oriented Model on your machine before you can use the PowerDesigner plug-in for PowerBuilder.

Enabling the plug-in

You enable the PowerDesigner plug-in through the Plug-in Manager. The Plug-in Manager displays when you select Tools>Plug-in Manager from the PowerBuilder menu. You must select the PowerDesigner item and click OK to enable the PowerDesigner plug-in.

Repository and online Help access

After you enable the plug-in, you can connect to the PowerDesigner Repository. The Repository is a tool for storing versioned documents. It helps you manage development in a multiuser environment. As in PowerDesigner, you can right-click on a model in the tree view of the Local tab and select menu items for Repository operations, such as Consolidate, Update from Repository, or Compare with Repository.

When you open a class diagram painter, the Repository menu is added to the main PowerBuilder menu.

If you installed online Help with PowerDesigner, that Help is also available to the plug-in.

What is an OOM?

An OOM is a structure that provides a close description of a system using Unified Modeling Language (UML) diagrams.

A diagram is a graphical view of a model or package, which displays object symbols. Diagrams allow you to split the display of large models and packages in order to focus on certain objects or subject areas. They can also be used to view the symbols of the same objects, displayed with different kinds of information. With the PowerDesigner plug-in for PowerBuilder, the class diagram is the single type available. A class diagram describes the structure of model elements.

You can create several diagrams in a model or in a package. The diagram window usually appears with a specialized toolbar called the tool palette, from which you can select tools to create objects in your models and packages.

All diagrams have a name and graphical contents. They are projections of the model and they represent it from different angles. They are sorted alphabetically in the PowerDesigner Browser except for the default diagram, which is the first in the list. Each diagram has its own icon to help you quickly identify its type in the Browser. Figure 4-1 shows a class diagram in the PowerDesigner Browser.

With the PowerDesigner plug-in for PowerBuilder, you can open a PowerDesigner OOM file in PowerBuilder, but you see only the class diagrams saved in the file. You can also save a class diagram that you create with the plug-in and open that file directly in PowerDesigner.

Figure 4-1: PowerDesigner Browser displaying a class diagram
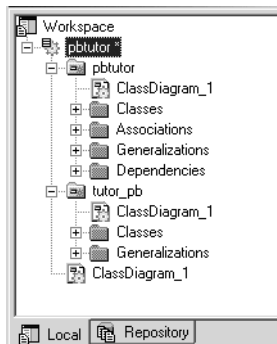
For more information about the OOM, see the *PowerDesigner Object Oriented User's Guide* and the *PowerDesigner Object Oriented Model Getting Started* on the Sybase Web site at http://sybooks.sybase.com/pd.html.

Advantages of an OOM class diagram in PowerBuilder

Purposes of a class diagram

The ability to represent a PowerBuilder application as a class diagram is especially useful for design, documentation, and maintenance purposes. In particular, if you are inheriting responsibility for a legacy application, perhaps one that was created by developers who are no longer in your group, converting it to a class diagram helps you to:

• Understand how the application was developed

• See a graphical display of the relationship between objects

• Improve existing code and regenerate any modified classes as objects in your PowerBuilder application

About the class diagram display

A class diagram displays each PowerBuilder object with a class stereotype and miniature icon, allowing instant recognition of the type of object it represents. PowerBuilder objects that can be represented as classes include windows, user objects, structure, function, and proxy objects. A class diagram illustrates dependencies between classes (that is, PowerBuilder objects), such as those between a menu or DataWindow and a user object, and it displays controls as inner classes.

Each class in a class diagram can be displayed as a box with three parts, corresponding to the class type, its attributes, and its operations. Operations are the equivalent of events on a PowerBuilder object. Figure 4-2 shows a diagram for a window class, w_products.

Figure 4-2: Window class in a plug-in class diagram
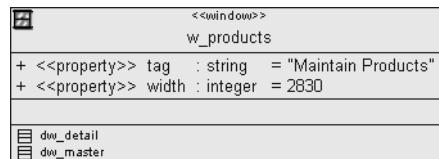
Before you generate a PowerBuilder application from a class diagram, you can check whether or not the application model is well defined. For more information about checking the validity of a class diagram, see "Checking the OOM model" on page 108.

Plug-in options

The Plug-in Options dialog box lets you set automatic synchronization between a PowerBuilder target and a linked OOM. It also lets you reload the linked OOM automatically when you open the PowerBuilder workspace containing the linked target. When you set these options, they are set automatically for all subsequent PowerBuilder sessions. By default, the automatic synchronization option is set to false and the automatic reload option is set to true.

Synchronization is one-way only: changes that you make in the class diagram are not automatically reflected in the linked PowerBuilder object. The PowerBuilder target must be generated again to update it with changes that you make to the OOM. If automatic synchronization is set to true and the linked class diagram is not open in the background when you make a change to a PowerBuilder object, the class diagram opens automatically to show the class that is abstracted from the new or modified PowerBuilder object.

Automatic synchronization is not activated for copy, move, or import operations. It is activated for additions, deletions, and attribute modifications.

Last opened versus last linked OOM file
A PowerDesigner general option lets you open the most recently used OOM file rather than the last linked OOM file for a specific PowerBuilder target. If you set this value for the plug-in, you also set it for PowerDesigner and vice versa. This option is accessible from the Tools>General Options menu of the plug-in, which is enabled whenever a class diagram is open. You can set this option to false to avoid loading an OOM file that has nothing to do with the PowerBuilder targets that you open.

For more information about generating PowerBuilder targets or objects from an OOM, see "Generating PowerBuilder objects" on page 105.

❖ *To set plug-in options*

1   Open a class diagram in the PowerBuilder painter area.

2   Select Tools>Plug-in Options from the PowerBuilder menu or select the Plug-in Options button on the plug-in View toolbar.

    The Plug-in Options dialog box displays.

3   Click in the row for the option you want to change and select the value you want from the drop-down list that displays in the clicked row.

4   Click OK.

Reverse-engineering a PowerBuilder target

Reverse-engineering is the process of examining and recovering data or source code that is then used to build or update an OOM. You can reverse-engineer PowerBuilder objects to obtain a diagram of the class structure of those objects.

The Reverse Engineer dialog box

After you enable the PowerDesigner plug-in, the Reverse Engineer menu item is added to pop-up menus for PowerScript targets. If you select this menu item, the Reverse Engineer dialog box prompts you to select PowerBuilder libraries and objects from the current target. After you click OK in this dialog box, the objects you select are abstracted as classes in a UML class diagram. Figure 4-3 shows the Reverse Engineer dialog box.

Figure 4-3: Reverse Engineer dialog box
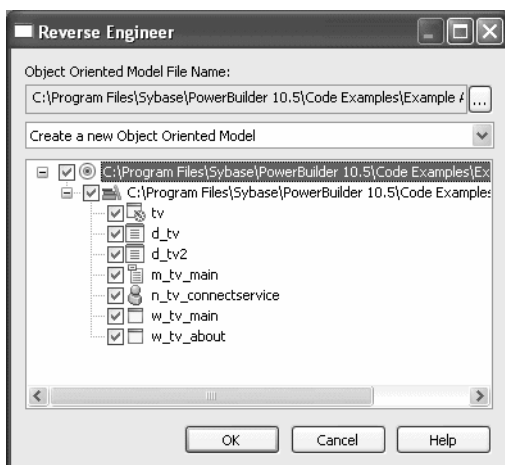
When objects in a PowerBuilder target are reverse-engineered for the first time, the target is linked with a generated class diagram and OOM file. By default, the generated OOM file name has the same name as the target file, but with an *oom* extension. For example, if the PowerBuilder target name is *test.pbt*, then the file *test.oom* is generated in the same directory as the target file.

If an OOM is already linked with the PowerBuilder target, you can determine how changes to PowerBuilder objects will affect the linked OOM when you reverse-engineer the target. The options you can select from are:

• Merge with existing Object Oriented Model (default)

• Replace existing Object Oriented Model

• Replace selected packages and classes

• Replace selected classes

If the target is not currently linked with an OOM, the only selection is:

• Create a new Object Oriented Model

Changing the OOM
file name

Before you create an OOM file from a PowerBuilder target through reverse engineering, you can change the name of the file and its directory, and you can decide whether to overwrite or merge the contents of the OOM with an existing OOM file.

❖ **To change the OOM file name linked to the target**

1 In the Reverse Engineer dialog box for the PowerBuilder target, click the ellipsis button next to the Object Oriented Model File Name field.

The Plug-in Attributes of Target dialog box displays.

2 Modify the path and name of the oompath attribute and click OK.

If you select an existing file name, the Delete Existing Object Oriented Model File check box in the Reverse Engineer dialog box is not grayed.

3 (Optional) Select a reverse-engineering option from the drop-down list.

If an OOM is already linked to the PowerBuilder target and you leave the default selection, new packages, classes, and attributes will be merged with existing packages, classes, and attributes in the linked OOM file. This means that other classes and attributes in the existing OOM file will not be overwritten or deleted when you reverse-engineer the PowerBuilder target.

Other selections allow you to merge the packages while replacing selected classes and their attributes; replace selected packages and classes without replacing nonselected packages and their classes; or replace the entire OOM file.

4 Click OK to reverse-engineer the target.

Using PowerDesigner menu items in PowerBuilder

After you generate and link a PowerBuilder target to an OOM through reverse-engineering, PowerDesigner menu items are enabled in the pop-up menus for objects in that target as described in Table 4-1.

Table 4-1: Items added to PowerBuilder object pop-up menus after plug-in is enabled

Object	Menu item added
Target	Reverse Engineer, Open Class Diagram, Plug-in Attributes
Library	Open Class Diagram
Object	Find in Class Diagram

The Open Class Diagram menu item opens or changes focus to the class diagram for the selected PowerBuilder library. The Find in Class Diagram menu item opens or changes focus to the class diagram for the library containing the selected object, then changes focus to the corresponding class for that object in the class diagram. These menu items cause an error message to display if the target is not linked to an OOM.

You link an OOM to a PowerBuilder target by reverse-engineering the target or by selecting an OOM file in the Plug-in Attributes dialog box. You can open this dialog box from the target pop-up menu or from the Reverse Engineer dialog box.

For more information, see "Reverse-engineering a PowerBuilder target" on page 102.

If a linked OOM is not present in the plug-in PowerDesigner browser, selecting the Open Class Diagram or Find in Class Diagram menu item adds the OOM to the Local tab of the browser in addition to opening the class diagram or finding an object in the diagram.

Class diagram menu commands

Main menu items

Menu items in the OOM plug-in interface also help to integrate PowerDesigner with PowerBuilder. The PowerDesigner menu items do not display in PowerBuilder unless a class diagram painter has focus. The following table lists menu items that display when a class diagram has focus.

Table 4-2: PowerDesigner menu items that display for a class diagram

Plug-in menu	PowerDesigner menu items added
File	The Page Setup, Print Preview, and Print Selection menu items are added for a class diagram. The Print, Save, and Save As menu items are used by PowerBuilder objects or plug-in class diagrams, depending on which painter has the focus.
Edit	All items from the PowerDesigner Edit menu.
View	All items from PowerDesigner View menu except the Browser, Output, and Result List menu items. In the plug-in, the View>Diagram>New Diagram cascading menu items are visible but not enabled.
Model	All items from the PowerDesigner Model menu.
Symbol	All items from the PowerDesigner Symbol menu.
Language	Only the Edit Current Object Language and Generate PowerBuilder menu items.
Repository	All items from the PowerDesigner Repository menu.
Tools	The Check Model, Compare Model, Merge Model, Execute Commands>Edit/Run Script, Display Preferences, General Preferences, and Model Options menu items. Menu items for the PowerBuilder Tools menu are not removed when a class diagram painter has focus.

Menu items in the File, Run, Window, and Help menus remain the same whether a PowerBuilder object painter or a plug-in class diagram has focus.

Generating PowerBuilder objects

You can generate a PowerBuilder target from an OOM by selecting Generate PowerBuilder from the OOM Language menu. The Language menu is visible only if a class diagram is displayed and has focus.

The Generate PowerBuilder menu item opens the PowerBuilder Generation dialog box, displayed in Figure 4-4. This dialog box prompts you to select packages and classes to generate a PowerBuilder target. All PowerBuilder painters must be closed before you click OK to generate the PowerBuilder target from the OOM.

Figure 4-4: PowerBuilder Generation dialog box
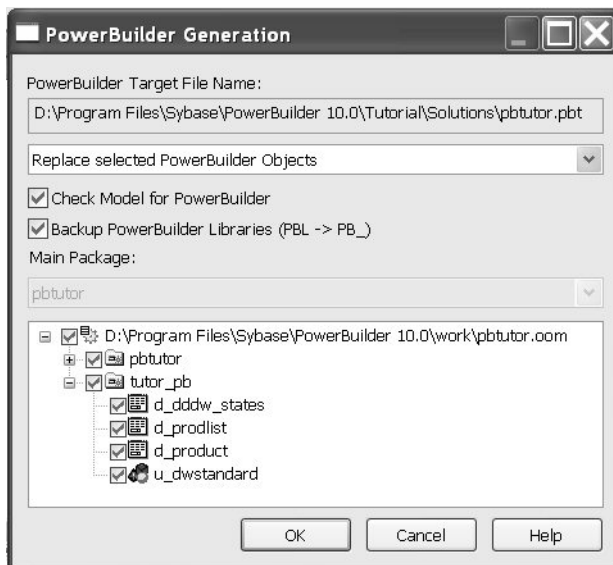
The PowerBuilder Generation dialog box allows you to select how changes to the OOM will affect an existing PowerBuilder target when you regenerate the target. The options are:

• Replace selected PowerBuilder objects (default)

• Replace selected PowerBuilder libraries and objects

• Replace existing PowerBuilder target

If the OOM is not already linked to an existing PowerBuilder target, you have only one option when generating the target:

• Create a new PowerBuilder target

By selecting the Check Model for PowerBuilder check box, you can verify the validity of the model. You can also select a check box to back up existing PowerBuilder libraries before the generation. (This check box is grayed when a PowerBuilder target is not already linked to the current OOM.) Existing PBLs are saved in their original directories with the extension PB_. If you are generating a PowerBuilder target for the first time, you can select which package in the OOM should be used to generate the target application.

The following happens when you generate selected classes or packages:

- Existing PowerBuilder objects are replaced by the code generated from the corresponding class

- Changes to existing PowerBuilder objects are rolled back if code generation is not successful

- The Workspace tab in the PowerBuilder System Tree is automatically refreshed after generation of PowerBuilder objects

- An incremental build is triggered to ensure the PB Target is in good condition

Pop-up menu items   Pop-up menus for the plug-in are the same as in PowerDesigner, except that plug-in classes have additional pop-up menu items.

The pop-up menu for a class in an OOM class diagram has a PowerBuilder menu item with subitems linking the class diagram to a PowerBuilder target:

- PowerBuilder>Open Painter

  This menu item opens the object corresponding to the selected class in its PowerBuilder painter. Double-clicking a class in the linked class diagram achieves the same result.

- PowerBuilder>Find in Workspace

  This menu item places focus on the corresponding PowerBuilder object in the Workspace tab of the System Tree.

The pop-up menu for the Workspace entry in the PowerDesigner Browser view of the plug-in includes the following menu item that is not available in PowerDesigner:

- New >PowerBuilder Object-Oriented Model

  This menu item for an OOM lets you create a new object-oriented model for PowerBuilder. After you create an OOM for PowerBuilder, you can select Generate PowerBuilder from the Language menu to generate a PowerBuilder target.

Checking the OOM model

When you create a new OOM in the plug-in, default properties and operations are added to the application, window, and function classes, making it easier to design and generate PowerBuilder code.

You can use the PowerDesigner Check Model feature to verify that an OOM is valid before generating a PowerBuilder target from the model. If you run Check Model in the plug-in, errors are reported for the following conditions:

- Class names that do not correspond to legal PowerBuilder object names

- More than one Application class in the same Class Diagram

- More than one operation in a Function class, or an operation name that differs from the Function class name

- Structure classes that contain identifiers or operations

- Top-level Class Diagram contains non-Package symbols

- There is no syntax (in the Script or Preview tab of the Class Properties dialog box) for an object to be generated

- Control or binary classes can be generated as inner classes only

- The DataWindow compiler cannot compile the script of a class with the DataWindow stereotype

Plug-in toolbars

PowerBars and PainterBars

The PowerDesigner View toolbar is named PowerBar2 in the plug-in and the PowerDesigner Repository toolbar is named PowerBar3. A PowerBar is always present by default in a PowerBuilder session, even when a plug-in painter is not open. PainterBars display only when the current PowerBuilder painter is a class diagram.

Table 4-3 shows the names of PowerDesigner toolbars supported by the plug-in and their corresponding names in the PowerBuilder user interface.

Table 4-3: Plug-in names for PowerDesigner toolbars

Plug-in toolbar	Corresponding PowerDesigner toolbar
PainterBar1	Standard
PainterBar2	Diagram
PainterBar3	Palette
PainterBar4	PowerBuilder
PowerBar2	View
PowerBar3	Repository

Figure 4-5 shows the Palette toolbar in the plug-in. Other PowerDesigner toolbars are not currently supported by the PowerDesigner plug-in.

Figure 4-5: PowerDesigner palette toolbar in the plug-in

Toolbar differences

When you add classes to the class diagram using the plug-in's PainterBar4 (PowerBuilder) toolbar, default propertes are automatically assigned to the new classes. If you add the same classes using the PowerBuilder toolbar in PowerDesigner, default properties are not assigned.

The plug-in's PowerBar2 (View) toolbar has an additional toolbar button that opens the Plug-in Options dialog box. These options are available for the plug-in, but not for PowerDesigner. You can also access the Plug-in Options dialog box from the Tools>Plug-in Options menu when a class diagram has focus.

P A R T 2

Working with PowerScript Targets

This part describes how to work with PowerScript targets in painters, how to set properties for an application, and how to manage PowerBuilder libraries.

Working with PowerScript Targets

About this chapter

This chapter describes PowerScript targets and the Application object associated with each target.

Contents

About PowerScript targets

A PowerScript target can be used to create:

- **An executable application**   A collection of PowerBuilder windows that perform related activities and that you deliver to users.

  An executable application can be a traditional client/server application that accesses a database server or an application that acts as a client in a distributed application and requests services from a server application.

- **A server component**   A component containing one or more custom class user objects that have methods to provide the required business logic and the characteristics needed for deployment to a distributed application server such as EAServer.

The first step in creating a new application or component is to use a Target wizard, described in Chapter 1, "Working with PowerBuilder."

Depending on the type of target you choose to create, the target can include only an Application object or it can include additional objects. If the target requires connection to a server or a SQL database, the Template Application wizard also creates a Connection object.

The Application object

All PowerScript targets include an Application object. It is a discrete object that is saved in a PowerBuilder library, just like a window, menu, function, or DataWindow object. When a user runs the application, the scripts you write for events are triggered in the Application object.

When you open an Application object, you enter the Application painter.

After you create the new application, you open the Application object and work in the Application painter to define application-level properties (such as which fonts are used by default for text) and application-level behavior (such as what processing should occur when the application begins and ends).

Working in painters

In PowerBuilder, you edit objects such as applications, windows, menus, DataWindow objects, and user objects in painters. In addition to painters that edit objects, other painters such as the Library painter and the Database painter provide you with the ability to work with libraries and databases.

Opening painters

Painters that edit objects

There are several ways to open painters that edit objects:

From here	You can
PowerBar	Click New or Inherit (to create new objects) or Open (to open existing objects)
Library painter	Double-click an object or select Edit from the object's pop-up menu
System Tree	Double-click an object or select Edit from the object's pop-up menu
Browser	Select edit from an object's pop-up menu

Other painters

Most other painters are accessible from the New dialog box. Some are also available on the PowerBar and from the Tools menu.

Select Target for Open

You may see the Select Target for Open dialog box if you use the same PBL in more than one target. When you open an object in a PBL that is used in multiple targets, PowerBuilder needs to set global properties for the specific target you are working on. If you open the object from the Workspace page when the root is not set to the current workspace, PowerBuilder asks you which target you want to open it in. A similar dialog box displays if you select Inherit, Run/Preview, Regenerate, Print, or Search.

Painter summary

The PowerBuilder painters are:

Painter	What you do
Application painter	Specify application-level properties and scripts.
Database painter	Maintain databases, control user access to databases, manipulate data in databases, and create tables.
DataWindow painter	Build intelligent objects called DataWindow objects that present information from the database.
Data Pipeline painter	Transfer data from one data source to another and save a pipeline object for reuse.
Function painter	Build global functions to perform processing specific to your application.
Library painter	Manage libraries, create a new library, and build dynamic libraries.
Menu painter	Build menus to be used in windows.
Project painter	Create executable files, dynamic libraries, components, and proxies.
Query painter	Graphically define and save SQL SELECT statements for reuse with DataWindow objects and pipelines.
SQL Select painter	Graphically define SQL SELECT statements for DataWindow objects and pipelines.
Structure painter	Define global structures (groups of variables) for use in your application.

Painter	What you do
User Object painter (visual)	Build custom visual objects that you can save and use repeatedly in your application. A visual user object is a reusable control or set of controls that has a certain behavior.
User Object painter (nonvisual)	Build custom nonvisual objects that you can save and use repeatedly in your application. A nonvisual user object lets you reuse a set of business rules or other processing that acts as a unit but has no visual component.
Window painter	Build the windows that will be used in the application.

Painter features

Painters that edit objects

Most painters that edit PowerBuilder objects have these features:

Feature	Notes
Painter window with views	See "Views in painters that edit objects" on page 116.
Unlimited undo/redo	Undo and redo apply to all changes.
Drag-and-drop operations	Most drag-and-drop operations change context or copy objects.
To-Do List support	When you are working in a painter, a linked item you add to the To-Do list can take you to the specific location. See "Using the To-Do List" on page 30.
Save needed indicator	When you make a change, PowerBuilder displays an asterisk after the object's name in the painter's Title bar to remind you that the object needs to be saved.

Other painters

Most of the painters that do not edit PowerBuilder objects have views and some drag-and-drop operations.

Views in painters that edit objects

Each painter has a View menu that you use for opening views. The views you can open depend on the painter you are working in. Every painter has a default arrangement of views. You can rearrange these views, choose to show or hide views, and save arrangements that suit your working style. See "Using views in painters" on page 41.

Many views are shared by some painters, but some views are specific to a single painter. For example, the Layout, Properties, and Control List views are shared by the Window, visual User Object, and Application painters, but the Design, Column Specifications, Data, Preview, Export/Import Template for XML, and Export Template for XHTML views are specific to the DataWindow painter. The WYSIWYG Menu and Tree Menu views are specific to the Menu painter.

The following sections describe the views you see in many painters. Views that are specific to a single object type are described in the chapter for that object.

Layout view

The Layout view shows a representation of the object and its controls. It is where you place controls on an object and design the layout and appearance of the object.
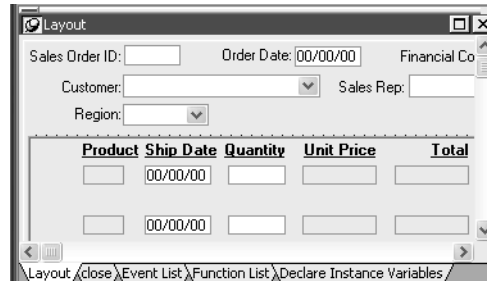
If the Properties view is displayed and you select a control in the Layout view or the Control List view, the properties for that control display in the Properties view. If you select several controls in the Layout view or the Control List view, the properties common to the selected controls display in the Properties view.

Properties view

The Properties view displays properties for the object itself or for the currently selected controls or nonvisual objects in the object. You can see and change the values of properties in this view.
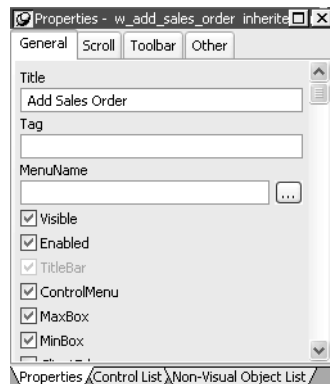
The Properties view dynamically changes when you change selected objects or controls in the Layout, Control List, and Non-Visual Object List views.

If you select several controls in the Layout view or the Control List view, the Properties view says *group selected* in the title bar and displays the properties common to the selected controls.

In the Properties view pop-up menu, you can select Labels On Top or Labels On Left to specify where the labels for the properties display. For help on properties, select Help from the pop-up menu.

If the Properties view is displayed and you select a nonvisual object in the Non-Visual Object List view, the properties for that nonvisual object display in the Properties view. If you select several nonvisual objects in the Non-Visual Object List view, the properties common to the selected nonvisual objects display in the Properties view.

Script view

The Script view is where you edit the scripts for events and functions, define and modify user events and functions, declare variables and external functions, and view the scripts for ancestor objects.

You can open the default script for an object or control by double-clicking it in the System Tree or the Layout, Control List, or NonVisual Object List views, and you can insert the name of an object, control, property, or function in a script by dragging it from the System Tree.

For information about the Script view, see Chapter 7, "Writing Scripts."

Control List view

The Control List view lists the visual controls on the object. You can click the Control column to sort the controls by control name or by hierarchy.

If you select one or more controls in the Control List view, the controls are also selected in the Layout view. Selecting a control changes the Properties view and double-clicking a control changes the Script view.

Event List view

The Event List view displays the full event prototype of both the default and user-defined events mapped to an object. Icons identify whether an event has a script, is a descendent event with a script, or is a descendent event with an ancestor script and a script of its own.
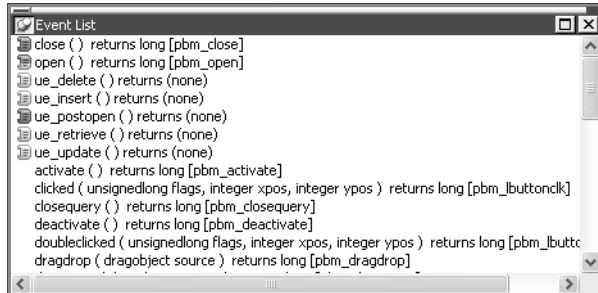
Non-Visual Object List view

The Non-Visual Object List view is a list of nonvisual objects that have been inserted in an Application object, window, or user object of any type. You can sort controls by control name or ancestor.
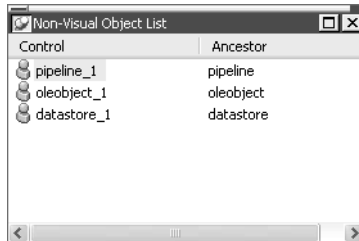
Function List view

The Function List view lists the system-defined functions and the object-level functions you defined for the object. Icons identify whether a function has a script, is a descendant of a function with a script, or is a descendant of a function with an ancestor script and script of its own.
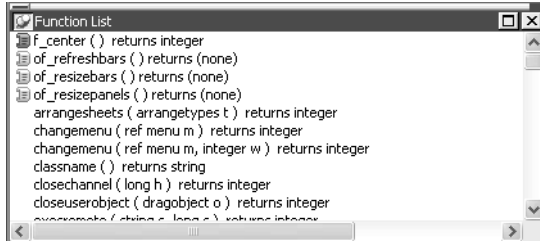
Note that although the half-colored icon identifies the myfunc user-defined function as having both an ancestor script and a script of its own, for a function this means that the function is overridden. This is different from the meaning of a half-colored icon in the Event List view.

Structure List view

The Structure List view lists the object-level structures defined for the object.
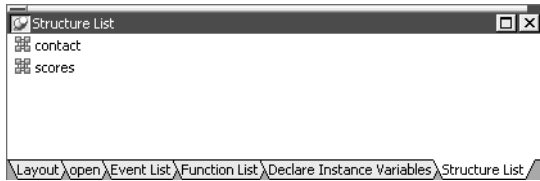
If you double-click a structure in the Structure List view, the structure's definition displays in the Structure view.

Structure view

The Structure view is where you edit the definition of object-level structures in the Window, Menu, and User Object painters.
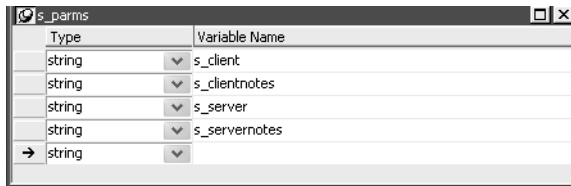
About the Application painter

Views in the
Application painter

The Application painter has several views where you specify properties for your application and how it behaves at start-up. Because the Application painter is an environment for editing a nonvisual object of type application, the Application painter looks like the User Object painter for nonvisual user objects and it has the same views. For details about the views, how you use them, and how they are related, see "Views in painters that edit objects" on page 116.

Application painter
layout

Most of your work in the Application painter is done in the Properties view and the Script view to set application-level properties and code application-level scripts. For information about specifying properties, see "Specifying application properties" on page 121. For information about coding in the Script view, see Chapter 7, "Writing Scripts."

Inserting nonvisual
objects

You can automatically create nonvisual objects in an application by inserting a nonvisual object in the Application object. You do this if you want the services of a nonvisual object available to your application. The nonvisual object you insert can be a custom class or standard class user object.

You insert a nonvisual object in an Application object in the same way you insert one in a user object. For more information, see "Using class user objects" on page 376.

Specifying application properties

You specify application properties in the Application painter's Properties view.

❖ **To specify application properties:**

1   In the Application painter, if the Properties view is not open, select View>Properties from the menu bar.

With the exception of the AppName property, the properties on the General and Toolbar tab pages can be modified in the Properties view and in scripts.

If you need help specifying properties in the Properties View, right-click on the background of the Properties view and select Help from the pop-up menu.

2 Select the General or Toolbar tab page, or, on the General tab page, click the Additional Properties button to display the Application properties dialog box.

The additional properties on the Application properties dialog box can be modified only in this dialog box. They cannot be modified in scripts.

3 Specify the properties:

To specify this	Use this tab page
Display name	General tab page
Application has toolbar text and toolbar tips	Toolbar tab page
Default font for static text as it appears in windows, user objects, and DataWindow objects	Additional Properties (Text Font)
Default font for data retrieved in a DataWindow object	Additional Properties (Column Font)
Default font for column headers in tabular and grid DataWindow objects	Additional Properties (Header Font)
Default font for column labels in freeform DataWindow objects	Additional Properties (Label Font)
Application icon	Additional Properties (Icon)
Global objects for the application	Additional Properties (Variable Types)

These sections have information about how you specify the following application properties in the Application painter:

- "Specifying default text properties" on page 122
- "Specifying an icon" on page 124
- "Specifying default global objects" on page 124

Specifying default text properties

You probably want to establish a standard look for the text in your application. There are four kinds of text whose properties you can specify in the Application painter: text, header, column, and label.

PowerBuilder provides default settings for the font, size, and style for each of these and a default color for text and the background. You can change these settings for an application in the Application painter and can override the settings for a window, user object, or DataWindow object.

Properties set in the Database painter override application properties
If extended attributes have been set for a database column in the Database
painter or Table painter, those font specifications override the fonts specified
in the Application painter.

❖ **To change the text defaults for an application:**

1   In the Properties view, click Additional Properties and select one of the
    following:

    Text Font tab
    Header Font tab
    Column Font tab
    Label Font tab

    The tab you choose displays the current settings for the font, size, style,
    and color. The text in the Sample box illustrates text with the current
    settings.

2   Review the settings and make any necessary changes:

    •   To change the font, select a font from the Font list.

    •   To change the size, select a size from the Size list or type a valid size
        in the list.

    •   To change the style, select a style (Regular, Italic, Bold, or Bold Italic)
        from the Font styles list.

    •   To change font effects, select one or more from the Effects group box
        (Strikeout and Underline).

    •   To change the text color, select a color from the Text Color list. (You
        do not specify colors for data, headings, and labels here; instead, you
        do that in the DataWindow painter.)

    •   To change the background color, select a color from the Background
        list.

    Using custom colors
    When specifying a text color, you can choose a custom color. You can
    define custom colors in several painters, including the Window painter or
    DataWindow painter.

3   When you have made all the changes, click OK.

Specifying an icon

Users can minimize your application during execution. If you specify an icon in the application painter, the icon will display when the application is minimized.

❖ **To associate an icon with an application:**

1 In the Properties view, click Additional Properties and select the Icon tab.

2 Specify a file containing an icon (an ICO file).

The button displays below the Browse button.

3 Click OK to associate the icon with the application.

Specifying default global objects

PowerBuilder provides five built-in global objects that are predefined in all applications.

Global object	Description
SQLCA	Transaction object, used to communicate with your database
SQLDA	DynamicDescriptionArea, used in dynamic SQL
SQLSA	DynamicStagingArea, used in dynamic SQL
Error	Used to report errors during execution
Message	Used to process messages that are not PowerBuilder-defined events and to pass parameters between windows

You can create your own versions of these objects by creating a standard class user object inherited from one of the built-in global objects. You can add instance variables and functions to enhance the behavior of the global objects.

For more information, see Chapter 15, "Working with User Objects."

After you do this, you can tell PowerBuilder that you want to use your version of the object in your application as the default, instead of the built-in version.

❖ **To specify the default global objects:**

1 In the Properties view, click Additional Properties and select the Variable Types tab.

The Variable Types property page displays.

2 Specify the standard class user object you defined in the corresponding field.

For example, if you defined a user object named mytrans that is inherited from the built-in Transaction object, type `mytrans` in the box corresponding to SQLCA.
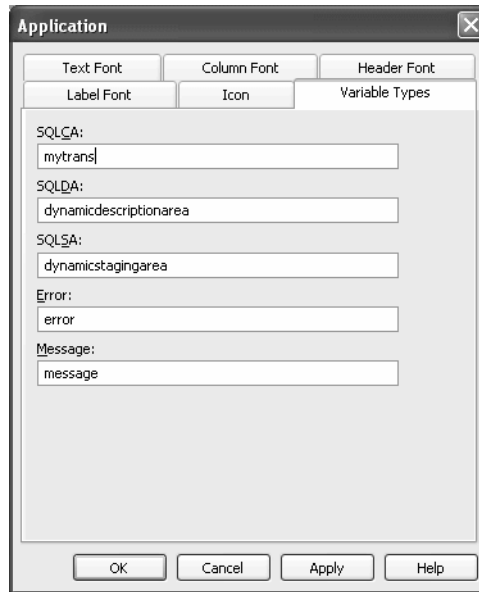
3   Click OK.

When you run your application, it will use the specified standard class user objects as the default objects instead of the built-in global objects.

Writing application-level scripts

When a user runs an application, an Open event is triggered in the Application object. The script you write for the Open event initiates the activity in the application. Typically it sets up the environment and opens the initial window. For a server component, there may be no application-level scripts.

When a user ends an application, a Close event is triggered in the Application object. The script you write for the Close event usually does all the cleanup required, such as closing a database or writing a preferences file.

If there are serious errors during execution, a SystemError event is triggered in the Application object.

Batch applications
If your application performs only batch processing, all processing takes place in the script for the application Open event.

Table 5-1 lists all events that can occur in the Application object (the ConnectionBegin and ConnectionEnd events are obsolete). The only event that requires a script is Open.

Table 5-1: Events in the Application object

Event	Occurs when
Open	The user starts the application.
Close	The user closes the application. Typically, you write a script for this event that shuts everything down (such as closing the database connection and writing out a preferences file).
SystemError	A serious error occurs during execution (such as trying to open a nonexistent window). If there is no script for this event, PowerBuilder displays a message box with the PowerBuilder error number and message text. If there is a script, PowerBuilder executes the script.
	For more about error handling, see "Handling errors at runtime" on page 884.
Idle	The Idle PowerScript function has been called and the specified number of seconds has elapsed with no mouse or keyboard activity.

Setting application properties in scripts

The Application object has several properties that specify application-level properties. For example, the property ToolbarText specifies whether text displays on toolbars in an MDI application.

You can reference these properties in *any* script in the application using this syntax:

```
AppName.property
```

For example, to specify that text displays on toolbars in the Test application, code this in a script:

```
Test.ToolbarText = TRUE
```

If the script is in the Application object itself, you do not need to qualify the property name with the application name.

Application name cannot be changed
The name of an application is one of the Application object's properties, but
you cannot change it.

For a complete list of the properties of the Application object, see *Objects and
Controls*.

Specifying the target's library search path

The objects you create in painters are stored in PowerBuilder libraries (PBLs).
You can use objects from one library or multiple libraries in a PowerScript
target. You define each library the target uses in the library search path.

PowerBuilder uses the search path to find referenced objects during execution.
When a new object is referenced, PowerBuilder looks through the libraries in
the order in which they are specified in the library search path until it finds the
object.

On the Library List tab page of the Target Properties dialog box, you can
modify the libraries associated with the current target.

❖ **To specify the target's library search path:**

1   In the Workspace tab of the System Tree, right-click on the target
    containing your application and select Properties from the pop-up menu.
    The Target Properties dialog box displays.

2   Select the Library List tab page.

    The libraries currently included in the library search path are displayed in
    the list.

3   Do one of the following:

    •   Enter the name of each library you want to include in the Library
        Search Path list, separating the names with semicolons.

    •   Use the Browse button to include other libraries in your search path.

    You must specify libraries using an absolute path. To change the order of
    libraries in the search path, use the pop-up menu to copy, cut, and paste
    libraries.

Make sure the order is correct
When you select multiple libraries from the Select Library dialog box
using Shift+Click or Ctrl+Click, the first library you select appears last in
the Library Search Path list and will be the last library searched.

To delete a library from the search path, select the library in the list and use
the pop-up menu or press Delete.

---

4 Click OK.

PowerBuilder updates the search path for the target.

Where PowerBuilder maintains the library search path
PowerBuilder stores your target's library search path in the target (*.pbt*) file in
a line beginning with `LibList`; for example:

```
LibList "pbtutor.pbl;tutor_pb.pbl";
```

Looking at an application's structure

If you are working with an application that references one or more objects in
an application-level script, you can look at the application's structure in the
Browser.

❖ **To display the application's structure:**

1 Click the Browser button on the PowerBar.

2 In the Browser, select the Uses tab page and select Expand All from the
Application object's pop-up menu.

PowerBuilder expands the display to show all the global objects that are
referenced in a script for the Application object. You can expand the
display further as needed.

Which objects are displayed

The Browser's Uses tab page shows global objects that are referenced in your application. It shows the same types of objects that you can see in the Library painter. It does not show entities that are defined within other objects, such as controls and object-level functions.
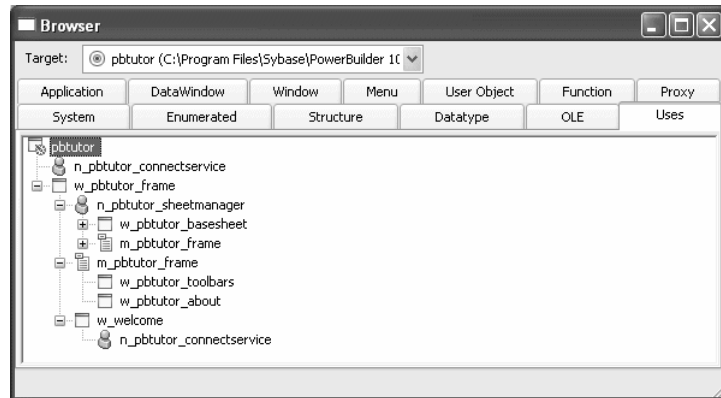
Which references are displayed

The Browser displays the following types of references when the Application object is expanded.

Objects referenced in painters

These are examples of objects referenced in painters:

•   If a menu is associated with a window in the Window painter, the menu displays when the window is expanded.

•   If a DataWindow object is associated with a DataWindow control in the Window painter, the DataWindow object displays when the window is expanded.

•   If a window contains a custom user object that includes another user object, the custom user object displays when the window is expanded, and the other user object displays when the custom user object is expanded.

Objects directly referenced in scripts

These are examples of objects referenced in scripts:

•   If a window script contains the following statement, w_continue displays when the window is expanded:

```
Open(w_continue)
```

Which referenced windows display in the Browser
Windows are considered referenced only when they are opened from
within a script. A use of another window's property or instance variable
will not cause the Browser to display the other window as a reference of
the window containing the script.

• If a menu item script refers to the global function f_calc, f_calc displays
  when the menu is expanded:

```
f_calc(EnteredValue)
```

• If a window uses a pop-up menu through the following statements, m_new
  displays when the window is expanded:

```
m_new    mymenu
mymenu = create m_new
mymenu.m_file.PopMenu(PointerX(), PointerY())
```

Which references are not displayed

The Browser does not display the following types of references.

Objects referenced
through instance
variables or properties

These are examples of objects referenced through instance variables or
properties:

If w_go has this statement (and no other statement referencing w_emp),
w_emp does not display as a reference for w_go:

```
w_emp.Title = "Managers"
```

Objects referenced
dynamically through
string variables

These are examples of objects referenced dynamically through string
variables:

• If a window script has the following statements, the window w_go does not
  display when the window is expanded. The window w_go is named only
  in a string:

```
window    mywin
string    winname = "w_go"
Open(mywin,winname)
```

• If the DataWindow object d_emp is associated with a DataWindow control
  dynamically through the following statement, d_emp does not display
  when the window containing the DataWindow control is expanded:

```
dw_info.DataObject = "d_emp"
```

Working with objects

In PowerScript targets, you can:

- Create new objects

- Create new objects using inheritance

- Open existing objects

- Run or preview objects

After you create or open an object, the object displays in its painter and you work on it there.

Creating new objects

To create new objects, you use the New dialog box.

❖ **To create a new object:**

1   Do one of the following:

- Click the New button in the PowerBar.

- Select File>New from the menu bar.

- On the Workspace tab of the System Tree, right-click on a workspace or target name and select New from the pop-up menu.

2   In the New dialog box, select the appropriate tab page for the object you want to create.

You use icons on the PB Object tab page for creating new user objects, windows, menus, structures, and functions.

3   Select an icon and click OK.

Creating new objects using inheritance

One of the most powerful features of PowerBuilder is inheritance. With inheritance, you can create a new window, user object, or menu (a descendent object) from an existing object (the ancestor object).

❖ **To create a new object by inheriting it from an existing object:**

1 Click the Inherit button in the PowerBar, or select File>Inherit from the menu bar.

2 In the Inherit From Object dialog box, select the object type (menu, user object, or window) from the Object Type drop-down list. Then select the target as well as the library or libraries you want to look in. Finally, select the object from which you want to inherit the new object.
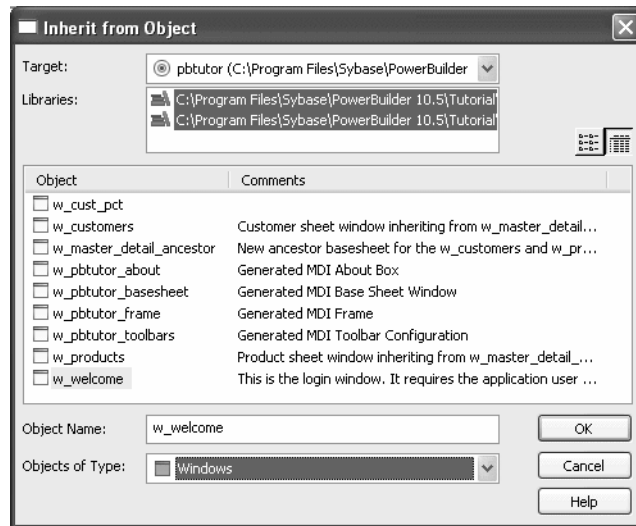
Displaying objects from many libraries
To find an object more easily, you can select more than one library in the Libraries list. Use Ctrl+Click to select additional libraries and Shift+Click to select a range.

3 Click OK.

The new object, which is a descendant of the object you chose to inherit from, opens in the appropriate painter.

For more information about inheritance, see Chapter 13, "Understanding Inheritance."

Naming conventions

As you use PowerBuilder to develop your application, you create many different components that require names. These components include objects such as windows and menus, controls that go into your windows, and variables for your event and function scripts.

You should devise a set of naming conventions and follow them throughout your project. When you are working in a team, this is critical to enforcing consistency and enabling others to understand your code. This section provides tables of common naming conventions. PowerBuilder does not require you to use these conventions, but they are followed in many PowerBuilder books and examples.

All indentifiers in PowerBuilder can be up to 40 characters long. The first few characters are typically used to specify a prefix that identifies the kind of object or variable, followed by an underscore character, followed by a string of characters that uniquely describes this particular object or variable.

Object naming conventions

Table 5-2 shows common prefixes for objects that you create in PowerBuilder.

Table 5-2: Common prefixes for objects

Prefix	Description
w_	Window
m_	Menu
d_	DataWindow
pipe_	Data Pipeline
q_	Query
n_ or n_*standardobject*_	Standard class user object, where *standardobject* represents the type of object; for example, n_trans
n_ or n_cst	Custom class user object
u_ or u_*standardobject*_	Standard visual user object, where *standardobject* represents the type of object; for example, u_cb
u_	Custom visual user object
f_	Global function
of_	Object-level function
s_	Global structure
str_	Object-level structure
ue_	User event

Variable naming
conventions

The prefix for variables typically combines a letter that represents the scope of the variable and a letter or letters that represent its datatype. Table 5-3 lists the prefixes used to indicate a variable's scope. Table 5-4 lists the prefixes for standard datatypes, such as integer or string.

The variable might also be a PowerBuilder object or control. Table 5-5 lists prefixes for some common PowerBuilder system objects. For controls, you can use the standard prefix that PowerBuilder uses when you add a control to a window or visual user object. To see these prefixes, open the Window painter, select Design>Options, and look at the Prefixes 1 and Prefixes 2 pages.

Table 5-3: Prefixes that indicate the scope of variables

Prefix	Description
a	Argument to an event or function
g	Global variable
i	Instance variable
l	Local variable
s	Shared variable

Table 5-4: Prefixes for standard datatypes

Prefix	Description
a	Any
blb	Blob
b	Boolean
ch	Character
d	Date
dtm	DateTime
dc	Decimal
dbl	Double
e	Enumerated
i	Integer
l	Long
r	Real
s	String
tm	Time
ui	UnsignedInteger
ul	UnsignedLong

Table 5-5: Prefixes for selected PowerBuilder system objects

Prefix	Description
ds	DataStore
dw	DataWindow
dwc	DataWindowChild
dwo	DWobject
env	Environment
err	Error
gr	Graph
inet	Inet
ir	InternetResult
lvi	ListViewItem
mfd	MailFileDescription
mm	MailMessage
mr	MailRecipient
ms	MailSession
msg	Message
nvo	NonVisualObject
tr	Transaction
tvi	TreeViewItem

Opening existing objects

You can open existing objects through the Open dialog box or directly from the System Tree.

❖ **To open existing objects:**

1   Click the Open button in the PowerBar or select File>Open from the menu bar.

When using the System Tree
To open an existing object directly from the System Tree, either double-click on the object name or select Edit from the pop-up menu.

2   In the Open dialog box, select the object type from the Object Type drop-down list. Then select the target as well as the library or libraries you want to look in. Finally select the object you want to open.
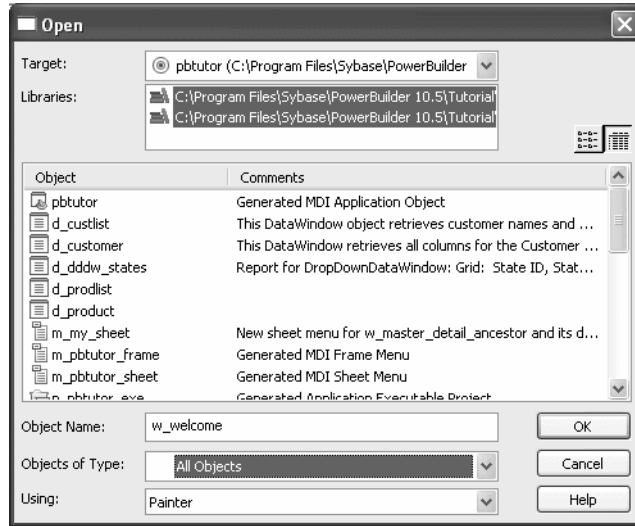
Displaying objects from many libraries
To find an object more easily, you can select more than one library in the Libraries list. Use Ctrl+Click to select additional libraries and Shift+Click to select a range.

3   Click OK.

The object opens in the appropriate painter.

Accessing recently opened objects

You can quickly open recently opened objects by selecting File>Recent Objects from the menu bar. The Recent Objects list includes the eight most recently opened objects by default, but you can include up to 36 objects on the list.

❖   **To modify the number of recent objects:**

1   Select Tools>System Options from the menu bar.

2   On the General page of the System Options dialog box, modify the number for the recent objects list.

Running or previewing objects

To run a window or preview a DataWindow object, you use the Run dialog box.

Using the System Tree
Instead of using the Run dialog box, you can right-click the object in the
System Tree and select Run/Preview from the pop-up menu.

❖ **To run or preview an object:**

1   Do one of the following:

•   Click the Run/Preview Object button in the PowerBar.

•   Select File>Run/Preview from the menu bar.

2   In the Run dialog box, select the object type from the Object Type drop-
down list.

3   Select the target as well as the library or libraries you want to look in.

4   Select the object you want to run or preview and click OK.

The object runs or is previewed.

For more specific information on running a window, see "Running a window"
on page 245. For information on using the DataWindow painter's Preview
view, see Chapter 18, "Defining DataWindow Objects."

Using the Source editor

You can use the Source editor to edit the source of most PowerScript objects
directly instead of making changes to an object in a painter. You cannot edit the
source of project or proxy objects. The Source editor makes it unnecessary to
export an object in order to edit it and then import it, as you do with the file
editor.

Caution: back up your objects

Although the Source editor provides a quick way to make global changes, you should use it with caution, and you must be familiar with the syntax and semantics of PowerScript source code before using the Source editor to change it.

Changes you make to an object's source code using the Source editor take effect *immediately* when you save the object, *before the code is validated.* If an error message displays in the Output window, you must fix the problem in the Source editor before you close the editor. If you do not, you will not be able to open the object in a painter.

Technical Support is not able to provide support if changes you make in the Source editor render an object unusable. For this reason, Sybase strongly recommends that you make backup copies of your PBLs or objects before you edit objects in the Source editor.

You can open an object in the Source editor in one of several ways:

• Use the Open dialog box

• Select the Edit Source menu item in the System Tree or Library painter

• Select the Edit Source menu item in the Output window for a line that contains an error

Unlike the file editor, the Source editor cannot be opened independently. It can only be used in conjunction with an object defined within a PowerScript target in the current workspace. You cannot open an object in the Source editor that is already open in a painter.

When you export an object and view the exported file in the file editor, a PBExportHeader line displays at the beginning of the file. If you saved the object with a comment from the object's painter, a PBExportComment also displays. The Source editor display is identical to the display in the file editor except that the PBExport lines are not present in the Source editor.

For more information on exporting objects, see "Exporting and importing entries" on page 163.

CHAPTER 6   **Working with Libraries**

About this chapter   PowerBuilder stores all the PowerScript objects you create in libraries. This chapter describes how to work with your libraries.

Contents

About libraries

Whenever you save an object, such as a window or menu, in a painter, PowerBuilder stores the object in a library (a PBL file). Similarly, whenever you open an object in a painter, PowerBuilder retrieves the object from the library.

Assigning libraries   PowerScript targets can use as many libraries as you want. Libraries can be on your own PC or workstation or on a server. When you create a target, you specify which libraries it uses. You can also change the library search path for a target at any time during development.

For information about specifying the library search path, see "Specifying the target's library search path" on page 127.

How the information is saved

Every object is saved in two parts in a library:

- **Source form** This is a syntactic representation of the object, including the script code.

- **Object form** This is a binary representation of the object, similar to an object file in the C and C++ languages. PowerBuilder compiles an object automatically every time you save it.

Using libraries

It is hard to predict the needs of a particular application, so the organization of a target's libraries generally evolves over the development cycle. PowerBuilder lets you reorganize your libraries easily at any time.

About library size

For small applications, you might use only one library, but for larger applications, you should split the application into different libraries.

There are no limits to how large libraries can be, but for performance and convenience, you should follow these guidelines:

- **Number of objects** It is a good idea not to have more than 50 or 60 objects saved in a library. This is strictly for your convenience; the number of objects does not affect performance. If you have many objects in a library, list boxes that list library objects become unmanageable and the System Tree and Library painter become more difficult to use.

- **Balance** Managing a large number of libraries with only a few objects makes the library search path too long and can slow performance by forcing PowerBuilder to look through many libraries to find an object. Try to maintain a balance between the size and number of libraries.

Organizing libraries

You can organize your libraries any way you want. For example, you might put all objects of one type in their own library, or divide your target into subsystems and place each subsystem in its own library.

Sharing objects with others

PowerBuilder provides basic source control using the PBNative check in/check out utility. PBNative allows you to lock the current version of PowerBuilder objects and prevents others from checking out these objects and modifying them while you are working on them.

The project administrator must design a directory hierarchy for the project's workspace. The administrator might create a separate subdirectory for each target in the workspace, or for each PBL in the workspace. After the administrator sets up the project and registers every object in the workspace, individual developers copy a template workspace to their own computers, open the workspace, and connect to source control.

PowerBuilder also provides a direct connection to external SCC-compliant source control systems.

For more about using PBNative and other source control systems, see "Using a source control system with PowerBuilder" on page 66.

Opening the Library painter

❖ **To open the Library painter:**

- Click the Library button in the PowerBar or select Tools>Library Painter.

What you can do in the Library painter

In the Library painter, you can:

- Create a new library

- Create new objects in targets in your current workspace

- Copy, move, and delete objects in any library

- Open objects in libraries that are on a library list in the current Workspace to edit them in the appropriate painters

- Migrate, rebuild, and regenerate libraries in the current Workspace

- Control modifications to library objects by using check-out and check-in or use version control systems

- Create a runtime library that includes objects in the current library and related resource objects

What you cannot do in the Library painter

You cannot migrate or open objects in PowerBuilder libraries that are *not* on the library list. You also cannot rename a library.

About the Library painter

Views in the Library painter

The Library painter has two views, the Tree view and the List view, that can display all the files in your file system, not just PowerBuilder objects. You use the painter primarily for displaying and working with workspaces, targets, library files (PBLs), and the objects they contain.

The Tree and List views are available from the View menu. By default, the Library painter displays one Tree view (on the left) and one List view (on the right). When the Library painter opens, both the Tree view and the List view display all the drives on your computer, including mapped network drives.
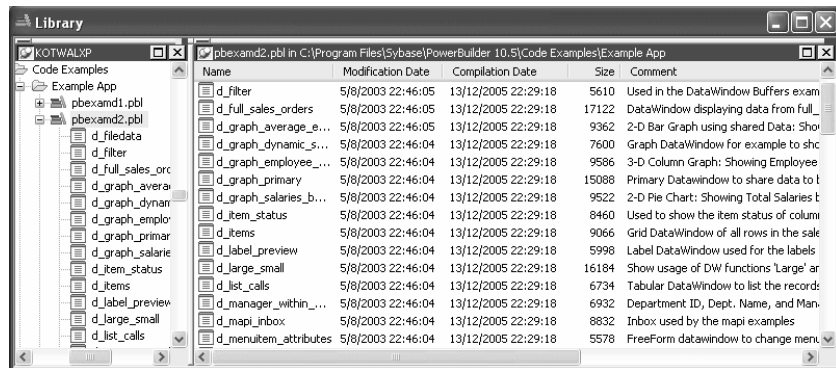
Using the System Tree

The Workspace tab page in the System Tree works like a Tree view in the Library painter. You can perform most tasks in either the System Tree or the Library painter Tree view, using the pop-up menu in the System Tree and the pop-up menu, PainterBar, or menu bar in the Library painter. When you have the System Tree and a Library painter open at the same time, remember that the PainterBar and menu bar apply only to the Library painter.

Each time you click the Library painter button on the PowerBar, PowerBuilder opens a new instance of the Library painter. One advantage of using the System Tree is that there is only one instance of it that you can display or hide by clicking the System Tree button on the PowerBar.

About the Tree view

The Tree view in the Library painter displays the drives and folders on the computer and the workspaces, targets, libraries, objects, and files they contain. You can expand drives, folders, and libraries to display their contents.

About the List view

The List view in the Library painter displays the contents of a selected drive, folder, or library and has columns with headers that provide extra information. For libraries, the comment column displays any comment associated with the library. For objects in libraries, the columns display the object name, modification date, size, and any comment associated with the object. You can resize columns by moving the splitter bar between columns, and you can sort a column's contents by clicking the column header.

About sorting the Name column
When you click the Name column header repeatedly to sort, the sort happens in four ways: by object type and then name, in both ascending and descending order, and by object name, in both ascending and descending order. You might not easily observe the four ways of sorting if all objects of the same type have names that begin with the same character or set of characters.

Displaying items in the Tree view and the List view

Most of the time, you select a library in the Tree view and display the objects in that library in the List view, but at any time, you can set a new root or move back and forward in the history of your actions in the List view and the Tree view to display libraries or other items. For more information, see "Setting the root" on page 152 and "Moving back, forward, and up one level" on page 153.

Using custom layouts

You might find that having more than one Tree view or List view makes your work easier. Using the View menu, you can display as many Tree views and List views as you need.

The following screen shows the Library painter with one Tree view and three List views.
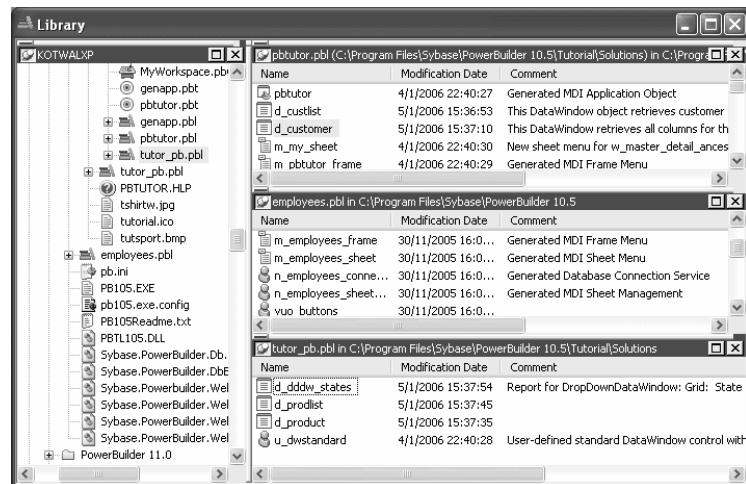
You can filter the objects in each of the List views so that one List view shows menus, another windows, and another user objects. For information about filtering objects in a view, see "Filtering the display of objects" on page 147.

To get this layout in the Library painter, use the View menu to display two more List views and then manipulate the views to fit this layout. For information about opening and closing views, manipulating views, returning to the default view layout, or saving your favorite layouts, see "Using views in painters" on page 41.

View synchronization

Tree and List views are synchronized with each other. When you are using more than one Tree view or List view, changes you make in one type of view are reflected in the last view you touched of the other type. For example, when an item is selected in a Tree view, the contents of that item display in the List view that you last touched. When you display new contents in a List view by double-clicking an item, that item is selected in the Tree view you last touched (if it can be done without resetting the root).

Each List view in the previous screen displays the contents of a different library because three libraries were dragged from the Tree view and dropped in different List views. For information about drag and drop, see "Displaying libraries and objects" on page 144.

Working with libraries

The Library painter is designed for working with PowerBuilder libraries.

Displaying libraries and objects

What you see in the views

In the Tree view, you can expand items and see the folders, libraries, or objects they contain. The List view displays the contents of a selection in the Tree view.

❖ **To expand or collapse an item in the Tree view:**

•   Double-click the item.

    If the item contains libraries or objects, they display in the List view.

❖ **To display the contents of an item in the List view:**

•   Select the item in the Tree view or double-click the item in the List view.

Using drag and drop
to expand items

You can drag and drop items to expand them and see the contents.

If you drag an item from a Tree view or List view to a List view, the List view sets the item as the root and displays its contents.

If you drag an item from a Tree view or List view to a Tree view, the Tree view expands to display the dragged item.

For example, you can drag a library from the Tree view and drop it in the List view to quickly display the objects the library contains in the List view. If you are using one Tree view and multiple List views, you can drag a specific library from the Tree view to each List view so each List view contains the contents of a specific library.

For information about using drag and drop to copy or move items, see "Copying, moving, and deleting objects" on page 150.

Using the pop-up menu

Like other painters, the Library painter has a pop-up menu that provides options that apply to the selected item in the Tree view or the List view. For example, from a library's pop-up menu, you can delete, optimize, or search the library, print the directory, specify the objects that display in the library, and import objects into it.

The actions available from an object's pop-up menu depend on the object type. For PowerBuilder objects that you can work with in painters, you can edit the object in a painter or in the Source editor, copy, move, or delete the object, export it to a text file, search it, regenerate it, or send it to a printer. You can also preview and inherit from some objects. For most of these actions, the object must be in a library in your current workspace.

Actions available from the pop-up menus are also available on the Entry menu on the menu bar.

Controlling columns that display in the List view

You can control whether to display the last modification date, compilation date, size, SCC version number, and comments (if a comment was created when an object or library was created) in the List view.

The version number column in the Library painter list view remains blank if the source control system for your workspace does not support the PowerBuilder extension to the SCC API. If your source control system supports this extension and if you are connected to source control, you can override the SCC version number of a PowerScript object in the local copy directory through the property sheet for that object.

For more information about listing the SCC version number and overriding it through the PowerBuilder interface, see "Extension to the SCC API" on page 65.

❖ **To control the display of columns in the List view:**

1   Select Design>Options from the menu bar.

2   On the General tab page, select or clear these display items: Modification Date, Compilation Date, Sizes, SCC Version Number, and Comments.

Selecting objects

In the List view, you can select one or more libraries or objects to act on.

❖ **To select multiple entries:**

•   In the List view, use Ctrl+Click (for individual entries) and Shift+Click (for a group of entries).

❖ **To select all entries:**

•   In the List view, select an object and click the Select All button on the PainterBar.

Filtering the display of objects

You can change what objects display in expanded libraries.

Settings are remembered
PowerBuilder records your preferences in the Library section of the PowerBuilder initialization file so that the next time you open the Library painter, the same information is displayed.

Specifying which objects display in all libraries

In the Tree and List views, the Library painter displays all objects in libraries that you expand, as well as targets, workspaces, folders, and files. You can specify that the Library painter display only specific kinds of objects and/or objects whose names match a specific pattern. For example, you can limit the display to only DataWindow objects, or limit the display to windows that begin with w_emp.

❖ **To restrict which objects are displayed:**

1   Select Design>Options from the menu bar and select the Include tab.

2   Specify the display criteria:

*   To limit the display to entries that contain specific text in their names, enter the text in the Name box. You can use the wildcard characters question mark (?) and asterisk (*) in the string. The ? represents one character; an * represents any string of characters. The default is all entries of the selected types.

*   To limit the display to specific entry types, clear the check boxes for the entry types that you do not want to display. The default is all entries.

3   Click OK.

The Options dialog box closes.

4   In the Tree view, expand libraries or select a library to display the objects that meet the criteria.

Overriding the choices you made for a specific view

In either the Tree view or the List view, you can override your choice of objects that display in all libraries by selecting a library, displaying the library's pop-up menu, and then clearing or selecting items on the list of objects.

Creating and deleting libraries

A library is created automatically when you create a new PowerScript target, but you can create as many libraries as you need for your project in the Library painter.

❖ **To create a library:**

1   Click the Create button or select Entry>Library>Create from the menu bar.

The Create Library dialog box displays showing the current directory and listing the libraries it contains.

2   Enter the name of the library you are creating and specify the directory in which you want to store it.

The file is given the extension *.PBL*.

3   Click Save.

The library properties dialog box displays.

4   Enter any comments you want to associate with the library.

Adding comments to describe the purpose of a library is important if you are working on a large project with other developers.

5   Click OK.

PowerBuilder creates the library.

❖ **To delete a library:**

1   In either the Tree view or the List view, select the library you want to delete.

2   Select Entry>Delete from the menu bar or select Delete from the pop-up menu.

Restriction
You cannot delete a library that is in the current target's library search path.

The Delete Library dialog box displays showing the library you selected.

3   Click Yes to delete the library.

The library and all its entries are deleted from the file system.

Creating and deleting libraries at runtime

You can use the LibraryCreate and LibraryDelete functions in scripts to create and delete libraries. For information about these functions, see the *PowerScript Reference*.

Filtering the display of libraries and folders

In either the Tree view or the List view, you can control what displays when you expand a drive or folder. An expanded drive or folder can display folders, workspaces, targets, files, and libraries.

❖ **To control display of the contents of drives and folders:**

• In either the Tree or List view, select a drive or folder, select Show from the pop-up menu, and select or clear items from the cascading menu.

Working in the current library

In PowerBuilder, the current library is the library that contains the object most recently opened or edited. That library becomes the default for Open and Inherit. If you click the Open or Inherit button in the PowerBar, the current library is the one selected in the Libraries list.

You can display the current library in the Library painter.

❖ **To display objects in the current library:**

1   Click in the Tree view or the List view.

2   Click the Display Most Recent Object button on the PainterBar or select Most Recent Object from the View menu.

The library that contains the object you opened or edited last displays in the view you selected with the object highlighted.

Opening and previewing objects

You can open and preview objects in the current workspace.

Opening PowerBuilder objects

PowerBuilder objects, such as windows and menus, are opened only if they are in a PBL in the current workspace.

❖ **To open an object:**

• In either the Tree view or the List view, double-click the object, or select Edit from the object's pop-up menu.

PowerBuilder takes you to the painter for that object and opens the object. You can work on the object and save it as you work. When you close it, you return to the Library painter.

Opening other objects

The Library painter allows you to open most of the different file types it displays. When you double-click on an object, PowerBuilder attempts to open it using the following algorithm:

1   PowerBuilder determines if the object can be opened in the File editor. For example, files with the extensions *.txt*, *.ini*, and *.sr** open in the File editor.

2   PowerBuilder determines if the object can be opened in a painter or HTML editor.

3   PowerBuilder checks to see if the object is associated with a program in the *HKEY_CLASSES_ROOT* section of the Windows registry and, if so, launches the application.

Previewing PowerBuilder objects

You can run windows and preview DataWindow objects from the Library painter.

❖ **To preview an object in the Library painter:**

• Select Run/Preview from the object's pop-up menu.

Copying, moving, and deleting objects

As the needs of your target change, you can rearrange the objects in libraries. You can copy and move objects between libraries or delete objects that you no longer need.

❖ **To copy objects using drag and drop:**

1 In the Tree view or the List view, select the objects you want to copy.

2 Drag the objects to a library in either view. If the contents of a library are displaying in the List view, you can drop it there.

PowerBuilder copies the objects. If an object with the same name already exists, PowerBuilder prompts you and if you allow it, replaces it with the copied object.

❖ **To move objects using drag and drop:**

1 In the Tree view or the List view, select the objects you want to move.

2 Press and hold Shift and drag the objects to a library in either view. If the contents of a library are displaying in the List view, you can drop it there.

PowerBuilder moves the objects and deletes them from the source library. If an object with the same name already exists, PowerBuilder prompts you and if you allow it, replaces it with the moved object.

❖ **To copy or move objects using a button or menu item:**

1 Select the objects you want to copy or move to another library.

2 Do one of the following:

• Click the Copy button or the Move button.

• Select Copy or Move from the pop-up menu.

• Select Entry>Library Item>Copy or Entry>Library Item>Move from the menu bar.

The Select Library dialog box displays.

3 Select the library to which you want to copy or move the objects and click OK.

❖ **To delete objects:**

1 Select the objects you want to delete.

2 Do one of the following:

• Click the Delete button.

• Select Delete from the pop-up menu.

• Select Entry>Delete from the menu bar.

You are asked to confirm the first deletion.

Being asked for confirmation
By default, PowerBuilder asks you to confirm each deletion. If you do not want to have to confirm deletions, select Design>Options to open the Options dialog box for the Library painter and clear the Confirm on Delete check box in the General tab page.

PowerBuilder records this preference as the DeletePrompt variable in the Library section of the PowerBuilder initialization file.

3   Click Yes to delete the entry or Yes To All to delete all entries. Click No to skip the current entry and go on to the next selected entry.

Setting the root

In either the Tree view or the List view, you can set the root location of the view.

❖   **To set the root of the current view:**

1   In either view, select View>Set Root from the menu bar or select Set Root from the pop-up menu to display the Set Root dialog box.
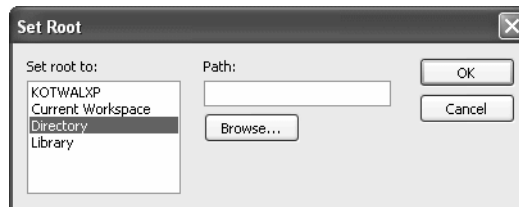
2   If you want the root to be a directory or library, type the path or browse to the path.

Setting the root to the current workspace

In the System Tree, the default root is the current workspace. If you prefer to work in the Library painter, you may find it convenient to set the root to the current workspace. Using the current workspace as your root is particularly helpful if you are using many libraries in various locations, because they are all displayed in the same tree.

Moving back, forward, and up one level

You can also set a new root by moving back to where you were before, moving forward to where you just were, or for the List view, moving up a level.

❖ **To move back, forward, or up one level:**

•   Do one of the following:

  •   Select View>Back, View>Forward, or View>Up One Level from the menu bar.

  •   Select Back, Forward, or Up One Level from the pop-up menu.

  The name of the location you are moving back to or forward to is appended to Back and Forward.

Modifying comments

You can use comments to document your objects and libraries. For example, you might use comments to describe how a window is used, specify the differences between descendent objects, or identify a PowerBuilder library.

You can associate comments with an object or library when you first save it in a painter and add or modify comments in the System Tree or Library painter. If you want to modify comments for a set of objects, you can do so quickly in the List view.

❖ **To modify comments for multiple objects:**

1   In the List view, select the objects you want.

2   Select Entry>Properties from the menu bar or select Properties from the pop-up menu.

  PowerBuilder displays the Properties dialog box. The information that displays is for one of the objects you selected. You can change existing comments, or, if there are no comments, you can enter new descriptive text.

3   Click OK when you have finished editing comments for this object.

  If you do not want to change the comments for an object, click OK. The next object displays.

4   Enter comments and click OK for each object until you have finished.

If you want to stop working on comments before you finish with the objects you selected, click Cancel. The comments you have entered until the most recent OK are retained and display in the List view.

❖   **To modify comments for a library:**

1   Select the library you want.

2   Click the Properties button or select Library from the pop-up menu.

3   Add or modify the comments.

Searching targets, libraries, and objects

Global search of
targets

You can search a target to locate where a specified text string is used. For example, you could search for:

•   All scripts that use the SetTransObject function

•   All windows that contain the CommandButton cb_exit (all controls contained in a window are listed in the window definition's source form in the library so they can be searched for as text)

•   All DataWindow objects accessing the Employee table in the database

---

Working with targets
To see the pop-up menu that lets you perform operations on a target, such as search, build, and migrate, you must set the root of the System Tree or the view in the Library painter to the current workspace.

Searching selected
libraries and objects

You can also select a library or one or more PowerBuilder objects to search. The following procedure applies whatever the scope of your search is.

❖   **To search a target, library, or object for a text string:**

1   Select the target, library, or objects you want to search.

You can select multiple objects in the List view using Shift+Click and Ctrl+Click.

2   Select Search from the pop-up menu or the PainterBar.

The Search Library Entries dialog box displays.
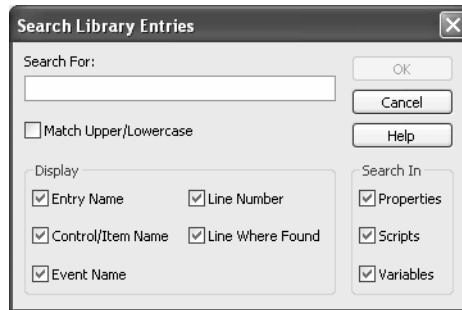
3   Enter the string you want to locate (the search string) in the Search For
    box.

    The string can be all or part of a word or phrase used in a property, script,
    or variable. You cannot use wildcards in the search string.

4   In the Display group box, select the information you want to display in the
    results of the search.

5   In the Search In group box, select the parts of the object that you want
    PowerBuilder to inspect: properties, scripts, and/or variables.

6   Click OK.

    PowerBuilder searches the libraries for matching entries. When the search
    is complete, PowerBuilder displays the matching entries in the Output
    window.

For example, the following screen displays the results of a search for the string
`garbagecollect`:
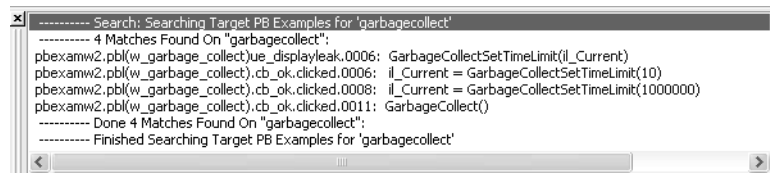
From the Output window, you can:

•   Jump to the painter in which an entry was created

    To do this, double-click the entry or select it and then select Edit from the
    pop-up menu.

•   Print the contents of the window

•   Copy the search results to a text file

Optimizing libraries

You should optimize your libraries regularly. Optimizing removes gaps in libraries and defragments the storage of objects, thus improving performance.

Optimizing affects only layout on disk; it does not affect the contents of the objects. Objects are not recompiled when you optimize a library.

Once a week
For the best performance, you should optimize libraries you are actively working on about once a week.

❖ **To optimize a library:**

1   In either Tree view or List view, choose the library you want to optimize.

2   Select Entry>Library>Optimize from the menu bar or select Optimize from the library's pop-up menu.

    PowerBuilder reorganizes the library structure to optimize object and data storage and index locations. Note that PowerBuilder does not change the modification date for the library entries. PowerBuilder saves the unoptimized version as a backup file in the same directory.

    The optimized file is created with the default permissions for the drive where it is stored. On some systems new files are not shareable by default. If you see "save of object failed" or "link error messages after optimizing," check the permissions assigned to the PBL.

    If you do not want a backup file
    If you do not want to save a backup copy of the library, clear the Save Optimized Backups check box in the Library painter's Design>Options dialog box. If you clear this option, the new setting will remain in effect until you change it.

Regenerating library entries

Occasionally you may need to update library entries by regenerating, rebuilding, or migrating them. For example:

• When you modify an ancestor object, you can *regenerate* descendants so they pick up the revisions to their ancestor.

• When you make extensive changes to a target, you can *rebuild* entire libraries so objects are regenerated sequentially based on interdependence.

• When you upgrade to a new version of PowerBuilder, you need to *migrate* your targets.

When you regenerate an entry, PowerBuilder recompiles the source form stored in the library and replaces the existing compiled form with the recompiled form. You can regenerate entries in the Library painter or by selecting regenerate from the object's pop-up menu in the System Tree.

You can also regenerate and rebuild from a command line. For more information, see Appendix B, "The OrcaScript Language."

❖ **To regenerate library entries in the Library painter:**

1   Select the entries you want to regenerate.

2   Click the Regenerate button or select Entry>Library Item>Regenerate from the menu bar.

PowerBuilder uses the source to regenerate the library entry and replaces the current compiled object with the regenerated object. The compilation date and size are updated.

Regenerating descendants

You can use the Browser to easily regenerate all descendants of a changed ancestor object.

❖ **To regenerate descendants:**

1   Click the Browser button in the PowerBar.

The Browser displays.

2   Select the tab for the object type you want to regenerate.

For example, if you want to regenerate all descendants of window w_frame, click the Window tab.

3   Select the ancestor object and choose Show Hierarchy from its pop-up menu.

The Regenerate item displays on the pop-up menu.
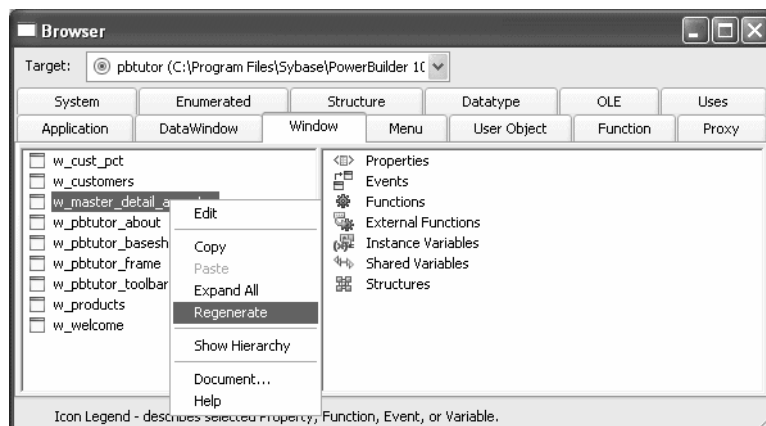
4   Click the Regenerate item.

PowerBuilder regenerates all descendants of the selected ancestor.

For more about the Browser, see "Browsing the class hierarchy" on page 308.

Regenerate limitations
If you regenerate a group of objects, PowerBuilder will regenerate them in the order in which they appear in the library, which might cause an error if an object is generated before its ancestor. For this reason, you should use a full or incremental build to update more than one object at a time.

Rebuilding workspaces and targets

When you make modifications to a target and need to update one or more libraries, you should use a rebuild option to update all the library objects in the correct sequence.

Working with targets
To see the pop-up menu that lets you perform operations on a target such as search, build, and migrate, you must set the root of the System Tree or the view in the Library painter to the current workspace.

There are two methods to use when you rebuild a workspace or target:

- **Incremental rebuild**   Updates all the objects and libraries that reference objects that have been changed since the last time you built the workspace or target

- **Full rebuild**   Updates all the objects and libraries in your workspace or target

❖ **To rebuild a workspace:**

- Do one of the following:

  - Select Incremental Build Workspace or Full Build Workspace from the PowerBar.

  - Select the Workspace in the System Tree or Library painter and select Incremental Build or Full Build from the pop-up menu.

❖ **To rebuild a target:**

- Do one of the following:

  - Select the target in the Library painter and select Entry>Target>Incremental Build or Entry>Target>Full Build from the menu bar.

  - Select the target in the System Tree or Library painter and select Incremental Build or Full Build from the pop-up menu.

Migrating targets

When you upgrade to a new version of PowerBuilder, your existing targets need to be migrated to the new version. Typically, when you open a workspace that contains targets that need to be migrated, or add a target that needs to be migrated to your workspace, PowerBuilder prompts you to migrate the targets. However, there are some situations when you need to migrate a target manually. For example, if you add a library that has not been migrated to a target's library list, you will not be able to open objects in that library until the target has been migrated.

You cannot migrate a target that is not in your current workspace and you must set the root of the System Tree or the view in the Library painter to the current workspace.

Before you migrate

There are some steps you should take before you migrate a target:

- Use the Migration Assistant to check for obsolete syntax or the use of reserved words in your code

- Check the release notes for migration issues

- Make backup copies of the target and libraries

- Make sure that the libraries you will migrate are writable

Always back up your PBLs before migrating
Make sure you make a copy of your PBLs before migrating. After migration, you cannot open them in an earlier version of PowerBuilder.

The Migration Assistant is available on the Tool page of the New dialog box. For help using the Migration Assistant, click the Help (?) button in the upper-right corner of the window and click the field you need help with, or click the field and press F1. If the Migration Assistant finds obsolete code, you can fix it in an earlier version of PowerBuilder to avoid errors when you migrate to the current version.

PowerBuilder libraries and migration

PowerBuilder libraries (PBLs) contain a header, source code for the objects in the PBL, and binary code. There are two differences between PowerBuilder 10 and later PBLs and PBLs developed in earlier versions of PowerBuilder:

- The source code in PowerBuilder 10 and later PBLs is encoded in Unicode (UTF-16LE, where LE stands for little endian) instead of DBCS (versions 7, 8, and 9) or ANSI (version 6 and earlier).

- The format of the header lets PowerBuilder determine whether it uses Unicode encoding. The header format for PowerBuilder 10 is the same as that used for PUL files in PowerBuilder 6.5 and for PKL files in PocketBuilder. These files do not need to be converted to Unicode when they are migrated to PowerBuilder 10 or later.

When PBLs are migrated

Before opening a PBL, PowerBuilder checks its header to determine whether or not it uses Unicode encoding. PBLs are not converted to Unicode unless you specifically request that they be migrated.

You cannot expand the icon for a PBL from PowerBuilder 9 or earlier in the Library painter. To examine its contents, you must migrate it to PowerBuilder 10 or later.

When you attempt to open a workspace that contains targets from a previous release in PowerBuilder, the Targets to be Migrated dialog box displays. You can migrate targets from this dialog box, or clear the No Prompting check box to open the Migrate Application dialog box.

PowerBuilder dynamic libraries
If you plan to reference a PowerBuilder dynamic library (PBD) that was encoded in ANSI formatting (for example, if it was created in PowerBuilder 9 or earlier), you must regenerate the PBD to use Unicode formatting. Dynamic libraries that you create in PowerBuilder 10 or later use Unicode formatting exclusively.

For information on creating PBDs, see "Creating runtime libraries" on page 166.

The Migrate Application dialog box

The Migrate Application dialog box lists each PBL that will be migrated and lets you choose the type of messages that display during the migration process.
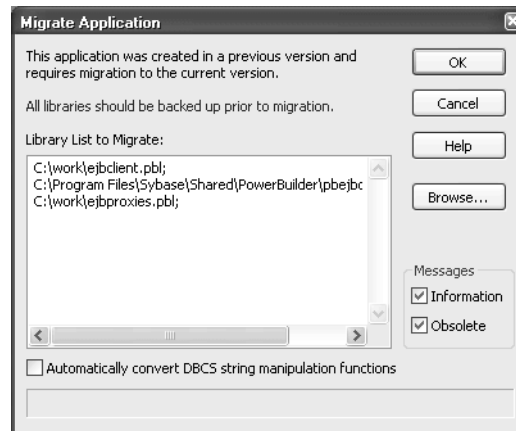
If you click OK, each PBL is first migrated to the new version of PowerBuilder. Then PowerBuilder converts the source code from DBCS to Unicode, performs a full build, and saves the source code back to the same file. It might also make some changes to your scripts. These changes display in informational messages in the Output window and are written to a log file for each PBL so that you can examine the changes later. Recommended changes are also written to the log file.

These two lines from a log file indicate that the FromAnsi function is obsolete and was replaced with the String function, and that an encoding parameter should be added to an existing instance of the String function:

```
2006/01/27 08:20:11test.pbl(w_main).cb_1.clicked.4:
Information C0205: Function 'FromAnsi' is replaced with
function 'String'.
2006/01/27 08:20:11test.pbl(w_main).cb_2.clicked.4:
Information C0206: Append extra argument
'EncodingAnsi!' to function 'String' for backward
compatibility.
```

The log file has the same name as the PBL with the string *_mig* appended and the extension *.log* and is created in the same directory as the PBL. If no changes are made, PowerBuilder creates an empty log file. If the PBL is migrated more than once, output is appended to the existing file.

PowerBuilder makes the following changes:

- The FromUnicode function is replaced with the String function and the second argument EncodingUTF16LE! is added

- The ToUnicode function is replaced with the Blob function and the second argument EncodingUTF16LE! is added

- The FromAnsi function is replaced with the String function and the second argument EncodingAnsi! is added

- The ToAnsi function is replaced with the Blob function and the second argument EncodingAnsi! is added

- An Alias For clause with the following format is appended to declarations of external functions that take strings, chars, or structures as arguments or return any of these datatypes:

    ALIAS FOR "*functionname*;ansi"

    If the declaration already has an Alias For clause, only the string ;ansi is appended.

DBCS users only
If you select the Automatically Convert DBCS String Manipulation Functions check box, PowerBuilder automatically makes appropriate conversions to scripts in PowerBuilder 9 applications. For example, if you used the LenW function, it is converted to Len, and if you used the Len function, it is converted to LenA. The changes are written to the Output window and the log file. This box should be selected only in DBCS environments.

Adding PBLs to a
PowerBuilder target

When you add PBLs from a previous release to a PowerBuilder target's library list, the PBLs display in the System Tree. The PBLs are not migrated when you add them to the library list. Their contents do not display because they have not yet been converted. To display their contents, you must migrate the target.

You can migrate a target from the Workspace tab of the System Tree by selecting Migrate from the pop-up menu for the target. You can also migrate targets in the Library painter if they are in your current workspace.

❖ **To migrate a target in the Library painter:**

1 Select the target you want to migrate and select Entry>Target>Migrate from the menu bar.

   The Migrate Application dialog box displays.

2 Select OK to migrate all objects and libraries in the target's path to the current version.

Exporting and importing entries

You can export object definitions to text files. The text files contain all the information that defines the objects. The files are virtually identical syntactically to the source forms that are stored in libraries for all objects.

You may want to export object definitions in the following situations:

• You want to store the objects as text files.

• You want to move objects to another computer as text files.

Later you can import the files back into PowerBuilder for storage in a library.

Caution
The primary use of the Export feature is exporting source code, modifying the source. You can use the Source editor to modify the source code of an object directly, but modifying source in an ASCII text file is not recommended for most users. See "Using the Source editor" on page 137.

❖ **To export entries to text files:**

1   Select the Library entries you want to export.

    You can select multiple entries in the List view.

2   Do one of the following:

    •   Select Export from the pop-up menu.

    •   Click the Export button on the PainterBar.

    •   Select Entry>Library Item>Export from the menu bar.

    The Export Library Entry dialog box displays, showing the name of the first entry selected for export in the File Name box and the name of the current directory. The current directory is the target's directory or the last directory you selected for saving exported entries or saving a file using the file editor.

    PowerBuilder appends the file extension *.srx*, where *x* represents the object type.

3   Specify the file name and directory for the export file. Do not change the file extension from the one that PowerBuilder appended.

4   Select the encoding for the exported file.

    The HEXASCII export format is used for source-controlled files. Unicode strings are represented by hexadecimal/ASCII strings in the exported file, which has the letters HA at the beginning of the header to identify it as a file that might contain such strings. You cannot import HEXASCII files into a previous version of PowerBuilder.

5   Click OK.

    PowerBuilder converts the entry to text, stores it with the specified name, then displays the next entry you selected for export.

    If a file already exists with the same name, PowerBuilder displays a message asking whether you want to replace the file. If you say no, you can change the name of the file and then export it, skip the file, or cancel the export of the current file and any selected files that have not been exported.

6   Repeat steps 3 through 5 until you have processed all the selected entries.

    If the Library painter is set to display files, you can see the saved files and double-click them to open them in the File editor.

❖ **To import text files to library entries:**

1   In the System Tree or Library painter, select the library into which you want to import an object.

2   Select Import from the pop-up menu, or, in the Library painter only, click the Import button on the PainterBar.

The Select Import Files dialog box displays, showing the current directory and a list of files with the extension *.sr** in that directory. The current directory is the target's directory or the last directory you selected for saving exported entries or saving a file using the file editor.
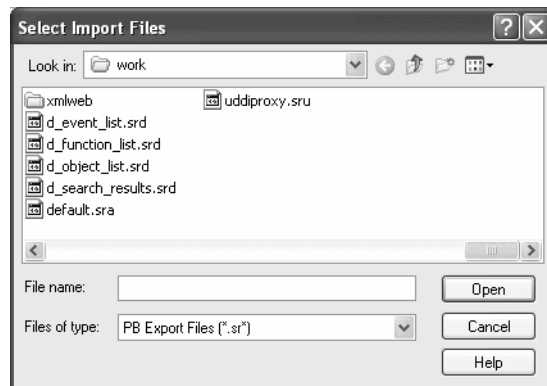
3   Select the files you want to import. Use Shift+Click or Ctrl+Click to select multiple files.

4   Click Open.

PowerBuilder converts the specified text files to PowerBuilder format, regenerates (recompiles) the objects, stores the entries in the specified library, and updates the entries' timestamps.

If a library entry with the same name already exists, PowerBuilder replaces it with the imported entry.

Caution
When you import an entry with the same name as an existing entry, the old entry is deleted before the import takes place. If an import fails, the old object will already be deleted.

Creating runtime libraries

If you want your deployed target to use dynamic runtime libraries, you can create them in the Library painter.

For information about using runtime libraries, see Chapter 34, "Creating Executables and Components." That chapter also describes the Project painter, which you can use to create dynamic runtime libraries automatically.

❖ **To create a runtime library:**

1 Select the library you want to use to build a runtime library.

2 Select Entry>Library>Build Runtime Library from the menu bar, or select Build Runtime Library from the library's pop-up menu.

The Build Runtime Library dialog box displays, listing the name of the selected library.
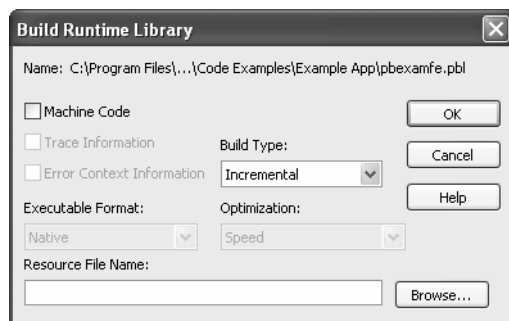
3 If any of the objects in the source library use resources, specify a PowerBuilder resource file in the Resource File Name box (see "Including additional resources" next).

4 Select other options as appropriate.

Most options are available only if you select Machine Code, which creates a DLL file. The default is Pcode, which creates a PBD file. For more information about build options, see "Executable application project options" on page 927.

5 Click OK.

PowerBuilder closes the dialog box and creates a runtime library with the same name as the selected library and the extension *.dll* or *.pbd*.

Including additional resources

When building a runtime library, PowerBuilder does not inspect the objects; it simply removes the source form of the objects. Therefore, if any of the objects in the library use resources (pictures, icons, and pointers)—either specified in a painter or assigned dynamically in a script—and you do not want to provide these resources separately, you must list the resources in a PowerBuilder resource file (PBR file). Doing so enables PowerBuilder to include the resources in the runtime library when it builds it.

For more on resource files, see "Using PowerBuilder resource files" on page 932.

After you have defined the resource file, specify it in the Resource File Name box to include the named resources in the runtime library.

Creating reports on library contents

You can generate three types of reports from the Library painter:

*   The search results report
*   Library entry reports
*   The library directory report

The search results report contains the matching-entries information that PowerBuilder displays after it completes a search, described in "Searching targets, libraries, and objects" on page 154. The other two types of reports are described in this section.

Creating library entry reports

Library entry reports provide information about selected entries in the current target. You can use these reports to get printed documentation about the objects you have created in your target.

❖ **To create library entry reports:**

1 Select the library entries you want information about in the List view.

2 Select Entry>Library Item>Print from the menu bar, or select Print from the pop-up menu.

The Print Options dialog box displays.
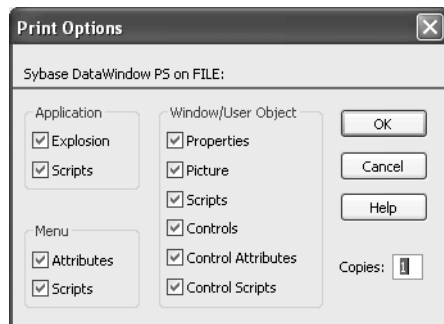
3 If you have selected the Application object or one or more menus, windows, or user objects to report on, select the information you want printed for each of these object types.

For example, if you want all properties for selected windows to appear in the report, make sure the Properties box is checked in the Window/User Object group box.

The settings are saved
PowerBuilder records these settings in the Library section of the PowerBuilder initialization file.

4 Click OK.

PowerBuilder generates the selected reports and sends them to the printer specified in Printer Setup in the File menu.

Creating the library directory report

The library directory report lists all entries in a selected library in your workspace, showing the following information for all objects in the library, ordered by object type:

- Name of object

- Modification date and time

- Size (of compiled object)

- Comments

❖ **To create the library directory report:**

1 Select the library that you want the report for.

The library must be in your current workspace.

2 Select Entry>Library>Print Directory from the menu bar, or select Print Directory from the pop-up menu.

PowerBuilder sends the library directory report to the printer specified under File>Printer Setup in the menu bar.

P A R T   3

Coding Fundamentals

This part describes how to code your application. It covers the basics of the PowerScript language, how to use the Script view, and how to create functions, structures, and user events to make your code more powerful and easier to maintain.

Writing Scripts

About this chapter PowerBuilder applications are event driven. You specify the processing that takes place when an event occurs by writing a script. This chapter describes how to use the Script view to write scripts using the PowerScript language.

Contents

Topic	Page

For more information For complete information about the PowerScript language, see the *PowerScript Reference*.

About the Script view

You use the Script view to code functions and events, define your own functions and events, and declare variables and external functions.

Script views are part of the default layout in the Application, Window, User Object, Menu, and Function painters. In Application, Window, and User Object painters, the initial layout has one Script view that displays the default event script for the object and a second Script view set up for declaring instance variables. You can open as many Script views as you need, or perform all coding tasks in a single Script view.

Title bar

The Script view's title bar shows the name and return type of the current event or function, as well as the name of the current control for events and the argument list for functions. If the Script view is being used to declare variables or functions, the titlebar shows the type of declaration.

Drop-down lists

There are three drop-down lists at the top of the Script view:
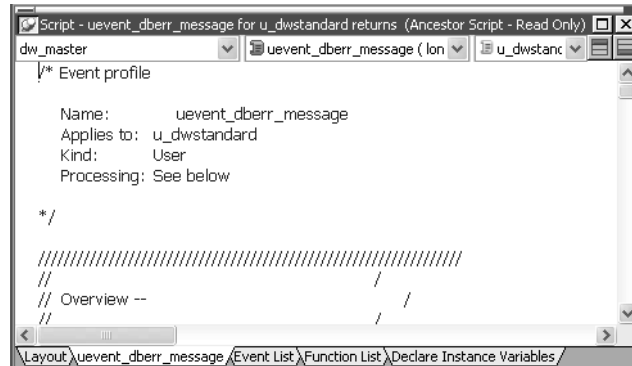
In the first list, you can select the object, control, or menu item for which you want to write a script. You can also select Functions to edit function scripts or Declare to declare variables and external functions.

The second list lets you select the event or function you want to edit or the kind of declaration you want to make. A script icon next to an event name indicates there is a script for that event, and the icon's appearance tells you more about the script:

Table 7-1: Script icons in the Script view

If there is a script	The script icon displays
For the current object or control	With text
In an ancestor object or control only	In color
In an ancestor as well as in the object or control you are working with	Half in color

The same script icons display in the Event List view.

The third list is available in descendent objects. It lists the current object and all its ancestors so that you can view scripts in the ancestor objects.

Toggle buttons for Prototype and Error windows

A Prototype window displays at the top of the Script view when you define a new function or event. An Error window displays at the bottom of the view when there are compilation errors. You can toggle the display of these windows with the two toggle buttons to the right of the lists.

For more information about the Prototype window, see Chapter 8, "Working with User-Defined Functions," and Chapter 9, "Working with User Events."

Opening Script views

If there is no open Script view, selecting a menu or PainterBar item that requires a Script view opens one automatically. If you want to edit more than one script at a time, you can open additional Script views from the View menu.

❖ **To open a new Script view:**

• Select View>Script from the menu bar.

❖ **To edit a script for a control:**

• Double-click a scriptable control, or select Script from the PainterBar or a pop-up menu.

The Script view shows the default script for the control. If the Script view is in a stacked pane and is hidden, it pops to the front. If there is no open Script view, PowerBuilder creates a new one.

> **Using drag and drop**
> If a Script view is visible, you can drag a control from the Control list view to the Script view to edit a script for the control.

❖ **To edit a script for a function or event:**

• Double-click an item in the Event list or Function list views, or select the function or event from the second drop-down list in an open Script view.

The Script view shows the script for the selected event or function. If the Script view is in a tabbed pane and is hidden, it pops to the front. If there is no open Script view, PowerBuilder creates a new one.

Modifying Script view properties

The Script view automatically:

- Color-codes scripts to identify data types, system-level functions, flow-of-control statements, comments, and literals

- Indents the script based on flow-of-control statements

You can modify these and other properties.

Some properties are shared
Some properties you specify for the Script view also affect the file editor, Source editor, Debugger, and the Interactive SQL and Activity Log views in the Database painter.

❖ **To specify Script view properties:**

1 Select Design>Options to display the Options dialog box for the painter.

The Options dialog box includes four tab pages that affect the Script view: Script, Font, Coloring, and AutoScript.

2 Choose the tab appropriate to the property you want to specify:

To specify	Choose this tab
Tab size, automatic indenting, whether dashes are allowed in identifiers, and which compiler and database messages display	Script
Font family, size, and color for the Script view	Font
Text and background coloring for PowerScript syntax elements	Coloring
Whether AutoScript is enabled and what kind of assistance it provides	AutoScript

Editing scripts

You can perform standard editing tasks in the Script view using the Edit menu, the pop-up menu in the Script view, or the PainterBars. There are shortcuts for many editing actions.

Setting up shortcuts
In a painter with a Script view, select Tools>Keyboard Shortcuts. Expand the Edit menu to view existing shortcuts and set up your own shortcuts.

Limiting size of scripts

There is an internal limit on the size of compiled Pcode on any script. Pcode is the interpreted language into which scripts are compiled. A script that exceeds this limit can be compiled successfully, but the error "Maximum script size exceeded" displays when you attempt to save the script. Note that the amount of Pcode generated from a given script is not directly proportional to the number of lines of code, so you might encounter this error in a script with 1200 lines of code, but not in a script with 1500 lines of code. To avoid receiving this error, move code to functions that you post or trigger in the event script.

Printing scripts

You can print a description of the object you are editing, including all its scripts, by selecting File>Print from the menu bar. To print a specific script, select File>Print Script.

Pasting information into scripts

You can paste the names of variables, functions, objects, controls, and other items directly into your scripts. (You can also use AutoScript. See "Using AutoScript" on page 182.) If what you paste includes commented text that you need to replace, such as function arguments or clauses in a statement, you can use Edit>Go To>Next Marker to move your cursor to the next commented item in the template.

Table 7-2: Pasting information into scripts

To paste	Use
PowerBuilder objects and their properties, functions, and events	System Tree
Properties, data types, functions, structures, variables, and objects	Browser
Contents of clipboard	Edit>Paste
Contents of Clipboard window	Drag and drop
Objects, controls, arguments, and global and instance variables	Paste buttons on PainterBar *or* Edit>Paste Special
PowerScript statements	Paste Statement button *or* Edit>Paste Special>Statement
SQL statements	Paste SQL button *or* Edit>Paste Special>SQL
Built-in, user-defined, and external functions	Paste Function button *or* Edit>Paste Special>Function
Contents of text files	Edit>Paste Special>From File

Undoing a paste
If you paste information into your script by mistake, click the Undo button or select Edit>Undo from the menu bar.

Some of these techniques are explained in the sections that follow.

Using the System Tree

To paste the name of a PowerBuilder object or of any of its properties, functions, or events, select the item you want to paste on the Workspace tab of the System Tree and drag it into your script.

Using the Browser

You can use the Browser to paste the name of any property, data type, function, structure, variable, or object in the application.

Most tab pages in the Browser have two panes:
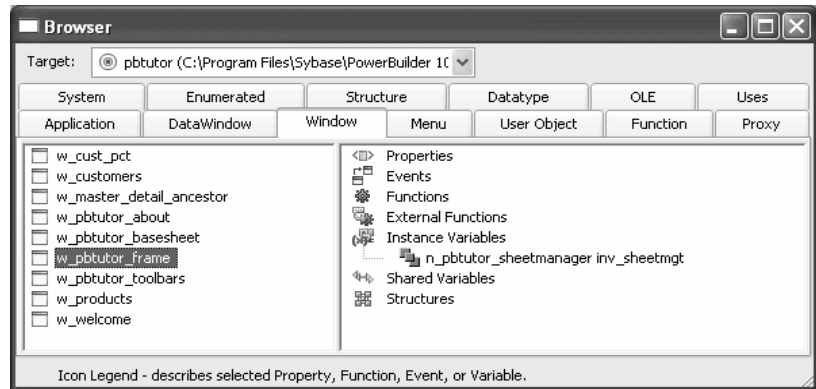
The left pane displays a single type of object, such as a window or menu. The right pane displays the properties, events, functions, external functions, instance variables, shared variables, and structures associated with the object.

Getting context-sensitive Help in the Browser
To get context-sensitive Help for an object, control, or function, select Help from its pop-up menu.

❖ **To use the Browser to paste information into the Script view:**

1 Click the Browser button in the PowerBar, or select Tools>Browser.

2 Select the PowerScript target you want to browse.

3 Select the appropriate tab and then select the object in the left pane.

4 Select the category of information you want to display by expanding the appropriate folder in the right pane.

5 Select the information and click Copy.

6 In the Script view, move the cursor where you want to paste the information and select any text you want to replace with the pasting.

7 Select Paste from the pop-up menu.

   PowerBuilder displays the information at the insertion point in the script, replacing any selected text.

For information about using the Browser to paste OLE object information into a script, see *Application Techniques*.

Pasting statements

You can paste a template for all basic forms of the following PowerScript statements:

- IF...THEN

- DO...LOOP

- FOR...NEXT

- CHOOSE CASE

- TRY...CATCH... FINALLY

When you paste these statements into a script, prototype values display in the syntax to indicate conditions or actions. By default, the statements are pasted in lowercase. To paste statements in uppercase, add the following line to the [PB] section of the *PB.INI* file:

```
PasteLowercase=0
```

❖ **To paste a PowerScript statement into the script:**

1 Place the insertion point where you want to paste the statement in the script.

2 Select the Paste Statement button from the PainterBar, or select Edit>Paste Special>Statement from the menu bar.

3 Select the statement you want to paste from the cascading menu.

The statement prototype displays at the insertion point in the script.

4 Replace the prototype values with the conditions you want to test and the actions you want to take based on the test results.

For more about PowerScript statements, see the *PowerScript Reference*.

Pasting SQL

You can paste a SQL statement into your script instead of typing the statement.

❖ **To paste a SQL statement:**

1 Place the insertion point where you want to paste the SQL statement in the script.

2 Click the Paste SQL button in the PainterBar, or select Edit>Paste Special>SQL from the menu bar.

3   Select the type of statement you want to insert from the cascading menu by double-clicking the appropriate button.

The appropriate dialog box displays so that you can create the SQL statement.

4   Create the statement, then return to the Script view.

The statement displays at the insertion point in the workspace.

For more about embedding SQL in scripts, see the *PowerScript Reference*.

Pasting functions

You can paste any function into a script.

❖ **To paste a function into a script:**

1   Place the insertion point where you want to paste the function in the script.

2   Click the Paste Function button in the PainterBar, or select Edit>Paste Special>Function from the menu bar.

3   Choose the type of function you want to paste: built-in, user-defined, or external.

4   Double-click the function you want from the list that displays.

PowerBuilder pastes the function into the script and places the cursor within the parentheses so that you can define any needed arguments.

For more about pasting user-defined functions, see "Pasting user-defined functions" on page 204. For more about external and built-in functions, see *Application Techniques*.

Pasting contents of files

If you have code that is common across different scripts, you can keep that code in a text file, then paste it into new scripts you write. For shorter snippets of code, you can also use the Clip window. See "The Clip window" on page 15.

❖ **To import the contents of a file into the Script view:**

1   Place the insertion point where you want the file contents pasted.

2   Select Edit>Paste Special>From File from the menu bar.

The Paste From File dialog box displays, listing all files with the extension SCR. If necessary, navigate to the directory that contains the script you want to paste.

3   Choose the file containing the code you want. You can change the type of files displayed by changing the file specification in the File Name box.

PowerBuilder copies the file into the Script view at the insertion point.

Saving a script to a file

To save all or part of a script to an external text file, select the code you want to save and copy and paste it to the file editor. Use the extension *.SCR* to identify it as PowerScript code. You might want to use this technique to save a backup copy before you make major changes or so that you can use the code in other scripts.

Reverting to the unedited version of a script

You can discard the edits you have made to a script and revert to the unedited version by selecting Edit>Revert Script from the menu.

Using AutoScript

AutoScript is a tool designed to help you write PowerScript code more quickly by providing a lookup and paste service inside the Script view. It is an alternative to using the paste toolbar buttons or the Browser—you do not need to move your hands away from the keyboard to paste functions, events, variables, properties, and templates for PowerBuilder TRY, DO, FOR, IF, and CHOOSE statements into your script.

If you are not sure what the name or syntax of a function is or what the names of certain variables are, AutoScript can show you a list to choose from and paste what you need right into the script. If you can remember part of the name, start typing and select Edit>Activate AutoScript or press F8 (or do nothing if automatic pop-up is turned on). If you cannot remember the name at all, place your cursor in white space and press F8.

Where you use AutoScript

You can use AutoScript in three different contexts:

- When you can remember part of the name and you want AutoScript to finish typing it for you or show you a list of alternatives.

- When you cannot remember the name or you just want a list. AutoScript options can help you narrow the list if you do not know the name but you do know the type you are looking for. For example, you can choose to see a list showing all variables, or only all local variables.

- When you want a list of the properties and/or functions and events that apply to an identifier followed by a dot.

For how to use AutoScript options, see "Customizing AutoScript" on page 184.

Two ways to use AutoScript

AutoScript can pop up a list automatically when you pause while typing, or when you request it:

- Turn automatic pop-up on to have AutoScript pop up the list or complete what you are typing when you pause for a few seconds after typing one or more characters or an identifier followed by a dot. See "Using automatic pop-up" on page 187.

- Invoke AutoScript when you need it by pressing F8 when you have typed one or more characters or an identifier followed by a dot, or when the cursor is in white space. Pressing F8 activates AutoScript only once. It does not turn automatic pop-up on.

For how to paste an item from the pop-up window into a script, see "Using the AutoScript pop-up window" next.

Using the AutoScript pop-up window

If there is more than one property, variable, method, or statement that could be inserted, AutoScript pops up an alphabetical list of possible completions or insertions. An icon next to each item indicates its type. The following screen includes an instance variable, events, properties, statements, and a function:
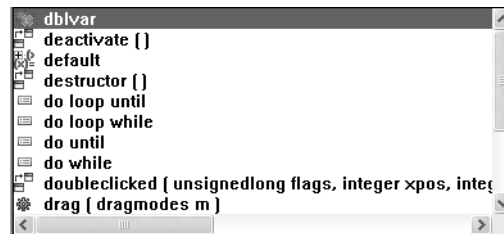
If a function is overloaded, each version displays on a different line in the AutoScript pop-up window.

If you have started typing a word, only completions that begin with the string you have already typed display in the list.

Case sensitivity

AutoScript always pastes lowercase characters, but the case of any characters you have already typed is preserved. For example, if you are using AutoScript to complete a function name and you want to use mixed case, you can type up to the last uppercase letter before invoking AutoScript. AutoScript completes the function name and pastes an argument template.

Pasting an item into the script

To paste an item into the script, press Tab or Enter or double-click the item. Use the arrow and page up and page down keys to scroll through the list. If the item is a function, event, or statement, the template that is pasted includes descriptive comments that you replace with argument names, conditions, and so forth. The first commented argument or statement is selected so that it is easy to replace. You can jump to the next comment by selecting Edit>Go To>Next Marker.

Go to next marker

You can use Edit>Go To>Next Marker to jump to the next comment enclosed by /* and */ anywhere in the Script view, not just in AutoScript templates. For the steps to create a shortcut for this menu item, see "Customizing AutoScript" next.

If you do not want to paste from the list

Press the Backspace key or click anywhere outside the pop-up window to dismiss it without pasting into the script.

If nothing displays

AutoScript does not pop up a list if the cursor is in a comment or string literal or if an identifier is complete. If neither of these conditions applies and nothing displays when you press F8, there may be no appropriate completions in the current context. Check that the options you need are selected on the AutoScript options page as described in "Customizing AutoScript" next.

Customizing AutoScript

There are four ways to customize AutoScript:

- Creating shortcut keys

- Specifying what displays in the list

- Using automatic pop-up

- Using AutoScript only with dot notation

Creating shortcut keys

AutoScript is easier to use if you create shortcuts for the menu items that you use frequently. The Edit>Activate AutoScript menu item uses F8 as a shortcut key by default, but you can change it if you prefer to use a different shortcut.

❖ **To modify or create shortcut keys for using AutoScript:**

1 Select Tools>Keyboard Shortcuts from the menu bar and expand the Edit menu in the Keyboard Shortcuts dialog box.

2 If you want to change the default key for activating AutoScript, scroll down and select Activate AutoScript and type a key sequence, such as Ctrl+Space.

3 Expand the Go To menu, select Next Marker, and type a key sequence, such as Ctrl+M.

   After you click OK, the shortcuts display in the Edit menu.

To make the next three customizations, select Design>Options from the menu bar and select the AutoScript tab:
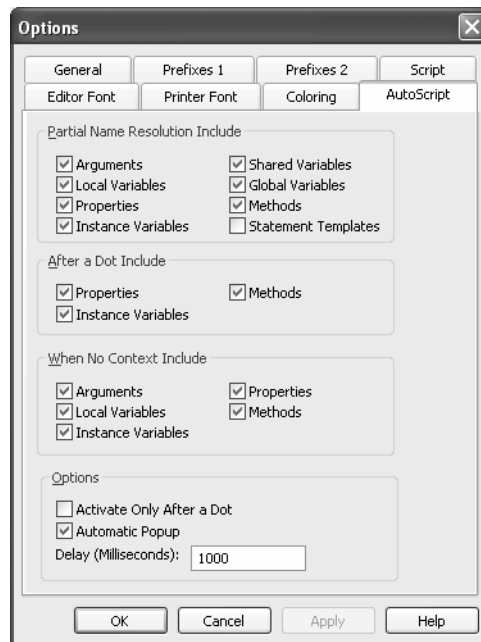
Specifying what displays in the list

You can select different items to include in three different contexts:

- When you have started typing a variable or method name or the beginning of a PowerScript statement

- When you have typed the name of an object followed by a dot

- When the cursor is at the beginning of a new line or in white space

Table 7-3 shows what is included in the list or pasted when you check each box.

Table 7-3: Setting options for AutoScript

Check box	Displays
Arguments	Arguments for the current function or event.
Local Variables	Variables defined in the current script.
Instance Variables	Variables defined for and associated with an instance of the current object or, after a dot, variables associated with the object preceding the dot.
Shared Variables	Variables defined for the current object and associated with all instances of it.
Global Variables	Variables defined for the current application.
Properties	Properties for the current object or, after a dot, properties for the object preceding the dot. Includes controls on the current window.
Methods	Functions and events for the current object or, after a dot, functions and events for the object preceding the dot.
Statement Templates	PowerScript statement templates for each type of IF, FOR, CHOOSE CASE, TRY, or DO statement with comments indicating what code should be inserted. This option is off by default.

Turning options off reduces the length of the list that displays when you invoke AutoScript so that it is faster and easier to paste a completion or insert code into the script:

- To show all variables and methods when typing, check all the boxes except Statement Templates in the Partial Name Resolution Include group box. When you pause or press F8, the list shows variables and methods that begin with the string you typed.

- To quickly find functions on an object, clear all the boxes except Methods in the After A Dot Include group box. When you type an instance name followed by a dot, only function and event names for the instance display.

- To see a list of arguments and local variables when the cursor is in white space, check the Arguments and Local Variables boxes in the When No Context Include group box. When you press F8, the list shows only arguments and local variables.

Using name completion shortens the list

You might not need to clear boxes on the AutoScript page to reduce the length of the list if you are using name completion and F8 to invoke AutoScript. For example, suppose you have created an instance called inv_ncst_dssrv of the class n_cst_dssrv and you know the function you want to use begins with of_g. Type the following into a script and then press F8:

```
inv_ncst_dssrv.of_g
```

AutoScript displays a pop-up window showing only the functions on n_cst_dssrv that begin with of_g.

Using automatic pop-up

Most of the time you will probably use a shortcut key to invoke AutoScript, but you can also have AutoScript pop up a list or paste a selection automatically whenever you pause for several seconds while typing. To do so, check the Automatic Popup box on the AutoScript options page. Automatic pop-up does not operate when the cursor is at the beginning of a line or in white space.

This feature is most useful when you are entering new code. You can customize the options in the Partial Name Resolution Include and After A Dot Include group boxes to reduce the number of times AutoScript pops up.

When you are editing existing code, it is easier to work with automatic pop-up off. AutoScript might pop up a list or paste a template for a function when you do not want it to. Using only the shortcut key to invoke AutoScript gives you complete control.

Using AutoScript only with dot notation

If you want AutoScript to work *only* when you have typed an identifier followed by a dot, check the Activate Only After a Dot box on the AutoScript options page. The effect of checking this box applies whether or not you have checked Automatic Popup. You might find it most useful when you have checked Automatic Popup, because it provides another way to limit the number of times AutoScript pops up automatically.

Example

The following simple example illustrates how AutoScript works with automatic pop-up turned off and different settings for each context. To set up the example:

- Create a new window and place on it a DataWindow control and a CommandButton control.

- Select all the boxes in the Partial Name Resolution Include group box.

- Clear all the boxes in the After A Dot Include group box except Methods.

- Clear all the boxes in the When No Context Include group box except Arguments and Local Variables.

- Clear both boxes in the Options group box.

Table 7-4: AutoScript example

Context	Do this	What happens
Partial name resolution	In the Clicked event script for cb_1, type long ll_rtn. On a new line, type ll and press F8.	AutoScript pastes the local variable ll_rtn into the script because it is the only completion that begins with ll.
	Type = d and press F8.	The list displays all properties, events, functions, variables, and statements that begin with d.
	Type w and press Tab or Enter.	The list scrolls to dw_1 and AutoScript pastes it into the script when you press Tab or Enter.
After a dot	Type a dot after dw_1 and press F8.	The list shows all the functions and events for a DataWindow control.
	Type GetNextM and press Tab or Enter.	AutoScript pastes the rest of the GetNextModified function name and template into the script, retaining your capitalization.
	Select Edit>Go To>Next Marker.	AutoScript selects the next function argument so you can replace it. Complete or comment out the statement.
No context	In the empty ItemChanged event for dw_1, declare some local variables, press Tab or Enter, and then press F8.	The list displays the local variables and the arguments for the ItemChanged event.

Getting context-sensitive Help

In addition to accessing Help through the Help menu and F1 key, you can use context-sensitive Help in the Script view to display Help for reserved words and built-in functions.

❖ **To use context-sensitive Help:**

1 Place the insertion point within a reserved word (such as DO or CREATE) or built-in function (such as Open or Retrieve).

2 Press Shift+F1.

The Help window displays information about the reserved word or function.

Copying Help text
You can copy text from the Help window into the Script view. This is an easy way to get more information about arguments required by built-in functions. You can also copy scripts directly from code examples and modify them for use in your application.

Compiling the script

Before you can execute a script, you must compile it.

❖ **To compile a script:**

• Click the Compile button, or select Edit>Compile from the menu bar.

PowerBuilder compiles the script and reports any problems it finds, as described in "Handling problems" next.

PowerBuilder compiles automatically
When you attempt to open a different script in a Script view, PowerBuilder compiles the current script. When you save the object, such as the window containing a control you wrote a script for, PowerBuilder recompiles all scripts in the object to make sure they are still valid. For example, PowerBuilder checks that all objects that were referenced when you wrote the script still exist.

Handling problems

If problems occur when a script is compiled, PowerBuilder displays messages in a Message window below the script.

```
(0003): Information C0146: The identifier 'error' conflicts with an existing gl
(0004): Warning     C0150: Function 'dbErrorCode' is obsolete.  The database er
(0005): Warning     C0014: Undefined variable: source
(0005): Error       C0124: Illegal expression on left side of assignment
(0012): Error       C0052: Bad argument list for function: messagebox
(0018): Error       C0031: Syntax error
```

There are three kinds of messages:

>   Errors
>   Warnings
>   Information messages

Understanding errors

Errors indicate serious problems that you must fix before a script will compile and before you can close the Script view or open another script in the same view. Errors are shown in the Message window as:

>   *line number:* Error *error number:message*

Understanding warnings

Warnings indicate problems that you should be aware of but that do not prevent a script from compiling.

There are three kinds of warnings.

Compiler warnings   Compiler warnings inform you of syntactic problems, such as undeclared variables. PowerBuilder lets you compile a script that contains compiler warnings, but you must fix the problem in the script before you can save the object that the script is for, such as the window or menu. Compiler warnings are shown in the Message window as:

>   *line number:* Warning *warning number:message*

Obsolete warnings   Obsolete warnings inform you when you use any obsolete functions or syntax in your script. Obsolete functions, although they still compile and run, have been replaced by more efficient functions and will be discontinued in a future release of PowerBuilder. You should replace all references to obsolete functions as soon as possible. Obsolete warnings are shown in the Message window as:

>   *line number:* Warning *warning number:message*

Database warnings Database warnings come from the database manager you are connected to. PowerBuilder connects to the database manager when you compile a script containing embedded SQL. Typically, these warnings arise because you are referencing a database you are not connected to. Database warnings are shown in the Message window as:

> *line number:* Database *warning number:message*

PowerBuilder lets you compile scripts with database warnings and also lets you save the associated object. It does this because it does not know whether the problem will apply during execution, since the execution environment might be different from the compile-time environment.

You should study database warnings carefully to make sure the problems will not occur at runtime.

Understanding information messages

Information messages are issued when there is a potential problem. For example, an information message is issued when you have used a global variable name as a local variable, because that might result in a conflict later.

Information messages are shown in the Message window as:

> *line number:* Information *number:message*

Displaying warnings and messages

To specify which messages display when you compile, select Design>Options to open the Options dialog box, select the Script tab page, and check or clear the Display Compiler Warnings, Display Obsolete Messages, Display Information Messages, and Display Database Warnings check boxes. The default is to display compiler and database warning messages. Error messages always display.

Fixing problems

To fix a problem, click the message. The Script view scrolls to display the statement that triggered the message. After you fix all the problems, compile the script again.

To save a script with errors
Comment out the lines containing errors.

Disabling database connection when compiling and building

When PowerBuilder compiles an application that contains embedded SQL, it connects to the database profile last used in order to check for database access errors during the build process. For applications that use multiple databases, this can result in spurious warnings during the build since the embedded SQL can be validated only against that single last-used database and not against the databases actually used by the application. In addition, an unattended build, such as a lengthy overnight rebuild, can stall if the database connection cannot be made.

To avoid these issues, you can add a new entry to the *PB.INI* file:

```
[pb]
dbsign=0
```

Caution
Set the value of dbsign to 0 only when you want to compile without signing on to the database. Compiling without connecting to a database prevents the build process from checking for database errors and may therefore result in runtime errors later.

Declaring variables and external functions

The default layout in the Application, Window, and User Object painters includes a Script view set up to declare variables. Keeping a separate Script view open makes it easy to declare any variables or external functions you need to use in your code without closing and compiling the script.

❖ **To declare variables and external functions:**

1   Select [Declare] from the first list in the Script view.

2   Select the variable type (instance, shared, or global) or the function type (local or global) from the second list.

3   Type the declaration in the Script view.

For more information about declaring variables, see the *PowerScript Reference*. For more information about declaring and using external functions, see the *PowerScript Reference* and *Application Techniques*.

CHAPTER 8   **Working with User-Defined Functions**

About this chapter   This chapter describes how to build and use user-defined functions.

Contents

Topic	Page
About user-defined functions	193
Defining user-defined functions	195
Modifying user-defined functions	202
Using your functions	204

About user-defined functions

The PowerScript language has many built-in functions, but you may find that you need to code the same procedure over and over again. For example, you may need to perform a certain calculation in several places in an application or in different applications. In such a situation, create a user-defined function to perform the processing.

A user-defined function is a collection of PowerScript statements that perform some processing. After you define a user-defined function and save it in a library, any application accessing that library can use the function.

There are two kinds of user-defined functions, global and object-level functions.

Global functions   Global functions are not associated with any object in your application and are always accessible anywhere in the application.

They correspond to the PowerBuilder built-in functions that are not associated with an object, such as the mathematical and string-handling functions. You define global functions in the Function painter.

Object-level functions

Object-level functions are defined for a window, menu, user object, or application object. These functions are part of the object's definition and can always be used in scripts for the object itself. You can choose to make these functions accessible to other scripts as well.

These functions correspond to built-in functions that are defined for specific PowerBuilder objects such as windows or controls. You define object-level functions in a Script view for the object.

Deciding which kind you want

When you design your application, you need to decide how you will use the functions you will define:

- If a function is general purpose and applies throughout an application, make it a global function.

- If a function applies only to a particular kind of object, make it an object-level function. You can still call the function from anywhere in the application, but the function acts only on a particular object type.

  For example, suppose you want a function that returns the contents of a SingleLineEdit control in one window to another window. Make it a window-level function, defined in the window containing the SingleLineEdit control. Then, anywhere in your application that you need this value, call the window-level function.

Multiple objects can have functions with the same name
Two or more objects can have functions with the same name that do different things. In object-oriented terms, this is called polymorphism. For example, each window type can have its own Initialize function that performs processing unique to that window type. There is never any ambiguity about which function is being called, because you always specify the object's name when you call an object-level function.

Object-level functions can also be overloaded—two or more functions can have the same name but different argument lists. Global functions *cannot* be overloaded.

Defining user-defined functions

Although you define global functions in the Function painter and object-level functions in the painter for a specific object, in both cases you define and code the function in a Script view.

When you add a new function, a Prototype window displays above the script area in the Script view. The fields in the Prototype window are in the same order as the function's signature:

- The function's access level, return type, and name

- For each parameter, how it is passed, its datatype, and its name

- The exceptions the function can throw, if any
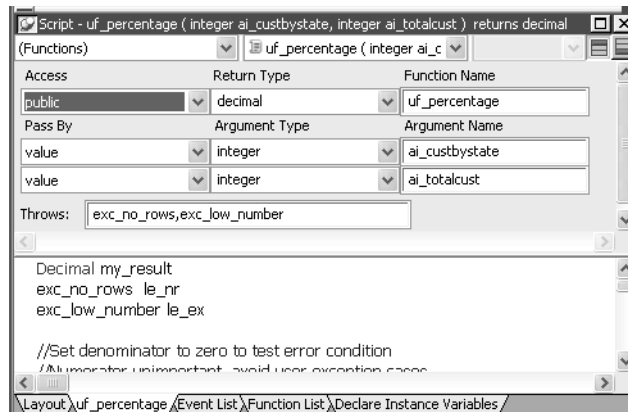
The following sections describe each of the steps required to define and code a new function:

1   Opening a Prototype window to add a new function.

2   Defining the access level (for object-level functions).

3   Defining a return type.

4   Naming the function.

5   Defining arguments.

6   Defining a THROWS clause.

7   Coding the function.

8   Compiling and saving the function.

Opening a Prototype window to add a new function

How you create a new function depends on whether you are defining a global function or an object-level function.

❖ **To create a new global function:**

- Select File>New from the menu bar and select Function from the PB Object tab.

  The Function painter opens, displaying a Script view with an open Prototype window in which you define the function.

❖ **To create a new object-level function:**

1 Open the object for which you want to declare a function.

  You can declare functions for windows, menus, user objects, or applications.

2 Select Insert>Function from the menu bar, or, in the Function List view, select Add from the pop-up menu.

  The Prototype window opens in a Script view or, if no Script view is open, in a new Script view.

Defining the access level

In the Prototype window, use the drop-down list labeled Access to specify where you can call the function in the application.

For global functions

Global functions can always be called anywhere in the application. In PowerBuilder terms, they are public. When you are defining a global function, you cannot modify the access level; the field is read-only.

For object-level functions

You can restrict access to an object-level function by setting its access level.

Table 8-1: Access levels for object-level functions

Access	Means you can call the function
Public	In any script in the application.
Private	Only in scripts for events in the object in which the function is defined. You cannot call the function from descendants of the object.
Protected	Only in scripts for the object in which the function is defined and scripts for that object's descendants.

If a function is to be used only internally within an object, you should define its access as private or protected. This ensures that the function is never called inappropriately from outside the object. In object-oriented terms, defining a function as protected or private encapsulates the function within the object.

Transaction server components

If you are defining functions for a custom class user object that you will use as an EAServer or COM component, remember that only public functions can appear in the interface for the component.

Defining a return type

Many functions perform some processing and then return a value. That value can be the result of the processing or a value that indicates whether the function executed successfully or not. To have your function return a value, you need to define its return type, which specifies the datatype of the returned value.

You must code a return statement in the function that specifies the value to return. See "Returning a value" on page 201. When you call the function in a script or another function, you can use an assignment statement to assign the returned value to a variable in the calling script or function. You can also use the returned value directly in an expression in place of a variable of the same type.

❖ **To define a function's return type:**

- Select the return type from the Return Type drop-down list in the Prototype window, or type in the name of an object type you have defined.

  You can specify any PowerBuilder datatype, including the standard datatypes, such as integer and string, as well as objects and controls, such as DataStore or MultiLineEdit.

  You can also specify as the return type any object type that you have defined. For example, if you defined a window named w_calculator and want the function to process the window and return it, type w_calculator in the Return Type list. You cannot select w_calculator from the list, because the list shows only built-in datatypes.

❖ **To specify that a function does not return a value:**

• Select (None) from the Return Type list.

  This tells PowerBuilder that the function does not return a value. This is similar to defining a procedure or a void function in some programming languages.

Examples of functions returning values

The following examples show the return type you would specify for some different functions:

If you are defining	Specify this return type
A mathematical function that does some processing and returns a real number	real
A function that takes a string as an argument and returns the string in reverse order	string
A function that is passed an instance of window w_calculator, does some processing (such as changing the window's color), then returns the modified window	w_calculator

Naming the function

Name the function in the Function Name box. Function names can have up to 40 characters. For valid characters, see the *PowerScript Reference*.

For object-level functions, the function is added to the Function List view when you tab off the Function Name box. It is saved as part of the object whenever you save the object.

Using a naming convention for user-defined functions makes them easy to recognize and distinguish from built-in PowerScript functions. A commonly used convention is to preface all global function names with f_ and object-level functions with of_, such as:

```
// global functions
f_calc
f_get_result

// object-level functions
of_refreshwindow
of_checkparent
```

Built-in functions do not usually have underscores in their names, so this convention makes it easy for you to identify functions as user defined.

Defining arguments

Like built-in functions, user-defined functions can have any number of arguments, including none. You declare the arguments and their types when you define a function.

Passing arguments

In user-defined functions, you can pass arguments by reference, by value, or read-only. You specify this for each argument in the Pass By list.

By reference   When you pass an argument by reference, the function has access to the original argument and can change it directly.

By value   When you pass by value, you are passing the function a temporary local copy of the argument. The function can alter the value of the local copy within the function, but the value of the argument is not changed in the calling script or function.

Read-only   When you pass as read-only, the variable's value is available to the function but it is treated as a constant. Read-only provides a performance advantage over passing by value for string, blob, date, time, and datetime arguments, because it does not create a copy of the data.

If the function takes no arguments
Leave the initial argument shown in the Prototype window blank.

❖   **To define arguments:**

1   Declare whether the first argument is passed by reference, by value, or read-only.

The order in which you specify arguments here is the order you use when calling the function.

2   Declare the argument's type. You can specify any datatype, including:

   •   Built-in datatypes, such as integer and real

   •   Object types, such as window, or specific objects, such as w_emp

   •   User objects

   •   Controls, such as CommandButtons

3   Name the argument.

4   If you want to add another argument, press the Tab key or select Add Parameter from the pop-up menu and repeat steps 1 to 3.

Passing arrays
You must include the square brackets in the array definition, for example, `price[ ]` or `price[50]`, and the datatype of the array must be the datatype of the argument. For information on arrays, see the *PowerScript Reference*.

Defining a THROWS clause

If you are using user-defined exceptions, you must define what exceptions might be thrown from a user-defined function or event. You use the Throws box to do this.

When you need to add a THROWS clause

Any developers who call the function or event need to know what exceptions can be thrown from it so that their code can handle the exceptions. If a function contains a THROW statement that is not surrounded by a try-catch block that can deal with that type of exception, then the function must be declared to throw that type of an exception or some ancestor of that exception type.

There are two exception types that inherit from the Throwable object: Exception and RuntimeError. Typically, you add objects that inherit from Exception to the THROWS clause of a function. Exception objects are the parents of all checked exceptions, which are exceptions that must be dealt with when thrown and declared when throwing. You do not need to add Runtime error objects to the THROWS clause, because they can occur at any time. You can catch these errors in a try-catch block, but you are not required to.

Adding a THROWS clause

You can add a THROWS clause to any PowerBuilder function or to any user event that is not defined by an event ID. To do so, drag and drop it from the System Tree, or type the name of the object in the box. If you type the names of multiple user objects in the Throws box, use a comma to separate the object names. When you drag and drop multiple user objects, PowerBuilder automatically adds the comma separators.

The PowerBuilder compiler checks whether a user-defined exception thrown on a function call in a script matches an exception in the THROWS clause for that function. It prompts you if there is no matching exception in the THROWS clause.

You can define a user-defined exception object, and inherit from it to define more specific lower-level exceptions. If you add a high-level exception to the throws clause, you can throw any lower-level exception in the script, but you risk hiding any useful information obtainable from the lower-level exception.

For more information about exception handling, see *Application Techniques*.

Coding the function

When you have finished defining the function prototype, you specify the code for the function just as you specify the script for an event in the Script view. For information about using the Script view, see Chapter 7, "Writing Scripts."

What functions can contain

User-defined functions can include PowerScript statements, embedded SQL statements, and calls to built-in, user-defined, and external functions.

You can type the statements in the Script view or use the buttons in the PainterBar or items on the Edit>Paste Special menu to insert them into the function. For more information, see "Pasting information into scripts" on page 177.

Returning a value

If you specified a return type for your function in the Prototype window, you must return a value in the body of the function. To return a value in a function, use the RETURN statement:

RETURN *expression*

where *expression* is the value you want returned by the function. The datatype of the expression must be the datatype you specified for the return value for the function.

Example

The following function returns the result of dividing *arg1* by *arg2* if *arg2* does not equal zero. It returns –1 if *arg2* equals zero:

```
IF arg2 <> 0 THEN
   RETURN arg1 / arg2
ELSE
   RETURN -1
END IF
```

Compiling and saving the function

When you finish building a function, compile it and save it in a library. Then you can use it in scripts or other user-defined functions in any application that includes the library containing the function in its library search path. You compile the script and handle errors as described in "Compiling the script" on page 189.

Modifying user-defined functions

You can change the definition of a user-defined function at any time. You change the processing performed by the function by modifying the statements in the Script view. You can also change the return type, argument list, or access level for a function.

❖ **To change a function's return type, arguments, or access level:**

1 Do one of the following:

   • In the Function painter, open the global function.

   • Open the object that contains the object-level function you want to edit and select the function from the Function list.

2 Make the changes you want in the Prototype window.

   If the Prototype window is hidden, click the toggle button to display it.

3 Select File>Save from the menu bar.

❖ **To change a function's name:**

1 If desired, modify the function's return type, arguments, or access level as described in the previous procedure.

2 Do one of the following:

   • In the Function painter, select File>Save As from the menu bar and enter a name.

   • In the Script view, enter a new name in the Function Name box.

   When you tab off the box, the new function name displays in the Function List view.

Changing the arguments

You can change a function's arguments at any time using the pop-up menu in the Prototype window:

• *Add* an argument by selecting Add Parameter. Boxes for defining the new argument display below the last argument in the list.

• *Insert* an argument by moving the pointer to the argument before which you want to insert the argument and selecting Insert Parameter. Boxes for defining the new argument display above the selected argument.

• *Delete* an argument by selecting it and clicking the Delete button.

To change the position of an argument
To change the position of an argument, delete the argument and insert it as a new argument in the correct position.

Recompiling other scripts

Changing arguments and the return type of a function affect scripts and other functions that call the function. You should recompile any script in which the function is used. This guarantees that the scripts and functions work correctly during execution.

Seeing where a function is used

PowerBuilder provides browsing facilities to help you find where you have referenced your functions. In the System Tree or Library painter, select a target, library, or object and select Search from the pop-up menu. You can also search multiple entries in the Library painter:

❖ **To determine which functions and scripts call a user-defined function:**

1   Open the Library painter.

2   In a List view, select all the entries you want to search for references to the user-defined function.

3   Select Entry>Search from the menu bar.

The Search Library Entries dialog box displays.

4   Specify the user-defined function as the search text and specify the types of components you want to search.

5   Click OK.

PowerBuilder displays all specified components that reference the function in the Output window. You can double-click a listed component to open the appropriate painter.

For more about browsing library entities, see "Searching targets, libraries, and objects" on page 154.

Using your functions

You use user-defined functions the same way you use built-in functions. You can call them in event scripts or in other user-defined functions.

For complete information about calling functions, see *Application Techniques*.

Pasting user-defined functions

When you build a script in the Script view, you can type the call to the user-defined function. You can also paste the function into the script. There are four ways to paste a user-defined function into a script:

• Drag the function from the System Tree to the Script view.

• Select Edit>Paste Special>Function>User-defined from the menu bar.

• Enable AutoScript, select the function's signature in the list that displays when you pause, and press Tab or Enter.

• Select the function in the Browser and copy and paste it into the script.

Using the System Tree, AutoScript, or the Browser pastes the function's prototype arguments as well as its name into the script.

For more information about AutoScript, see "Using AutoScript" on page 182.

❖ **To paste a user-defined function into a script from the Browser:**

1 Select Tools>Browser from the menu bar.

2 Do one of the following:

   • Select a global function from the Function page.

   • Select the object that contains the object-level function you want to paste from the corresponding page (such as the Window page).

3 Double-click the Functions category in the right pane.

4 Select the function you want to paste and select Copy from its pop-up menu.

5 In the Script view, move the insertion point to where you want to paste the function and select Paste from the pop-up menu.

   The function and its prototype parameters display at the insertion point in your script.

6 Specify the required arguments.

Working with User Events

About this chapter

This chapter introduces user events, describes how to define them, and discusses how to use them in an application.

Contents

Topic	Page

About user events

Windows, user objects, controls, menus, and Application objects each have a predefined set of events. In most cases, the predefined events are all you need, but there are times when you want to declare your own user event. You can use predefined event IDs to trigger a user event, or you can trigger it exclusively from within your application scripts.

Features that you might want to add to your application by creating user events include keystroke processing, providing multiple ways to perform a task, and communication between a user object and a window.

Keystroke processing

Suppose that you want to modify the way keystrokes are processed in your application. For example, in a DataWindow control, you want the user to be able to press the Down Arrow and Up Arrow keys to scroll among radio buttons in a DataWindow column. Normally, pressing these keys moves the focus to the next or preceding row.

To do this, you define user events corresponding to Windows events that PowerBuilder does not define.

Multiple methods	Suppose that you want to provide several ways to accomplish a certain task within a window. For example, you want the user to be able to update the database by either clicking a button or selecting a menu item. In addition, you want to provide the option of updating the database when the user closes the window.
	To do this, you define a user event to update the database.
Communication between user object and window	Suppose that you have placed a custom visual user object in a window and need to communicate between the user object and the window. For information, see "Communicating between a window and a user object" on page 379.

User events and event IDs

An event ID connects events related to user actions or system activity to a system message. PowerBuilder defines (or maps) events to commonly used event IDs, and when it receives a system message, it uses the mapped event ID to trigger an event.

User-defined events do not have to be mapped to an event ID. See "Defining user events" on page 208.

Event ID names

The PowerBuilder naming convention for user event IDs is similar to the convention Windows uses to name messages. All PowerBuilder event IDs begin with pbm_.

Event IDs associated with Windows messages	Several Windows messages and notifications map to PowerBuilder event IDs.
	For Windows messages that begin with wm_, the PowerBuilder event ID typically has the same name with pbm_ substituted for wm_. For messages from controls, the PowerBuilder event ID typically has the same name but begins with pbm_ and has the Windows prefix for the control added to the message name. For example:

- wm_keydown maps to pbm_keydown

- bm_getcheck (a button control message) maps to pbm_bmgetcheck

- bn_clicked (a button control notification message) maps to pbm_bnclicked

To see a list of event IDs to which you can map a user-defined event, select Insert>Event and display the Event ID drop-down list in the Prototype window that displays.

Windows messages that are not mapped to a PowerBuilder event ID map to the pbm_other event ID. The PowerBuilder Message object is populated with information about system events that are not mapped to PowerBuilder event IDs. For more information about the Message object, see *Objects and Controls* or *Application Techniques*.

For more information about Windows messages and notifications, see the information about Windows controls and Windows management in the section on user interface design and development in the Microsoft MSDN Library at http://msdn.microsoft.com/library/default.asp.

Event IDs associated with PowerBuilder events

PowerBuilder has its own events, each of which has an event ID. For example, the PowerBuilder event DragDrop has the event ID pbm_dragdrop. The event name and event ID of the predefined PowerBuilder events are protected; they cannot be modified. The event IDs for predefined events are shown in the Event List view:

Custom event IDs

The list of event IDs that displays in the Event ID drop-down list in the Prototype window includes custom event IDs. Custom user events can be mapped from Windows wm_user message numbers to pbm_custom*xx* event IDs.

Obsolete technique
This technique is not recommended and is considered to be obsolete. The ability to use this technique has been retained for backward compatibility. If you do not want to map a user event to a named pbm_ code, use an unmapped user event as described in "Unmapped user events" on page 209.

These event IDs were intended for use with DataWindow controls, windows, and user objects other than standard visual user objects, which behave like the built-in controls they inherit from. They were not intended for use with standard controls.

Defining custom user events for standard controls can cause unexpected behavior because all standard controls respond to standard events in the range 0 to 1023. Most controls also define their own range of custom events beyond 1023, corresponding to wm_user messages, and some controls have custom events that overlap with the PowerBuilder custom events. The pbm_custom01 event ID maps to wm_user+0, pbm_custom02 maps to wm_user+1, and so on, through pbm_custom75, which maps to wm_user+74.

Defining user events

In PowerBuilder, you can define both mapped and unmapped user events for windows, user objects, controls, menus, and the Application object.

When you add a new event, a Prototype window displays above the script area in the Script view. Most of the fields in the Prototype window are the same as when you define a user-defined function. They are in the same order as the event's signature: access level, return type, and name; then for each parameter, how it is passed, its datatype, and its name; and finally, the THROWS clause. For information about filling in these fields, see "Defining user-defined functions" on page 195.

The access level for events is always public.
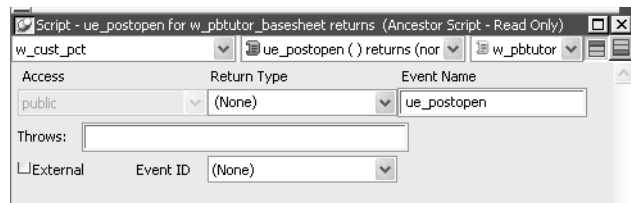
The Prototype window for user events has an additional field that you use if you want to map the user event to an event ID.

External check box

When you select the External check box, PowerBuilder sets the IsExternalEvent property of the ScriptDefinition object associated with the event to true. This has no effect on your application in this release. The feature may be used in a future release.

Mapped user events

When a system message occurs, PowerBuilder triggers any user event that has been mapped to the message and passes the appropriate values to the event script as arguments. When you define a user event and map it to an event ID, you must use the return value and arguments that are associated with the event ID.

Unmapped user events

Unmapped user events are associated with a PowerBuilder activity and do not have an event ID. When you define an unmapped user event, you specify the arguments and return datatype; only your application scripts can trigger the user event. For example, if you create an event called ue_update that updates a database, you might trigger or post the event in the Clicked event of an Update command button.

❖ **To define a mapped user event:**

1 Open the object for which you want to define a user event.

2 If you want to define a user event for a control on a window or visual user object, double-click the control to select it.

3 Select Insert>Event from the menu bar, or, in the Event List view, select Add from the pop-up menu.

The Prototype window opens in the Script view. If you display the Script view's title bar, you see (Untitled) because you have not named the event yet. If there is no open Script view, a new view opens.

4 Name the event and tab to the next field.

Event names can have up to 40 characters. For valid characters, see the *PowerScript Reference*.

To recognize user events easily, consider prefacing the name with an easily recognizable prefix such as ue_.

When you tab to the next field, the user event is added to the Event List view. It is saved as part of the object whenever you save the object.

5 Select an ID from the drop-down list box at the bottom of the Prototype window.

❖ **To define an unmapped user event:**

1 Open the object for which you want to define a user event.

2 If you want to define a user event for a control on a window or visual user object, double-click the control to select it.

3 Select Insert>Event from the menu bar, or, in the Event List view, select Add from the pop-up menu.

The Prototype window opens in the Script view. If you display the Script view's title bar, you see (Untitled) because you have not named the event yet. If there is no open Script view, a new view opens.

4 Select a return type and tab to the next field.

Defining return types for events is similar to defining them for functions. See "Defining a return type" on page 197.

When you can specify return type and arguments
If you map the user event to an event ID, you cannot change its return type or specify arguments.

---

5 Name the event and tab to the next field.

Event names can have up to 40 characters. For valid characters, see the *PowerScript Reference*.

To recognize user events easily, consider prefacing the name with an easily recognizable prefix such as ue_.

When you tab to the next field, the user event is added to the Event List view. It is saved as part of the object whenever you save the object.

6 If the event will take arguments, define arguments for the event.

Defining arguments for events is similar to defining them for functions. See "Defining arguments" on page 199 and "Changing the arguments" on page 202.

7 Optionally enter the name of exceptions that can be thrown by the event.

❖ **To open a user event for editing:**

• In the Event List view, double-click the event's name.

❖ **To delete a user event:**

• In the Event List view, select the user event's name and select Delete from the Edit menu or the pop-up menu.

Using a user event

After you define a user event, you must write the script that PowerBuilder will execute when that user event is triggered. If it is an unmapped user event, you also write the code that will trigger the user event.

User events display in alphabetical order in the Event List view and the event list box in the Script view, along with the predefined events. As with predefined events, the script tells PowerBuilder what processing to perform when the user event occurs.

If the user event is not mapped to a Windows message (that is, if there is no event ID associated with it), you must trigger the event in a script. You can trigger the user event in an object using the EVENT syntax. For information about calling events, see the *PowerScript Reference*.

Examples of user event scripts

This section includes two examples that use a mapped user event and one example that uses an unmapped user event. For more user event examples, see "Communicating between a window and a user object" on page 379.

Example 1: mapped user event for a control

Situation   You have several SingleLineEdit controls in a window and want the Enter key to behave like the Tab key (if users press Enter, you want them to tab to the next SingleLineEdit).

Solution   Define a user event for each SingleLineEdit. Give the event any name you want, such as ue_CheckKey. Map the event to the event ID pbm_keydown. Write a script for the user event that tests for the key that was pressed. If Enter was pressed, set the focus to the SingleLineEdit that you want the user to go to.

For example, in the script for the user event for sle_1, you could code:

```
// Script for user event ue_CheckKey
// which is mapped to pbm_keydown.
IF KeyDown(KeyEnter!) THEN   // Go to sle_2 if
   sle_2.SetFocus( )          // Enter pressed.
END IF
```

Similarly, in the script for the user event for sle_2, you could code:

```
// Script for user event ue_CheckKey,
// which is mapped to pbm_keydown.
IF KeyDown(KeyEnter!) THEN  // Go to sle_3 if
    sle_3.SetFocus( )       // Enter pressed.
END IF
```

Example 2: mapped user event for an edit style

Situation   You have a DataWindow control with a column that uses the RadioButton edit style and you want to allow users to scroll through the RadioButtons when they press Down Arrow or Up Arrow (normally, pressing Down Arrow or Up Arrow scrolls to the next or preceding row).

Solution   Declare a user event for the DataWindow control that maps to the event ID pbm_dwnkey and write a script like the following for it. dwn stands for DataWindow notification.

```
// Script is in a user event for a DataWindow control.
// It is mapped to pbm_dwnkey. If user is in column
// number 6, which uses the RadioButton edit style, and
// presses DownArrow, the cursor moves to the next item
// in the RadioButton list, instead of going to the next
// row in the DataWindow, which is the default behavior.
// Pressing UpArrow moves to preceding RadioButton.
//
// Note that the CHOOSE CASE below tests for data
// values, not display values, for the RadioButtons.

int colnum = 6                     // Column number
long rownum
rownum = dw_2.GetRow( ) // Current row

IF KeyDown(KeydownArrow!) AND &
    This.GetColumn( ) = colnum THEN
    CHOOSE CASE dw_2.GetItemString(rownum, colnum)
       case "P"            // First value in RB
          This.SetItem(rownum, colnum,"L")  // Next
       case "L"            // Second value in RB
          This.SetItem(rownum, colnum,"A")  // Next
       case "A"            // Last value in RB
          This.SetItem(rownum, colnum,"P")  // First
    END CHOOSE
    This.SetActionCode(1)   // Ignore key press
END IF
```

```
// The following code does same thing for UpArrow.
IF KeyDown(KeyupArrow!) AND &
   This.GetColumn( ) = colnum THEN
   CHOOSE CASE dw_2.GetItemString(rownum, colnum)
      case "P"               // First value in RB
         This.SetItem(rownum, colnum,"A")   // Last
      case "L"               // Another value in RB
         This.SetItem(rownum, colnum,"P")
      case "A"               // Last value in RB
         This.SetItem(rownum, colnum,"L")
   END CHOOSE
   This.SetActionCode(1)
END IF
```

Example 3: unmapped user event for menu options

Situation   Suppose you use the same menu in all your windows, but you want to enable or disable some menu items, in this case database update items, depending on which window the user is in.

Solution   In the window that will be the ancestor of all the sheet windows that do not have database update capability, define an unmapped user event called ue_ct_menu_enable. The event takes a boolean argument, ab_state, to set or clear the enabled property on various menus. This is the script for the ue_ct_menu_enable user event in the ancestor window:

```
//  Enable / Disable Menu Options
im_CurrMenu.m_maint.m_add.enabled = Not ab_state
im_CurrMenu.m_maint.m_delete.enabled = Not ab_state
im_CurrMenu.m_maint.m_undelete.enabled = Not ab_state
im_CurrMenu.m_maint.m_update.enabled = Not ab_state
im_CurrMenu.m_maint.m_close.enabled = ab_state
```

Then, in the script for the Activate event in the ancestor window, call the user event and pass the value "true" for the boolean variable ab_state.

```
this.EVENT ue_ct_menu_enable ( TRUE )
```

Write a similar script for the Deactivate event with the value "false" for ab_state.

You can use this window as the ancestor of any sheet window in your application that does not have database update capability. When the window is active, the Add, Delete, Undelete, and Update menu items are grayed out. When it is not active, the Close item is grayed out.

For windows that have database update capability, you can create a second ancestor window that inherits from the ancestor window in which you defined ue_ct_menu_enable. In the second ancestor window, you can override the ue_ct_menu_enable event script so that the appropriate menu options are enabled.

CHAPTER 10 **Working with Structures**

About this chapter   This chapter describes how to build and use structures.

Contents

About structures

A structure is a collection of one or more related variables of the same or different data types grouped under a single name. In some languages, such as Pascal and COBOL, structures are called records.

Structures allow you to refer to related entities as a unit rather than individually. For example, if you define the user's ID, address, access level, and a picture (bitmap) of the employee as a structure called s_employee, you can then refer to this collection of variables as s_employee.

Two kinds   There are two kinds of structures:

• Global structures, which are not associated with any object in your application. You can declare an instance of the structure and reference the instance in any script in your application.

• Object-level structures, which are associated with a particular type of window, menu, or user object, or with the Application object. These structures can always be used in scripts for the object itself. You can also choose to make the structures accessible from other scripts.

Deciding which kind you want

When you design your application, think about how the structures you are defining will be used:

• If the structure is general-purpose and applies throughout the application, make it a global structure.

• If the structure applies only to a particular type of object, make it an object-level structure.

Defining structures

Although you define object-level structures in the painter for a specific object and global structures in the Structure painter, in both cases you define the structure in a Structure view. The following sections describe each of the steps you take to define a new structure:

1 Open a Structure view.

2 For object-level structures, name the structure.

3 Define the variables that make up the structure.

4 Save the structure.

Opening a Structure view

How you open the Structure view depends on whether you are defining an object-level structure or a global structure.

❖ **To define an object-level structure:**

1 Open the object for which you want to declare the structure.

You can declare structures for windows, menus, user objects, or applications.

2 Select Insert>Structure from the menu bar.

A Structure view opens.
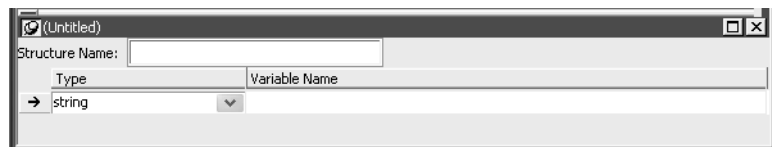
❖ **To define a global structure:**

• Select Structure from the Objects tab in the New dialog box.

    The Structure painter opens. It has one view, the Structure view. In the Structure painter, there is no Structure Name text box in the Structure view.

Naming the structure

If you are defining an object-level structure, you name it in the Structure Name box in the Structure view. If you are defining a global structure, you name it when you save the structure.

Structure names can have up to 40 characters. For information about valid characters, see the *PowerScript Reference*.

You might want to adopt a naming convention for structures so that you can recognize them easily. A common convention is to preface all global structure names with s_ and all object-level structure names with str_.

Defining the variables

❖ **To define the variables that compose the structure:**

1   Enter the datatype of a variable that you want to include in the structure.

    The default for the first variable is string; the default for subsequent variables is the datatype of the previous variable. You can specify any PowerBuilder datatype, including the standard data types such as integer and string, as well as objects and controls such as Window or MultiLineEdit.

    You can also specify any object types that you have defined. For example, if you are using a window named w_calculator that you have defined and you want the structure to include the window, type w_calculator as the datatype. (You cannot select w_calculator from the list, since the list shows only built-in data types.)

    A structure as a variable
    A variable in a structure can itself be a structure. Specify the structure's name as the variable's datatype.

    Specifying decimal places
    If you select decimal as the datatype, the default number of decimal places is 2. You can also select decimal{2} or decimal{4} to specify 2 or 4 decimal places explicitly.

2   Enter the name of the variable.

3   Repeat until you have entered all the variables.

Saving the structure

How you save the structure depends on whether it is an object-level structure or a global structure.

The names of object-level structures are added to the Structure List view and display in the title bar of the Structure view as soon as you tab off the Structure Name box. As you add variables to the structure, the changes are saved automatically. When you save the object that contains the structure, the structure is saved as part of the object in the library where the object resides.

Comments and object-level structures
You cannot enter comments for an object-level structure, because it is not a PowerBuilder object.

❖ **To name and save a global structure:**

1 Select File>Save from the menu bar, or close the Structure painter.

The Save Structure dialog box displays.

2 Name the structure.

See "Naming the structure" on page 217.

3 (Optional) Add comments to describe your structure.

4 Choose the library in which to save the structure.

5 Click OK.

PowerBuilder stores the structure in the specified library. You can view the structure as an independent entry in the Library painter.

Modifying structures

❖ **To modify a structure:**

1 Do one of the following:

• In the Open dialog box, select the global structure you want to modify.

• Open the painter for the object that contains the object-level structure and select the structure from the Structure List view.

If the Structure List view is not open, select it from the View menu.

2   Review the variable information displayed in the Structure view and modify the structure as necessary.

To insert a variable before an existing variable, highlight it and select Insert>Row from the menu bar or Insert Row from the pop-up menu.

To delete a variable, select Delete Row from the pop-up menu.

3   Save the modified structure.

Building a similar structure

If you want to create a structure that is similar to one that already exists, you can use the existing structure as a starting point and modify it.

❖   **To build an object-level structure that is similar to an existing object-level structure:**

1   Select the existing structure in the Structure List view.

2   Select Duplicate from the pop-up menu.

3   Name the new structure in the Structure Name box.

4   Modify variables as needed.

❖   **To build a global structure that is similar to an existing global structure:**

1   Open and modify the existing structure.

2   Select File>Save As to save the structure under another name or in another library.

Using structures

After you define the structure, you can:

•   Reference an instance of the structure in scripts and functions

•   Pass the structure to functions

•   Display and paste information about structures by using the Browser

Referencing structures

When you define a structure, you are defining a new datatype. You can use this new datatype in scripts and user-defined functions as long as the structure definition is stored in a library in the application's library search path.

❖ **To use a structure in a script or user-defined function:**

1   Declare a variable of the structure type.

2   Reference the variable in the structure.

Referencing global structures

The variables in a structure are similar to the properties of a PowerBuilder object. To reference a global structure's variable, use dot notation:

structure.variable

Example   Assume that s_empdata is a global structure with the variables emp_id, emp_dept, emp_fname, emp_lname, and emp_salary. To use this structure definition, declare a variable of type s_empdata and use dot notation to reference the structure's variables, as shown in the following script:

```
s_empdata   lstr_emp1, lstr_emp2 // Declare 2 variables
                                 // of type emp_data.


lstr_emp1.emp_id = 100        // Assign values to the
lstr_emp1.emp_dept = 200      // structure variables.
lstr_emp1.emp_fname = "John"
lstr_emp1.emp_lname = "Paul-Jones"
lstr_emp1.emp_salary = 99908.23

// Retrieve the value of a structure variable.
lstr_emp2.emp_salary = lstr_emp1.emp_salary * 1.05

// Use a structure variable in a
// PowerScript function.
MessageBox ("New Salary", &
   String(lstr_emp2.emp_salary,"$###,##0.00"))
```

Referencing object-level structures

You reference object-level structures in scripts for the object itself exactly as you do global structures: declare a variable of the structure type, then use dot notation:

structure.variable

Example   Assume that the structure str_custdata is defined for the window
w_history and you are writing a script for a CommandButton in the window. To
use the structure definition in the script, you write:

```
str_custdata lstr_cust1
lstr_cust1.name = "Joe"
```

No access to object-level structures outside the object
You cannot make object-level structures accessible outside the object because
object-level structures are implicitly private.

Copying structures

❖   **To copy the values of a structure to another structure of the same type:**

•   Assign the structure to be copied to the other structure using this syntax:

    struct1 = struct2

    PowerBuilder copies all the variable values from *struct2* to *struct1*.

Example   These statements copy the values in lstr_emp2 to lstr_emp1:

```
str_empdata lstr_emp1, lstr_emp2
...
lstr_emp1 = lstr_emp2
```

Using structures with functions

You can pass structures as arguments in user-defined functions. Simply name
the structure as the datatype when defining the argument. Similarly,
user-defined functions can return structures. Name the structure as the return
type for the function.

You can also define external functions that take structures as arguments.

Example   Assume the following:

•   Revise is an external function that expects a structure as its argument.

•   lstr_empdata is a declared variable of a structure datatype.

You can call the function as follows:

```
Revise(lstr_empdata)
```

Declare the function first
The external function must be declared before you can reference it in a script.

For more about passing arguments to external functions, see *Application Techniques*.

Displaying and pasting structure information

You can display the names and variables of defined structures in the Browser. You can also paste these entries into a script.

❖ **To display information about a global structure in the Browser:**

1 Select the Structure tab and select a structure.

2 Double-click the properties folder in the right pane.

The properties folder expands to show the structure variables as properties of the structure.

❖ **To display information about an object-level structure in the Browser:**

1 Select the tab for the type of object for which the structure is defined.

2 Select the object that contains the structure.

3 Double-click the structure folder in the right pane.

The structure folder expands to display the structure variables using dot notation.

❖ **To paste the information into a script:**

1 Scroll to the structure variable you want to paste.

2 Select Copy from the variable's pop-up menu.

3 Insert the cursor in the script where you want to paste the variable and select Paste from the pop-up menu.

The variable name displays at the insertion point in the script.

P A R T 4

Working with Windows

This part describes how to create windows for your
application. It covers the properties of windows, the
controls you can place in windows, how to use inheritance
to save time and effort, and how to define menus. It also
introduces user objects.

Working with Windows

About this chapter

This chapter describes how to build windows in the Window painter.

Contents

About windows

Windows form the interface between the user and a PowerBuilder application. Windows can display information, request information from a user, and respond to the user's mouse or keyboard actions.

A window consists of:

- Properties that define the window's appearance and behavior

  For example, a window might have a title bar or a minimize box.

- Events

  Windows have events like other PowerBuilder objects.

- Controls placed in the window

At the window level

When you create a window, you specify its properties in the Window painter's Properties view. You can also dynamically change window properties in scripts during execution.

You can write scripts for window events that specify what happens when a window is manipulated. For example, you can connect to a database when a window is opened by coding the appropriate statements in the script for the window's Open event.

At the control level   You place PowerBuilder controls, such as CheckBox, CommandButton, or MultiLineEdit controls, in the window to request and receive information from the user and to present information to the user.

After you place a control in the window, you can define the style of the control, move and resize it, and build scripts to determine how the control responds to events.

Designing windows

The Microsoft Windows operating environment has certain standards that graphical applications are expected to conform to. Windows, menus, and controls are supposed to look and behave in predictable ways from application to application.

This chapter describes some of the guidelines you should follow when designing windows and applications, but a full discussion is beyond the scope of this book. You should acquire a book that specifically addresses design guidelines for applications on the Windows platform and apply the rules when you use PowerBuilder to create your application.

Building windows

When you build a window, you:

- Specify the appearance and behavior of the window by setting its properties

- Add controls to the window

- Build scripts that determine how to respond to events in the window and its controls

   To support these scripts, you can define new events for the window and its controls, and declare functions, structures, and variables for the window.

Two ways

There are two ways to build a window. You can:

- Build a new window from scratch

    You use this technique to create windows that are not based on existing windows.

- Build a window that inherits its style, events, functions, structures, variables, and scripts from an existing window

    You use inheritance to create windows that are derived from existing windows, thereby saving you time and coding.

For more information

For information on building windows from scratch, see "Building a new window" on page 231.

For information on using inheritance to build a window, see "Using inheritance to build a window" on page 246.

Types of windows

PowerBuilder provides the following types of windows: main, pop-up, child, response, Multiple Document Interface (MDI) frame, and MDI frame with MicroHelp.

Main windows

Main windows are standalone windows that are independent of all other windows. They can overlap other windows and can be overlapped by other windows.

You use a main window as the anchor for your application. The first window your application opens is a main window unless you are building a Multiple Document Interface (MDI) application, in which case the first window is an MDI frame.

For more on building MDI applications, see *Application Techniques*.

Using main windows

Define your independent windows as main windows. For example, assume that your application contains a calculator or scratch pad window that you want to have always available to the user. Make it a main window, which can be displayed at any time anywhere on the screen. As a main window, it can overlap other windows on the screen.

Pop-up windows

Pop-up windows are typically opened from another window, which in most cases becomes the pop-up window's parent.

Using the application's Open event
If you open a pop-up window from the application's Open event, the pop-up window does not have a parent and works the same way a main window works.

A pop-up window can display outside its parent window. It cannot be overlaid by its parent. A pop-up window is hidden when its parent is minimized and when its parent is closed. When you minimize a pop-up window, the icon for the window displays at the bottom of the desktop.

Using pop-up windows

Pop-up windows are often used as supporting windows. For example, say you have a window containing master information, such as film listings. You can use a pop-up window to allow a user to see details of a particular entry.

Explicitly naming a parent

In most cases, the window that opens a pop-up window becomes that window's parent. For example, if a script in w_go has this statement, w_go is the parent of w_popup:

```
Open(w_popup)
```

You can also explicitly name a pop-up window's parent when you use Open in this way:

Open (*popupwindow, parentwindow*)

For example, the following statement opens w_popup and makes w_parent its parent:

```
Open(w_popup, w_parent)
```

However, there are also other considerations regarding which window becomes the parent of an opened window.

For more information, see the Open function in the *PowerScript Reference*.

Child windows

Child windows are always opened from within a main or pop-up window, which becomes the child window's parent.

A child window exists only within its parent. You can move the child window within the parent window, but not outside the parent. When you move a portion of a child window beyond the parent, PowerBuilder clips the child so that only the portion within the parent window is visible. When you move the parent window, the child window moves with the parent and maintains the same position relative to the parent.

Child windows cannot have menus and are never considered the active window. They can have title bars and can be minimizable, maximizable, and resizable. When they are maximized, they fill the space of their parent; when they are minimized, their icon displays at the bottom of their parent.

The initial position of the child is relative to the parent and not to the entire screen. A child window closes when you close its parent.

Using child windows

Child windows are used by the PowerBuilder window plug-in and PowerBuilder window ActiveX tools. These are tools you can use when you are building applications for the Internet. You will probably not use child windows very often in other applications. Typically, if you want to display windows inside other windows, you will write MDI applications, where much of the window management happens automatically.

For more on the plug-in and ActiveX tools and on building MDI applications, see *Application Techniques*.

Response windows

Response windows request information from the user. They are always opened from within another window (its parent). Typically, a response window is opened after some event occurs in the parent window.

Response windows are application modal. That is, when a response window displays, it is the active window (it has focus) and no other window in the application is accessible until the user responds to the response window. The user *can* go to other applications, but when the user returns to the application, the response window is still active. Response windows act like modal pop-up windows.

Using response windows	For example, if you want to display a confirmation window when a user tries to close a window with unsaved changes, use a response window. The user is not allowed to proceed until the response window is closed.
Using message boxes	PowerBuilder also provides message boxes, which are predefined windows that act like response windows in that they are application modal. You open message boxes using the PowerScript MessageBox function.
	For more information, see MessageBox in the *PowerScript Reference*.

MDI frames

An MDI window is a frame window in which you can open multiple document windows (sheets) and move among the sheets. There are two types of MDI frame windows: MDI frame and MDI frame with MicroHelp.

For more on building MDI applications, see *Application Techniques*.

About the Window painter

Views in the Window painter	You design windows in the Window painter. The Window painter has several views where you specify how a window looks and how it behaves. The Window painter looks similar to the User Object painter for visual user objects and it has the same views. For details about the views, how you use them, and how they are related, see "Views in painters that edit objects" on page 116.
Window painter workspace	The default layout for the Window painter workspace has two stacked panes with the Script and Properties views at the top of the stacks.
	Most of your work in the Window painter is done in three views:

- The Layout view, where you design the appearance of the window

- The Properties view, where you set window properties and control properties

- The Script view, where you modify behavior by coding window and control scripts

This illustration shows the Layout view at the top of one of the stacks.
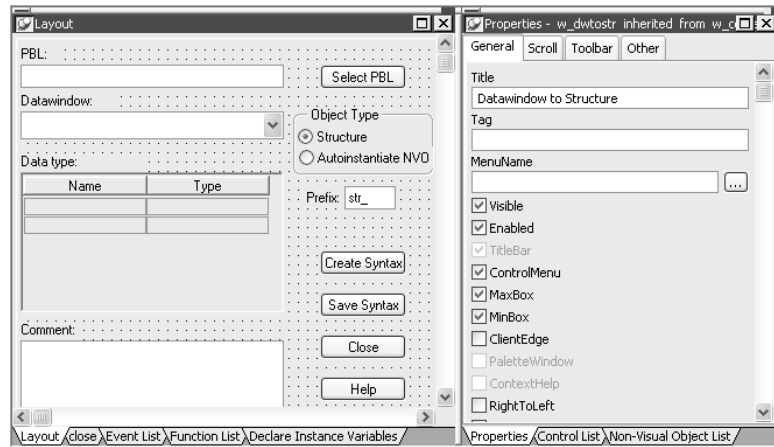
For information about specifying window properties, see "Defining the window's properties" on page 232.

For information about adding controls and nonvisual objects to a window, see "Adding controls" on page 238 and "Adding nonvisual objects" on page 239.

For information about coding in the Script view, see "Writing scripts in windows" on page 242 and Chapter 7, "Writing Scripts."

Building a new window

This section describes how to build windows from scratch. You use this technique to create windows that are not based on existing windows.

Creating a new window

❖ **To create a new window:**

1   Open the New dialog box.

2   On the PB Object tab page, select Window.

3   Click OK.

The Window painter opens. The new window displays in the Window painter's Layout view and its default properties display in the Properties view.

Defining the window's properties

Every window and control has a style that determines how it appears to the user. You define a window's style by choosing settings in the Window painter's Properties view. A window's style encompasses its:

Type
Basic appearance
Initial position on the screen
Icon when minimized
Pointer

About defining a window's style
When you define a window's style in the Window painter, you are actually assigning values to the properties for the window. You can programmatically change a window's style during execution by setting its properties in scripts. For a complete list of window properties, see *Objects and Controls*.

❖ **To specify window properties:**

1   Click the window's background to display the window's properties in the Properties view.

Another way to display window properties
You can also select the window name in the Control List view.

---

2   Choose the tab appropriate to the property you want to specify:

To specify the window's	Choose this tab
Name, type, state, color, and whether a menu is associated with it	General
Icon to represent the window when you minimize it	General
Position and size when it displays at execution time	Other
Default cursor whenever the mouse moves over the window	Other
Horizontal and vertical scroll bar placement	Scroll
Toolbar placement	Toolbar

Using the General property page

Use the General property page to specify the following window information:

Window type
Title bar text
Menu name
Color

Specifying the
window's type

The first thing you should do is specify the type of window you are creating.

❖ **To specify the window's type:**

1   In the Properties view for the window, select the General tab.

2   Scroll down the property page and select the appropriate window type
from the WindowType drop-down list.
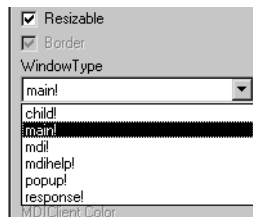
Depending on the type of window, PowerBuilder enables or disables certain
check boxes that specify other properties of the window. For example, if you
are creating a main window, the Title Bar check box is disabled. Main windows
always have title bars, so you cannot clear the Title Bar check box.

Specifying other basic
window properties

By selecting and clearing check boxes on the General property page, you can
specify whether the window is resizable or minimizable, is enabled, has a
border, and so on.

Note the following:

•   A main window must have a title bar.

•   A child window cannot have a menu.

•   A response window cannot have a menu, Minimize box, or Maximize box

.

Associating a menu
with the window

Many of your windows will have a menu associated with them.

❖ **To associate a menu with the window:**

1 Do one of the following:

- Enter the name of the menu in the Menu Name text box on the General property page

- Click the Browse button and select the menu from the Select Object dialog box, which displays a list of all menus available to the application

2 Click the Preview button in the PainterBar to see the menu.

For information about preview, see "Viewing your work" on page 240.

Changing the menu
You can change a menu associated with a window during execution using the
ChangeMenu function. For more information, see the *PowerScript Reference*.

Choosing a window
color

You can change the background color of your window.

❖ **To specify the color of a window:**

- Do one of the following:

- Specify the color of the window from the BackColor drop-down list on the General property page.

- If the window is an MDI window, specify a color in the MDI Client Color drop-down list.

Changing default
window colors

For main, child, pop-up, and response windows, the default color is ButtonFace if you are defining a 3D window, and white if you are not. If you or the user specified different display colors in the Windows Control Panel, a 3D window will display in the color that is set for the window background.

You can change the default for windows that are not 3D in the Application painter Properties view. To do so, click the Additional Properties button on the General page and modify the Background color on the Text Font tab page. New windows that are not 3D will have the new color you specified.

For more about using colors in windows, including how to define your own custom colors, see Chapter 12, "Working with Controls."

Choosing the window icon

If the window can be minimized, you can specify an icon to represent the minimized window. If you do not choose an icon, PowerBuilder uses the application icon for the minimized window.

❖ **To choose the window icon:**

1   Click the window's background so the Properties view displays window properties.

2   Select the General tab.

3   Choose the icon from the Icon drop-down list or use the Browse (...) button to select an icon (*.ICO*) file.

The icon you chose displays in the Icon list.

Changing the icon during execution
You can change the window icon during execution by assigning in code the name of the icon file to the window's Icon property, window.Icon.

Choosing the window's size and position

❖ **To resize a window in the Layout view:**

•   Drag the edge of the window in the Window painter's Layout view.

Resizing a window is easiest using the Layout view, but you can also change the window's width and height properties in the Properties view.

❖ **To specify a window's position and size:**

1   Click the window's background so the Properties view displays window properties.

2   Select the Other tab.

3   Enter values for x and y locations in PowerBuilder units.

About x and y values
For main, pop-up, response, and MDI frame windows, x and y locations are relative to the upper-left corner of the screen. For child windows, x and y are relative to the parent.

---

4   Enter values for width and height in PowerBuilder units.

The size of the window changes in the Layout view.

5   To see the position of the window, click the Preview button in the
     PainterBar (not the Preview button on the PowerBar).

6   To return to PowerBuilder, close the window.

     For information about preview, see "Viewing your work" on page 240.

About PowerBuilder
units

All window measurements are in PowerBuilder units (PBUs). Using these
units, you can build applications that look similar on different resolution
screens. A PBU is defined in terms of logical inches. The size of a logical inch
is defined by your operating system as a specific number of pixels. The number
is dependent on the display device. Windows typically uses 96 pixels per
logical inch for small fonts and 120 pixels per logical inch for large fonts.

Almost all sizes in the Window painter and in scripts are expressed as
PowerBuilder units. The two exceptions are text size, which is expressed in
points, and grid size in the Window and DataWindow painters, which is in
pixels.

For more about PowerBuilder units, see the *PowerScript Reference*.

Choosing the window's pointer

The default pointer used when the mouse is over a window is an arrow. You can
change this default on the Other page in the properties view.

❖   **To choose the window pointer:**

1   Click the window's background so the Properties view displays window
     properties.

2   Select the Other tab.

3   At the bottom of the property page, choose the pointer from the Pointer
     drop-down list or use the Browse (...) button to select a cursor (*.CUR*) file.

Specifying the pointer for a control
You can specify the pointer that displays when the mouse is over an individual
control. Select the control to display the Properties view for the control, then
specify the Pointer property on the Other page.

Specifying window scrolling

If your window is resizable, it is possible that not all the window's contents will be visible during execution. In such cases, you should make the window scrollable by providing vertical and horizontal scroll bars. You do this on the Scroll property page.

By default, PowerBuilder controls scrolling when scroll bars are present. You can control the amount of scrolling.

❖ **To specify window scrolling:**

1 Click the window's background so the Properties view displays window properties.

2 Select the Scroll tab.

3 Indicate which scroll bars you want to display by selecting the HScrollBar and VScrollBar check boxes.

4 Specify scrolling characteristics as follows:

Option	Meaning
UnitsPerLine	The number of PowerBuilder units to scroll up or down when the user clicks the up or down arrow in the vertical scroll bar. When the value is 0 (the default), it scrolls 1/100 the height of the window.
UnitsPerColumn	The number of PowerBuilder units to scroll right or left when the user clicks the right or left arrow in the horizontal scroll bar. When the value is 0 (the default), it scrolls 1/100 the width of the window.
ColumnsPerPage	The number of columns to scroll when the user clicks the horizontal scroll bar itself. When the value is 0 (the default), it scrolls 10 columns.
LinesPerPage	The number of lines to scroll when the user clicks the vertical scroll bar itself. When the value is 0 (the default), it scrolls 10 lines.

Specifying toolbar properties

You can specify whether or not you want to display a menu toolbar (if the menu you associate with your window assigns toolbar buttons to menu objects) in your window. If you choose to display the toolbar, you can specify the location for it.

❖ **To specify toolbar properties:**

1 Click the window's background so the Properties view displays window properties.

2 Select the Toolbar tab.

3 To display the toolbar with your window, select the ToolbarVisible check box.

4 Set the location of the toolbar by selecting an alignment option from the ToolbarAlignment drop-down list.

5 If you choose Float as your toolbar alignment, you must set the following values:

   • X and Y coordinates for the toolbar

   • Width and Height for the toolbar

For more information about defining toolbars, see Chapter 14, "Working with Menus and Toolbars."

Adding controls

When you build a window, you place controls, such as CheckBox, CommandButton, and MultiLineEdit controls, in the window to request and receive information from the user and to present information to the user.

After you place a control in the window, you can define its style, move and resize it, and write scripts to determine how the control responds to events.

For more information, see Chapter 12, "Working with Controls."

Adding nonvisual objects

You can automatically create nonvisual objects in a window by inserting a nonvisual object in the window. You do this if you want the services of a nonvisual object available to your window. The nonvisual object you insert can be a custom class or standard class user object.

You insert a nonvisual object in a window in the same way you insert one in a user object. For more information, see "Using class user objects" on page 376.

Saving the window

You can save the window you are working on at any time.

❖   **To save a window:**

1   Select File>Save from the menu bar.

   If you have previously saved the window, PowerBuilder saves the new version in the same library and returns you to the Window painter workspace.

   If you have not previously saved the window, PowerBuilder displays the Save Window dialog box.

2   Name the window in the Windows text box (see below).

3   Type comments in the Comments text box to describe the window.

   These comments display in the Select Window window and in the Library painter. It is a good idea to use comments so you and others can easily remember the purpose of the window later.

4   Specify the library where you want to save the window.

5   Click OK.

Naming the window   The window name can be any valid PowerBuilder identifier of up to 40 characters. For information about PowerBuilder identifiers, see the *PowerScript Reference*.

A commonly used convention is to preface all window names with w_ and use a suffix that helps you identify the particular window. For example, you might name a window that displays employee data w_empdata.

Viewing your work

While building a window, you can preview it and print its definition.

Previewing a window

As you develop a window, you can preview its appearance from the Window painter. By previewing the window, you get a good idea of how it will look during execution.

Preview button on the PainterBar and the PowerBar
You can preview a window from the Window painter using the Preview button on the PainterBar or by clicking the Preview button on the PowerBar. When you use the Preview button on the PainterBar, you do not have to save the window first, but you cannot trigger events as described below. For information about previewing using the PowerBar button, see "Running a window" on page 245.

❖ **To preview a window:**

- Click the Preview button in the PainterBar (not the PowerBar), or select Design>Preview from the menu bar.

  PowerBuilder minimizes and the window displays with the properties you have defined, such as title bar, menu, Minimize box, and so on.

What you can do

While previewing the window, you can get a sense of its look and feel. You can:

- Move the window

- Resize it (if it is resizable)

- Maximize, minimize, and restore it (if these properties were enabled)

- Tab from control to control

- Select controls

What you cannot do

You cannot:

- Change properties of the window

  Changes you make while previewing the window, such as resizing it, are not saved.

- Trigger events

  For example, clicking a CommandButton while previewing a window does not trigger its Clicked event.

- Connect to a database

❖ **To return to the Window painter:**

- Do one of the following:

  - If the Window has a control menu, select Close from the control menu or click the Close button in the upper right corner of the window.

  - If the window is visible, shut down the process.

  - If the window is not visible, click PowerBuilder on the task bar and then click the Terminate button.

Printing a window's definition

You can print a window's definition for documentation purposes.

❖ **To print information about the current window:**

- Select File>Print from the menu bar.

  Information about the current window is sent to the printer specified in Printer Setup. The information sent to the printer depends on variables specified in the [Library] section of the PowerBuilder initialization file.

  Print settings
  You can view and change the print settings in the Library painter. Select any PowerBuilder object, then select Entry>Library Item>Print from the menu bar.

Writing scripts in windows

You write scripts for window and control events. To support these scripts, you can define:

- Window-level and control-level functions

- Instance variables for the window

About events for windows and controls

Windows have several events including Open, which is triggered when the window is opened (before it is displayed), and Close, which is triggered when the window is closed. For example, you might connect to a database and initialize some values in the window's Open event, and disconnect from a database in the Close event.

Each type of control also has its own set of events. Buttons, for example, have Clicked events, which trigger when a user clicks the button. SingleLineEdits and MultiLineEdits have Modified events, which trigger when the contents of the edit control change.

Defining your own events

You can also define your own events, called user events, for a window or control, then use the EVENT keyword to trigger your user event.

For example, assume that you offer the user several ways to update the database from a window, such as clicking a button or selecting a menu item. In addition, when the user closes the window, you want to update the database after asking for confirmation. You want the same type of processing to happen after different system events.

You can define a user event for the window, write a script for that event, and then everywhere you want that event triggered, use the EVENT keyword.

To learn how to use user events, see Chapter 9, "Working with User Events."

About functions for windows and controls

PowerBuilder provides built-in functions that act on windows and on different types of controls. You can use these functions in scripts to manipulate your windows and controls. For example, to open a window, you can use the built-in window-level function Open, or you can pass parameters between windows by opening them with the function OpenWithParm and closing them with CloseWithReturn.

You can define your own window-level functions to make it easier to manipulate your windows. For more information, see Chapter 8, "Working with User-Defined Functions."

About properties of windows and controls

In scripts, you can assign values to the properties of objects and controls to change their appearance or behavior. You can also test the values of properties to obtain information about the object.

For example, you can change the text displayed in a StaticText control when the user clicks a CommandButton, or use data entered in a SingleLineEdit to determine what information is retrieved and displayed in a DataWindow control.

To refer to properties of an object or control, use dot notation to identify the object and the property:

object.property

control.property

Unless you identify the object or control when you refer to a property, PowerBuilder assumes you are referring to a property of the object or control the script is written for.

The reserved word *Parent*
In the script for a window control, you can use the reserved word Parent to refer to the window containing the control. For example, the following line in a script for a CommandButton closes the window containing the button:

```
close(Parent)
```

It is easier to reuse a script if you use Parent instead of the name of the window.

All properties, events, and built-in functions for all PowerBuilder objects, including windows, and each type of control are described in *Objects and Controls*.

Declaring instance variables

Often, data needs to be accessible in several scripts within a window. For example, assume a window displays information about one customer. You might want several CommandButtons to manipulate the data, and the script for each button needs to know the customer's ID. There are several ways to accomplish this:

• Declare a global variable containing the current customer ID

    All scripts in the application have access to this variable.

• Declare an instance variable within the window

    All scripts for the window and controls in the window have access to this variable.

• Declare a shared variable within the window

    All scripts for the window and its controls have access to this variable. In addition, all other windows of the same type have access to the same variable.

When declaring variables, you need to consider what the scope of the variable is. If the variable is meaningful only within a window, declare it as a window-level variable, generally an instance variable. If the variable is meaningful throughout the entire application, make it a global variable.

For a complete description of the types of variables and how to declare them, see the *PowerScript Reference*.

Examples of statements

The following assignment statement in the script for the Clicked event for a CommandButton changes the text in the StaticText object st_greeting when the button is clicked:

```
st_greeting.Text = "Hello User"
```

The following statement tests the value entered in the SingleLineEdit sle_state and displays the window w_state1 if the text is "AL":

```
if sle_State.Text= "AL" then Open(w_state1)
```

Running a window

During development, you can test a window without running the whole application.

You can preview a window from the Window painter using the Preview button on the PainterBar or run the window by clicking the Preview button on the PowerBar. The PowerTip text for this button is `Run/Preview Object`. For information about previewing using the PainterBar button, see "Previewing a window" on page 240.

When you run the window using the PowerBar button, you must save the window first. You can also trigger events and open other windows because the window is functional.

❖ **To run a window:**

1   Click the Preview button in the PowerBar (not the PainterBar).

2   In the Run/Preview dialog box, select Windows as the Objects of Type.

3   Select the target that includes the window you want to run.

4   Select the library that includes the window.

5   Select the window you want to run and click OK.

   You must save your work before running a window. If you have not saved your work, PowerBuilder prompts you to do so.

   PowerBuilder runs the window.

You can trigger events, open other windows, connect to a database, and so on when running a window. The window is fully functional. It has access to global variables that you have defined for the application and to built-in global variables, such as SQLCA. The SystemError event is not triggered if there is an error, because SystemError is an Application object event.

❖ **To return to the Window painter:**

•   Do one of the following:

   •   If the Window has a control menu, select Close from the control menu or click the Close button in the upper right corner of the window.

   •   If the window is not visible, click PowerBuilder on the task bar and then click the Terminate button

   .

Using inheritance to build a window

When you build a window that inherits its definition—its style, events, functions, structures, variables, controls, and scripts—from an existing window, you save coding time. All you have to do is modify the inherited definition to meet the requirements of the current situation.

This section provides an overview of using inheritance in the Window painter. The issues concerning inheritance with windows are the same as the issues concerning inheritance with user objects and menus. They are described in more detail in Chapter 13, "Understanding Inheritance."

Building two windows with similar definitions

Assume your application needs two windows with similar definitions. One window, w_employee, needs:

- A title (Employee Data)

- Text that says Select a file:

- A drop-down list with a list of available employee files

- An Open button with a script that opens the selected file in a multiline edit box

- An Exit button with a script that asks the user to confirm closing the window and then closes the window

The window looks like this:
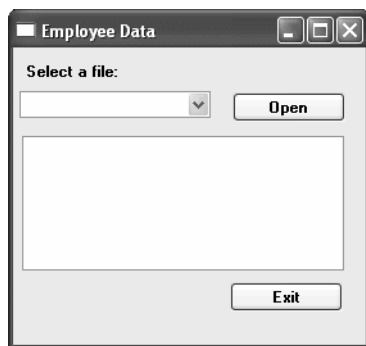
The only differences in the second window, w_customer, are that the title is Customer Data, the drop-down list displays customer files instead of employee files, and there is a Delete button so the user can delete files.

Your choices

To build these windows, you have three choices:

- Build two new windows from scratch as described in "Building a new window" on page 231

- Build one window from scratch and then modify it and save it under another name

- Use inheritance to build two windows that inherit a definition from an ancestor window

Using inheritance

To build the two windows using inheritance, follow these steps:

1   Create an ancestor window, w_ancestor, that contains the text, drop-down list, and the open and exit buttons, and save and close it.

   Note You cannot inherit a window from an existing window when the existing window is open, and you cannot open a window when its ancestor or descendant is open.

2   Select File>Inherit, select w_ancestor in the Inherit From dialog box, and click OK.

3   Add the Employee Data title, specify that the DropDownListBox control displays employee files, and save the window as w_employee.

4   Select File>Inherit, select w_ancestor in the Inherit From dialog box, and click OK.

5   Add the Customer Data title, specify that the DropDownListBox control displays customer files, add the Delete button, and save the window as w_customer.

Advantages of using inheritance

Using inheritance has a number of advantages:

- When you change the ancestor window, the changes are reflected in all descendants of the window. You do not have to make changes manually in the descendants as you would in a copy. This saves you coding time and makes the application easier to maintain.

- Each descendant inherits the ancestor's scripts, so you do not have to re-enter the code to add to the script.

- You get consistency in the code and in the application windows.

When you use inheritance to build an object, everything in the ancestor object is inherited in all its descendants. In the descendant, you can:

- Change the properties of the window

- Add controls to the window and modify existing controls

- Size and position the window and the controls in the window

- Build new scripts for events in the window or its controls

- Reference the ancestor's functions and events

- Reference the ancestor's structures if the ancestor contains a public or protected instance variable of the structure data type

- Access ancestor properties, such as instance variables, if the scope of the property is public or protected

- Extend or override inherited scripts

- Declare functions, structures, and variables for the window

- Declare user events for the window and its controls

The only thing you cannot do is delete inherited controls. If you do not need an inherited control, you can make it invisible in the descendent window.

Instance variables in descendants

If you create a window by inheriting it from an existing window that has public or protected instance variables with simple data types, the instance variables display and can be modified in the descendent window's Properties view. You see them at the bottom of the General tab page. In this illustration, the last two properties are inherited instance variables.
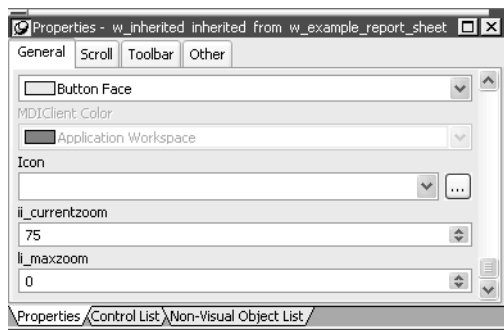
All public instance variables with simple data types such as integer, boolean, character, date, string, and so on display. Instance variables with the any or blob data type or instance variables that are objects or arrays do not display.

Control names in descendants

PowerBuilder uses this syntax to show names of inherited controls:

> *ancestorwindow***::***control*

For example, if you select the Open button in w_customer, which is inherited from w_ancestor, its name displays on the General page in the properties view as `w_ancestor::cb_open`.

Names of controls must be unique in an inheritance hierarchy. For example, you cannot have a CommandButton named cb_close defined in an ancestor and a different CommandButton named cb_close defined in a child. You should develop a naming convention for controls in windows that you plan to use as ancestors.

CHAPTER 12   **Working with Controls**

About this chapter   Users run your application primarily by interacting with the controls you place in windows. This chapter describes the use of controls.

Contents

About controls

About window controls   You place controls in a window to request and receive information from the user and to present information to the user. For a complete list of standard window controls, open a window in the Window painter and select Insert>Control.

If you often use a control or set of controls with certain properties, such as a group of related radio buttons, you can create a visual user object that contains the control or set of controls. For more about user objects, see Chapter 15, "Working with User Objects."

About events

All window controls have events so that users can act on the controls. You write scripts that determine the processing that takes place when an event occurs in the control.

Drawing controls are usually used only to make your window more attractive or to group controls. Only constructor and destructor events are defined for them, but you can define your own events if needed. The drawing controls are Line, Oval, Rectangle, and RoundRectangle.

Inserting controls in a window

You insert controls in a window in the Window painter.

❖ **To insert a control in a window:**

1   Select Insert>Control from the menu bar, or display the Controls drop-down toolbar on the PainterBar.

2   Select the control you want to insert.

If you select User Object, the Select Object dialog box displays listing all user objects defined for the application. Select the library and the user object and click OK.

3   In the Layout view, click where you want the control.

After you insert the control, you can size it, move it, define its appearance and behavior, and create scripts for its events.

Duplicating controls

To place multiple controls of the same type in a window, place a control in the window and make sure it is selected. Then press Ctrl+T or select Duplicate from the pop-up menu once for each duplicate control you want to place in the window. The controls are placed one under another. You can drag them to other locations if you want.

Inserting controls with undefined content

When you insert a DataWindow, Picture, PictureButton, or PictureHyperLink control in a window, you are inserting only the control. You see only an empty box for a DataWindow control, the dotted outline of a box for Picture and PictureHyperLink controls, and a large button resembling a CommandButton for a PictureButton control. You must specify a DataWindow object or picture later.

Dragging and dropping DataWindow objects
You can insert a DataWindow control with a predefined DataWindow object in a window by dragging the DataWindow object from the System Tree to the window's Layout view.

Placing OLE controls

You can place objects from applications that support OLE, such as Excel worksheets and Visio drawings, in your windows. For information about using OLE with PowerBuilder, see *Application Techniques*.

Selecting controls

You select controls so that you can change their properties or write scripts using the Layout view or the Control List view.

❖ **To select a control:**

• Click the control in the Layout view, or click the control in the Control List view.

In the Layout view, the control displays with handles on it. Previously selected controls are no longer selected.

Acting on multiple controls

You can act on all or multiple selected controls as a unit. For example, you can move all of them or change the fonts for all the text displayed in the controls.

❖ **To select multiple controls:**

• In the Layout or Control List view, click the first control and then press and hold the Ctrl key and click additional controls.

❖ **To select neighboring multiple controls:**

• In the Layout view, press the left mouse button, drag the mouse over the controls you want to select, and release the mouse button.

Selecting all controls
You can select all controls by selecting Edit>Select All from the menu bar.

Information displayed in the MicroHelp bar

The name, x and y coordinates, width, and height of the selected control are displayed in the MicroHelp bar. If you select multiple objects, `Group Selected` displays in the Name area and the coordinates and size do not display.

Defining a control's properties

Just like the window object, each control has properties that determine how the control looks and behaves at runtime (the control's style).

You define a control's properties by using the Properties view for the control. The properties and values displayed in the Properties view change dynamically when you change the selected object or control. To see this, click the window background to display the window properties in the Properties view and then click a control in the window to display the control's properties in the Properties view.

❖ **To define a control's properties:**

1   Select the control.

    The selected control's properties display in the Properties view.

2   Use the tab pages in the Properties view to change the control's properties.

About tab pages in the Properties view

The Properties view presents information in a consistent arrangement of tabbed property pages. You select items on the individual property pages to change the control's definition.

All controls have a General property page, which contains much of the style information (such as the visibility of the control, whether it is enabled, and so on) about the control. The General property page is always the first page.

Getting Help on properties

You can get Help when you are defining properties. In any tab page in the Properties view, right-click on the background and select Help from the pop-up menu. The Help displays information about the control with a link to an alphabetical list of properties for the control.

Naming controls

When you place a control in a window, PowerBuilder assigns it a unique name. The name is the concatenation of the default prefix for the control name and the lowest 1- to 4-digit number that makes the name unique.

For example, assume the prefix for ListBoxes is lb_ and you add a ListBox to the window:

- If the names lb_1, lb_2, and lb_3 are currently used, the default name is lb_4

- If lb_1 and lb_3 are currently used but lb_2 is not, the default name is lb_2

About the default prefixes

Each type of control has a default prefix for its name. Table 12-1 lists the initial default prefix for each control (note that there is no prefix for a window).

Table 12-1: Default prefixes for window control names

Control	Prefix
Animation	am_
CheckBox	cbx_
CommandButton	cb_
DataWindow	dw_
DatePicker	dp_
DropDownListBox	ddlb_
DropDownPictureListBox	ddplb_
EditMask	em_
Graph	gr_
GroupBox	gb_
HProgressBar	hpb_
HScrollBar	hsb_
HTrackBar	htb_
InkEdit	ie_
InkPicture	ip_
Line	ln_
ListBox	lb_
ListView	lv_
MonthCalendar	mc_
MultiLineEdit	mle_
OLE 2.0	ole_
Oval	ov_
Picture	p_
PictureHyperLink	phl_
PictureButton	pb_
PictureListBox	plb_
RadioButton	rb_
Rectangle	r_
RichTextEdit	rte_
RoundRectangle	rr_
SingleLineEdit	sle_
StaticText	st_

Control	Prefix
StaticHyperLink	shl_
Tab	tab_
TreeView	tv_
User Object	uo_
VProgressBar	vpb_
VScrollBar	vsb_
VTrackBar	vtb_

Changing the default prefixes

You can change the default prefixes for controls in the Window painter's Options dialog box. Select Design>Options from the menu bar to open the Options dialog box. The changes you make are saved in the PowerBuilder initialization file. For more about the PowerBuilder initialization file, see "How the PowerBuilder environment is managed" on page 57.

Changing the name

You should change the default suffix to a suffix that is meaningful in your application. For example, if you have command buttons that update and retrieve database information, you might call them cb_update and cb_retrieve. If you have many controls on a window, using intuitive names makes it easier for you and others to write and understand scripts for these controls.

Using application-based names instead of sequential numbers also minimizes the likelihood that you will have name conflicts when you use inheritance to create windows.

❖ **To change a control's name:**

1   Select the control to display the control's properties in the Properties view.

2   On the General tab page, select the application-specific suffix (for example, the 1 in the cb_1 command button name) and type a more meaningful one.

You can use any valid PowerBuilder identifier with up to 40 characters. For information about PowerBuilder identifiers, see the *PowerScript Reference*.

Changing text

You can specify the text and text display characteristics for a control in the Properties view for the control. You can also use the Window painter StyleBar to change:

- The text itself

- The font, point size, and characteristics such as bold

- The alignment of text within the control

CommandButton text
Text in CommandButtons is always center aligned.

The default text for most controls that have a text property is none. To display an empty StaticText or SingleLineEdit control, clear the Text box in the Properties view or the StyleBar.

When you add text to a control's text property, the width of the control changes automatically to accommodate the text as you type it in the StyleBar, or when you tab off the Text box in the Properties view.

❖ **To change text properties of controls:**

1 Select one or more controls whose properties you want to change.

2 Specify changes in the Font tab page in the Properties view, or specify changes using the StyleBar.

How text size is stored

A control's text size is specified in the control's TextSize property. PowerBuilder saves the text size in points, using negative numbers.

For example, if you define the text size for the StaticText control st_prompt to be 12 points, PowerBuilder sets the value of st_prompt's TextSize property to –12. PowerBuilder uses negative numbers to record point size for compatibility with previous releases, which saved text size in pixels as positive numbers.

If you want to change the point size of text at runtime in a script, remember to use a negative value. For example, to change the point size for st_prompt to 14 points, code:

```
st_prompt.TextSize = -14
```

You can specify text size in pixels if you want, by using positive numbers. The following statement sets the text size to be 14 pixels:

```
st_prompt.TextSize = 14
```

Moving and resizing controls

There are several ways to move and resize controls in the Layout view.

Moving and resizing controls using the mouse

To move a control, drag it with the mouse to where you want it.

To resize a control, select it, then grab an edge and drag the edge with the mouse.

Moving and resizing controls using the keyboard

To move a control, select it, then press an arrow key to move it in the corresponding direction.

To resize a control, select it, and then press:

- Shift+Right Arrow to make the control wider

- Shift+Left Arrow to make the control narrower

- Shift+Down Arrow to make the control taller

- Shift+Up Arrow to make the control shorter

Aligning controls using the grid

The Window painter provides a grid to help you align controls at design time. You can use the grid options to:

- Make controls snap to a grid position when you place them or move them in a window

- Show or hide the grid when the workspace displays

- Specify the height and width of the grid cells

❖ **To use the grid:**

1   Choose Design>Options from the menu bar and select the General tab.

2   Do one or more of the following:

   •   Select Snap to Grid to align controls with a horizontal and vertical grid when you place or move them

   •   Select Show Grid to display the grid in the Layout view

   •   Specify the width of each cell in the grid in pixels in the X text box

   •   Specify the height of each cell in the grid in pixels in the Y text box

Hiding the grid
Window painting is slower when the grid is displayed, so you might want to display the grid only when necessary.

Aligning controls with each other

You can align selected controls by their left, right, top, or bottom edges or their horizontal or vertical centers.

PainterBars in the Window painter
The Window painter has three PainterBars. PainterBar1 includes buttons that perform operations that are common to many painters, including save, cut, copy, paste, and close. PainterBar2 includes buttons used with the Script view. PainterBar3 contains buttons that manipulate the display of the selected control or controls. The tools used to align, resize, and adjust the space between controls are on a drop-down toolbar on PainterBar3.

❖ **To align controls:**

1   Select the control whose position you want to use to align the others.

   PowerBuilder displays handles around the selected control.

2   Press and hold the Ctrl key and click the controls you want to align with the first one.

   All the selected controls have handles on them.

3   Select Format>Align from the menu bar, or select the Layout drop-down toolbar in PainterBar3.

4   Select the dimension along which you want to align the controls.

PowerBuilder aligns all the selected controls with the first control selected.

Equalizing the space between controls

You can manually move controls by dragging them with the mouse. You can also equalize the space around selected controls using the Format menu or the Layout drop-down toolbar.

❖   **To equalize the space between controls:**

1   Select the two controls whose spacing is correct.

To do so, select one control, then press and hold Ctrl and click the second control.

2   Press Ctrl and click to select the other controls whose spacing should match that of the first two controls.

3   Select Format>Space from the menu bar, or select the Layout drop-down toolbar in PainterBar3.

4   Select horizontal or vertical spacing.

Equalizing the size of controls

Using the Format menu or the Layout drop-down toolbar, you can adjust selected controls so that they are all the same size as the first control selected. You might do this if you have several SingleLineEdit or CommandButton controls on a window.

❖   **To equalize the size of controls:**

1   Select the control whose size is correct.

2   Press and hold Ctrl and click to select the other controls that should be the same size as the first control.

3   Select Format>Size from the menu bar, or select the Layout drop-down toolbar in PainterBar3.

4   Select the size for width, height, or both width and height.

Copying controls

You can copy controls within a window or to other windows. All properties of the control, as well as all of its scripts, are copied. You can use this technique to make a copy of an existing control and change what you want in the copy.

❖ **To copy a control:**

1   Select the control.

2   Select Edit>Copy from the menu bar or press Ctrl+C.

    The control is copied to a private PowerBuilder clipboard.

3   Do one of the following:

    •   To copy the control within the same window, select Edit>Paste Controls from the menu bar or press Ctrl+V.

    •   To copy the control to another window, click the Open button in the PowerBar and open the window in another instance of the Window painter. Make that window active and select Edit>Paste Controls from the menu bar or press Ctrl+V.

        If the control you are pasting has the same name as a control that already exists in the window, the Paste Control Name Conflict dialog box displays.

4   If prompted, change the name of the pasted control to be unique.

    PowerBuilder pastes the control in the destination window at the same location as in the source window. If you are pasting into the same window, move the pasted control so it does not overlay the original control. You can make whatever changes you want to the copy; the source control will be unaffected.

Defining the tab order

When you place controls in a window, PowerBuilder assigns them a default tab order, the default sequence in which focus moves from control to control when the user presses the Tab key.

Tab order in user objects
When the user tabs to a custom user object in a window and then presses the Tab key, focus moves to the next control in the tab order for the user object. After the user tabs to all the controls in the tab order for the user object, focus moves to the next control in the window tab order.

Establishing the default tab order

PowerBuilder uses the relative positions of controls in a window to establish the default tab order. It looks at the positions in the following order:

- The distance of the control from the top of the window (Y)

- The distance of the control from the left edge of the window (X)

The control with the smallest Y distance is the first control in the default tab order. If multiple controls have the same Y distance, PowerBuilder uses the X distance to determine the tab order among them.

Default tab values
The default tab value for drawing objects and RadioButtons in a GroupBox is 0, which means the control is skipped when the user tabs from control to control.

When you add a control to the window, PowerBuilder obtains the tab value of the control that precedes the new control in the tab order and assigns the new control the next number.

For example, if the tab values for controls A, B, and C are 30, 10, and 20 respectively and you add control D between controls A and B, PowerBuilder assigns control D the tab value 40.

Changing the window's tab order

❖ **To change the tab order:**

1   Select Format>Tab Order from the menu bar, or click the Tab Order button on PainterBar1 (next to the Preview button).

The current tab order displays. If this is the first time you have used Tab Order for the window, the default tab order displays.

2   Use the mouse or the Tab key to move the pointer to the tab value you want to change.

3   Enter a new tab value from 0 to 9999.

The value 0 removes the control from the tab order. It does not matter exactly what value you use, other than 0. Only the relative value is significant. For example, if you want the user to tab to control B after control A but before control C, set the tab value for control B so it is between the value for control A and the value for control C.

Tab tips
A tab order value of 0 does not prevent a control from being selected or activated or from receiving keyboard events. To prevent a user from activating a control with the mouse, clear the Enabled check box on its General property page.

To permit tabbing in a group box, change the tab value of the GroupBox control to 0, then assign nonzero tab values to the controls in the group box.

4   Repeat the procedure until you have the tab order you want.

5   Select Format>Tab Order or the Tab Order button again.

PowerBuilder saves the tab order.

Each time you select Tab Order, PowerBuilder renumbers the tab order values to include any controls that have been added to the window and to allow space to insert new controls in the tab order. For example, if the original tab values for controls A, B, and C were 10, 20, and 30, and you insert control D between A and B and give it a tab value of 15, when you select tab order again, the controls A, B, and C will have the tab values 10, 30, and 40, and control D will have the tab value 20.

Defining accelerator keys

You can define accelerator keys for controls to allow users to change focus from one control to another. An accelerator key is sometimes referred to as a mnemonic access key.

Users press Alt followed by the accelerator key to use an accelerator. If the currently selected control is not an editable control (such as a SingleLineEdit, MultiLineEdit, ListBox, or DropDownListBox), users only have to press the accelerator key. They do not need to press the Alt key.

How you define accelerator keys depends on whether the type of control has displayed text associated with it. If there is no displayed text, you must define the accelerator key in the control itself and in a label that identifies the control.

❖ **To define an accelerator key for a CommandButton, CheckBox, or RadioButton:**

1   Click the control to display the control's properties in the Properties view.

2   In the Text box on the General page, precede the letter that you want to use as the accelerator key with an ampersand character (&).

When you perform your next action (such as tab to the next property or select the window or a control in the Layout view), the property is set and PowerBuilder displays an underline to indicate the accelerator key.

Displaying an ampersand
If you want to display an ampersand character in the text of a control, type a double ampersand. The first ampersand acts as an escape character.

❖ **To define an accelerator key for a SingleLineEdit, MultiLineEdit, ListBox, or DropDownListBox:**

1   Click the control to display the control's properties in the Properties view.

2   In the General tab page, type the letter of the accelerator key in the Accelerator box.

For example, if the control contains a user's name and you want to make Alt+N the accelerator for the control, type n in the Accelerator box.

At this point you have defined the accelerator key, but the user has no way of knowing it, so you need to label the control.

3   Place a StaticText control next to the control that was assigned the accelerator key.

4   Click the StaticText control to display its properties in the Properties view.

5   In the Text box on the General page, precede the letter that you want to use as the accelerator key with an ampersand character (&).

For example, if the StaticText control will display the label Name, type &Name in the Text box so that the letter N is underlined. Now your user knows that there is an accelerator key associated with the control.

Specifying accessibility of controls

Controls have two boolean properties that affect accessibility of the control:

• Visible

• Enabled

Using the Visible property

If the Visible property of a control is selected, the control displays in the window. If you want a control to be initially invisible, be sure the Visible property is not selected in the General tab page in the control's Properties view.

Hidden controls do not display by default in the Window painter's Layout view.

❖ **To display hidden controls in the Layout view:**

• Select Design>Show Invisibles from the menu bar.

To display a control at runtime, assign the value true to the Visible property:

*controlname.*Visible = TRUE

Using the Enabled property

If the Enabled property is selected, the control is active. For example, an enabled CommandButton can be clicked, a disabled CommandButton cannot.

If you want a control to display but be inactive, be sure the Enabled property is not selected in the General tab page in the control's Properties view. For example, a CommandButton might be active only after the user has selected an option. In this case, display the CommandButton initially disabled so that it appears grayed out. Then, when the user selects the option, enable the CommandButton in a script:

*CommandButtonName.*Enabled = TRUE

Choosing colors

The Window painter has two Color drop-down toolbars on PainterBar3 that display colors that you can use for the background and foreground of components of the window. Initially, the drop-down toolbars display these color selections:

• 20 predefined colors

• 16 custom colors (labeled C)

• The full set of Windows system colors

Windows system colors

The Windows system colors display in the same order as in the TextColor and BackColor lists in the Properties view for a control. They are labeled with letters that indicate the type of display element they represent:

• W for windows

• T for text in windows, title bars, menus, buttons, and so on

• A for the application workspace

- • B for button face, highlight, shadows, and borders

- • S for scroll bars

- • D for the desktop

- • M for menu bars

- • F for window frames

- • H for highlight

- • L for links

The Windows system colors are those defined by the user in the Windows Control Panel, so if you use these colors in your window, the window colors will change to match the user's settings at runtime.
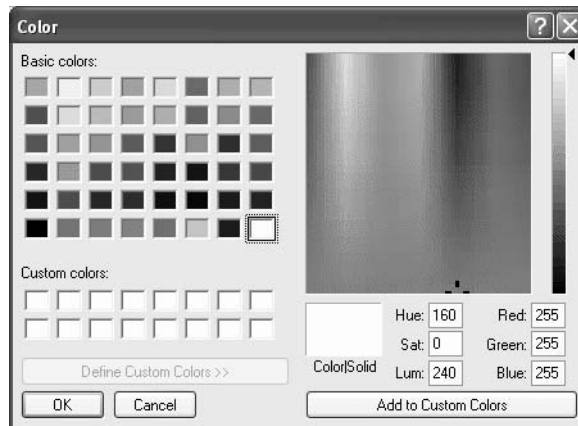
Defining custom colors

You can define your own custom colors for use in windows, user objects, and DataWindow objects.

❖ **To define and maintain custom colors:**

1   Select Design>Custom Colors from the menu bar.

The Color dialog box displays.

2   Click in an empty color box in the list of custom colors.

3   Choose an existing color or create the color you want. You can start with one of the basic colors and customize it in the palette to the right by dragging the color indicator with the mouse. You can also specify precise values to define the color.

4   When you have the color you want, click Add to Custom Colors.

The new color displays in the list of custom colors.

5   Select the new color in the list of custom colors.

6   Click OK.

The new color displays in the Color drop-down toolbars and is available in all windows, user objects, and DataWindow objects you create.

PowerBuilder saves custom colors in the [Colors] section of the PowerBuilder initialization file, so they are available across sessions.

Specifying foreground and background colors

You can assign colors to controls using the Painterbar or the Properties view. The page in the Properties view that you use depends on the control. For some controls you can change only the background color, and for others you can change neither the foreground nor the background color. These controls include CommandButton, PictureButton, PictureHyperLink, Picture, Scrollbar, TrackBar, ProgressBar, and OLE controls.

❖   **To assign a color using the PainterBar:**

1   Select the control.

2   Select either the foreground or background color button from the PainterBar.

3   Select a color from the drop-down toolbar.

Using the 3D look

Applications sometimes have a three-dimensional look and feel. To use this appearance for an application, select a 3D border for your SingleLineEdit boxes and other controls and make the window background gray.

❖   **To use the 3D look by default:**

1   Select Design>Options from the menu bar.

The Options dialog box displays.

2   On the General property page, select Default to 3D.

When you build a new window, PowerBuilder automatically sets the window background color to gray and uses 3D borders when you place controls.

PowerBuilder records this preference in the Default3D variable in the [Window] section of the PowerBuilder initialization file, so the preference is maintained across sessions.

Mapping 3D colors for pictures

You can make the background of Picture, PictureHyperlink, and PictureButton controls blend in with the background of your window. This applies to whatever color scheme the user has selected on the Appearance page of the Display Properties dialog box in the Windows Control Panel.

Use this feature if you want to place a control containing a picture on a window and have the picture blend in with the background color of the window when the window's background is using Button Face for a 3D effect. The control's picture takes on the 3D colors the user has selected.

The window's background must be set to Button Face. To make the image blend in with the window, give it a background color in the range between RGB(160,160,160) and RGB(223,223,223), such as silver. Lighter shades of gray map to the button highlight color and darker shades to the button shadow color.

This option can affect other colors used in the bitmap. It does not affect the control's border settings, and it has no effect if there is no image associated with the control.

Using the individual controls

There are four basic types of controls with different purposes.

Table 12-2: Summary of control types by function

Function	Controls include
Invoke actions	CommandButtons, PictureButtons, PictureHyperLinks, StaticHyperLinks, Tabs, User Objects
Display or accept data, or both	ListBoxes, PictureListBoxes, DropDownListBoxes, DropDownPictureListBoxes, DataWindow controls, StaticText, ListViews, TreeViews, RichTextEdit, Graphs, Pictures, ProgressBars, ScrollBars, SingleLineEdits, MultiLineEdits, EditMasks, Tabs, user objects, OLE controls, MonthCalendar, DatePicker, InkEdit, and InkPicture controls
Indicate choices	RadioButtons, CheckBoxes (you can group these controls in a GroupBox), TrackBars
Decorative	Line, Rectangle, RoundRectangle, Oval, Animation

How to use the controls

You should use the controls only for the purpose shown in the table. For example, users expect radio buttons for selecting an option. Do not use a radio button also to invoke an action, such as opening a window or printing. Use a command button for that.

There are, however, several exceptions: user objects can be created for any purpose, and ListBoxes, ListViews, TreeViews, and Tabs are often used both to display data and to invoke actions. For example, double-clicking a ListBox item often causes some action to occur.

Individual controls

The following sections describe some features that are unique to individual controls. The controls are listed in the order in which they display on the Insert>Control menu and the drop-down controls palette:

- "CommandButton" on page 271

- "PictureButton" on page 272

- "CheckBox" on page 273

- "RadioButton" on page 274

- "StaticText" on page 275

- "StaticHyperLink" on page 276

- "Picture" on page 276

- "PictureHyperLink" on page 277

- "GroupBox" on page 277

- "Drawing controls" on page 278

- "SingleLineEdit and MultiLineEdit" on page 278

- "EditMask" on page 279

- "HScrollBar and VScrollBar" on page 281

- "HTrackBar and VTrackBar" on page 282

- "HProgressBar and VProgressBar" on page 283

- "DropDownListBox" on page 283

- "DropDownPictureListBox" on page 284

- "ListBox" on page 285

- "PictureListBox" on page 286

- "ListView" on page 288

Some controls are not covered in this chapter:

- DataWindow controls and objects. See Chapter 18, "Defining DataWindow Objects."

- RichTextEdit controls. See Chapter 30, "Working with Rich Text."

- User objects. See Chapter 15, "Working with User Objects."

- Graph controls. See Chapter 26, "Working with Graphs."

- OLE controls. See Chapter 31, "Using OLE in a DataWindow Object."

CommandButton

CommandButtons are used to carry out actions. For example, you can use an OK button to confirm a deletion or a Cancel button to cancel a requested deletion. If there are many related CommandButtons, place them along the right side of the window; otherwise, place them along the bottom of the window.

You cannot change the color or alignment of text in a CommandButton.

If clicking the button opens a window that requires user interaction before any other action takes place, use ellipsis points in the button text; for example, "`Print...`".

Specifying Default and Cancel buttons

You can specify that a CommandButton is the default button in a window by selecting Default in the General property page in the button's Properties view.

When there is a default CommandButton and the user presses the Enter key:

• If the focus is *not* on another CommandButton, the default button's Clicked event is triggered

• If the focus is on another CommandButton, the Clicked event of the button with focus is triggered

---

Other controls affect default behavior
If the window does not contain an editable field, use the SetFocus function or the tab order setting to make sure the default button behaves as described above.

A bold border is placed around the default CommandButton (or the button with focus if the user explicitly tabs to a CommandButton).

You can define a CommandButton as being the cancel button by selecting Cancel in the General property page in the button's Properties view. If you define a cancel CommandButton, the cancel button's Clicked event is triggered when the user presses the Esc key.

PictureButton

PictureButtons are identical to CommandButtons in their functionality. The only difference is that you can specify a picture to display on the button. The picture can be a bitmap (BMP) file, a run-length encoded (RLE) file, an Aldus-style Windows metafile (WMF), a GIF file, an animated GIF file, or a JPEG file.

You can choose to display one picture if the button is enabled and a different picture if the button is disabled.

Use these controls when you want to be able to represent the purpose of a button by using a picture instead of text.

❖ **To specify a picture:**
1 Select the PictureButton to display its properties in the Properties view.

2 On the General tab page, enter the name of the image file you want to display when the button is enabled, or use the Browse button and choose a file.

3   Enter the name of the image file you want to display when the button is disabled, or use the Browse Disabled button and choose a file.

If the PictureButton is defined as initially enabled, the enabled picture displays in the Layout view. If the PictureButton is defined as initially disabled, the disabled picture displays in the Layout view.

❖   **To specify button text alignment:**

1   Select the PictureButton to display its properties in the Properties view.

2   On the General tab page, enter the text for the PictureButton in the Text box.

3   Use the HTextAlign and VTextAlign lists to choose how you want to display the button text.

CheckBox

CheckBoxes are square boxes used to set independent options. When they are selected, they contain a check mark; when they are not selected, they are empty.

CheckBoxes are independent of each other. You can group them in a GroupBox or rectangle to make the window easier to understand and use, but that does not affect the CheckBoxes' behavior; they are still independent.

Using three states   CheckBoxes usually have two states: on and off. But sometimes you need to represent a third state, such as Unknown. The third state displays as a grayed box with a check mark.

❖ **To enable the third state:**

- Select the ThreeState property in the General page of the CheckBox Properties view.

❖ **To specify that a CheckBox's current state is the third state:**

- Select the ThreeState and the ThirdState properties in the General page of the CheckBox Properties view.

RadioButton

RadioButtons are round buttons that represent mutually exclusive options. They always exist in groups. Exactly one RadioButton is selected in each group.

When a RadioButton is selected, it has a dark center; when it is not selected, the center is blank.

In the following example, the text can be either plain, bold, or italic (plain is selected):

When the user clicks a RadioButton, it becomes selected and the previously selected RadioButton in the group becomes deselected.

Use RadioButtons to represent the state of an option. Do not use them to invoke actions.

When a window opens, one RadioButton in a group must be selected. You specify which is the initially selected RadioButton by selecting the Checked property in the General property page in the RadioButton's Properties view.

Grouping
RadioButtons

By default, all RadioButtons in a window are in one group, no matter what their location in the window. Only one RadioButton can be selected at a time.

You use a GroupBox control to group related RadioButtons. All RadioButtons inside a GroupBox are considered to be in one group. One button can be selected in each group.
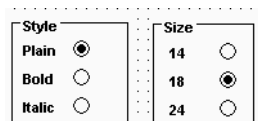
| The Automatic property | When a window contains several RadioButtons that are outside of a GroupBox, the window acts as a GroupBox. Only one RadioButton can be active at a time *unless* the check box for the Automatic property on the RadioButton's General property page is cleared. |

When the Automatic property is not set, you must use scripts to control when a button is selected. Multiple RadioButtons can be selected outside of a group.

The Automatic property does not change how RadioButtons are processed inside a GroupBox.

StaticText

You use a StaticText control to display text to the user or to describe a control that does not have text associated with it, such as a list box or edit control.

The user cannot change the text, but you can change the text for a StaticText control in a script by assigning a string to the control's Text property.

StaticText controls have events associated with them, but you will probably seldom write scripts for them because users do not expect to interact with static text.

Indicating accelerator keys

One use of a StaticText control is to label a list box or edit control. If you assign an accelerator key to a list box or edit control, you need to indicate the accelerator key in the text that labels the control. Otherwise, the user would have no way of knowing that an accelerator key is defined for the control. This technique is described in "Defining accelerator keys" on page 264.

Indicating a border style

You can select a border style using the BorderStyle property on the General property page.

Selecting the Border property
The BorderStyle property will affect only the StaticText control if the Border property check box is selected.

StaticHyperLink

A StaticHyperLink is display text that provides a hot link to a specified Web page. When a user clicks the StaticHyperLink in a window, the user's Web browser opens to display the page.

The StaticHyperLink control has a URL property that specifies the target of the link. You specify the text and URL on the StaticHyperLink control's General tab page in the Properties view.

If you know that your users have browsers that support URL completion, you can enter a partial address—for example, `sybase.com` instead of the complete address, `http://www.sybase.com`.

When the StaticHyperLink control is in an MDI Frame window with MicroHelp, the URL you specify displays in the status bar when the user's pointer is over the control.

A hand is the default pointer and blue underlined text is the default font. To change the pointer, use the Other property page. To change the font, use the Font property page.

Picture

Pictures are PowerBuilder-specific controls that display a bitmap (BMP) file, a run-length encoded (.RLE) file, an Aldus-style Windows metafile (WMF), a GIF file, an animated GIF file, or a JPEG file.

❖ **To display a picture:**

1   Place a picture control in the window.

2   In the General tab page in the Properties view, enter in the PictureName text box the name of the file you want to display, or browse to select a file.

   The picture displays.

You can choose to resize or invert the image.

Be careful about how you use picture controls. They can serve almost any purpose. They have events, so users can click on them, but you can also use them simply to display images. Be consistent in their use so users know what they can do with them.

PictureHyperLink

A PictureHyperLink is a picture that provides a hot link to a specified Web page. When a user clicks the PictureHyperLink in a window, the user's Web browser opens to display the page.

The PictureHyperLink control has a URL property that specifies the target of the link. You specify the picture and URL in the PictureHyperLink control's Properties view in the General tab page. If you know that your users have browsers that support URL completion, you can enter a partial address—for example, `sybase.com`—instead of the complete address, `http://www.sybase.com`.

When the PictureHyperLink control is in an MDI Frame window with MicroHelp, the URL you specify appears in the status bar when the user's pointer is over the control.

A hand is the default pointer. To change the pointer, use the Other property page.

The PictureHyperLink control is a descendant of the Picture control. Like a Picture control, a PictureHyperLink control can display a bitmap (BMP) file, a run-length encoded (RLE) file, an Aldus-style Windows metafile (WMF), a GIF file, an animated GIF file, or a JPEG file.

You display a picture in a PictureHyperLink control in the same way you display a picture in a picture control. For more information, see "Picture" on page 276.

GroupBox

You use a GroupBox to group a set of related controls. When a user tabs from another control to a GroupBox, or selects a GroupBox, the first control in the GroupBox gets focus. To tab between controls in a GroupBox, set the tab value of the GroupBox to 0 and assign a tab value to each control within it.

All RadioButtons in a GroupBox are considered to be in a group. For more information about using RadioButtons in GroupBoxes, see "RadioButton" on page 274.

Drawing controls

PowerBuilder provides the following drawing controls: Line, Oval, Rectangle, and RoundRectangle. Drawing controls are usually used only to enhance the appearance of a window or to group controls. However, constructor and destructor events are available, and you can define your own unmapped events for a drawing control. A drawing control does not receive Windows messages, so a mapped event would not be useful.

You can use the following functions to manipulate drawing controls at runtime:

Hide
Move
Resize
Show

In addition, each drawing control has a set of properties that define its appearance. You can assign values to the properties in a script to change the appearance of a drawing control.

Never in front
You cannot place a drawing control on top of another control that is not a drawing control, such as a GroupBox. Drawing controls *always* appear behind other controls whether or not the Bring to Front or Send to Back items on the pop-up menu are set. However, drawing controls can be on top of or behind other drawing controls.

SingleLineEdit and MultiLineEdit

A SingleLineEdit is a box in which users can enter a single line of text. A MultiLineEdit is a box in which users can enter more than one line of text.

SingleLineEdits and MultiLineEdits are typically used for input and output of data.

For these controls, you can specify many properties, including:

- Whether the box has a border (the Border property)

- Whether the box automatically scrolls as needed (AutoHScroll and, for MultiLineEdits, AutoVScroll)

- For SingleLineEdits, whether the box is a Password box so asterisks are displayed instead of the actual entry (Password)

- The case in which to accept and display the entry (TextCase)

- Whether the selection displays when the control does not have focus (Hide Selection)

For more information about properties of these controls, right-click in any tab page in the Properties view and select Help from the pop-up menu.

EditMask

Sometimes users need to enter data that has a fixed format. For example, U.S. and Canadian phone numbers have a three-digit area code, followed by three digits, followed by four digits. You can use an EditMask control that specifies that format to make it easier for users to enter values. Think of an EditMask control as a smart SingleLineEdit: it knows the format of the data that can be entered.

An edit mask consists of special characters that determine what can be entered in the box. An edit mask can also contain punctuation characters to aid the user.

For example, to make it easier for users to enter phone numbers in the proper format, you can specify the following mask, where # indicates a number:

```
(###) ###-####
```

At runtime, the punctuation characters (the parentheses and dash) display in the box and the cursor jumps over them as the user types.

Masks in EditMask controls in windows work in a similar way to masks in display formats and in the EditMask edit style in DataWindow objects. For more information about specifying masks, see the discussion of display formats in Chapter 22, "Displaying and Validating Data."

Edit mask character for Arabic and Hebrew
The b mask character allows the entry of Arabic characters when you run PowerBuilder on an Arabic-enabled version of Windows and Hebrew characters when running on a Hebrew-enabled version. It has no effect on other operating systems.

❖ **To use an EditMask control:**

1   Select the EditMask to display its properties in the Properties view.

2   Name the control on the General property page.

3   Select the Mask tab.

4   In the MaskDataType drop-down list, specify the type of data that users will enter into the control.

5   In the Mask edit box, type the mask.

You can click the button on the right and select masks. The masks have the special characters used for the specified data type.

6   Specify other properties for the EditMask control.

For information on the other properties, right-click in any tab page in the Properties view and select Help from the pop-up menu.

Control size and text entry
The size of the EditMask control affects its behavior. If the control is too small for the specified font size, users might not be able to enter text.

To correct this, either specify a smaller font size or resize the EditMask control.

Validation for EditMask controls

The EditMask control checks the validity of a date when you enter it, but if you change a date so that it is no longer valid, its validity is not checked when you tab away from the control. For example, if you enter the date 12/31/2005 in an EditMask control with the mask mm/dd/yyyy, you can delete the 1 in 12, so that the date becomes 02/31/2005. To catch problems like this, add validation code to the LoseFocus event for the control.

Keyboard behavior

Some keystrokes have special behavior in EditMask controls. For more information, see "The EditMask edit style" on page 616.

Using a drop-down calendar

You can use a drop-down calendar that is similar to the DatePicker control in EditMask controls that have a Date or DateTime edit mask. The user can choose to edit the date in the control or to select a date from a drop-down calendar.

To specify that an EditMask control uses a drop-down calendar to display and set dates, select the Drop-down Calendar check box on the Mask page in the Properties view. You can set display properties for the calendar on the Calendar page. Users navigate and select dates within the calendar as they do in the calendar in a DatePicker control.

Using spin controls
You can define an EditMask as a spin control, which is an edit control that contains up and down arrows that users can click to cycle through fixed values. For example, assume you want to allow your users to select how many copies of a report to print. You could define an EditMask as a spin control that allows users to select from a range of values.

❖ **To define an EditMask as a spin control:**

1 Name the EditMask and provide the data type and mask, as described above.

2 Select the Spin check box on the Mask property page.

3 Specify the needed information.

For example, to allow users to select a number from 1 to 20 in increments of 1, specify a spin range of 1 to 20 and a spin increment of 1.

For more information on the options for spin controls, right-click in any tab page in the Properties view and select Help from the pop-up menu.

HScrollBar and VScrollBar

You can place freestanding scroll bar controls within a window. Typically, you use these controls to do one of the following:

• Act as a slider control with which users can specify a continuous value

• Graphically display information to the user

You can set the position of the scroll box by specifying the value for the control's Position property. When the user drags the scroll box, the value of Position is automatically updated.

HTrackBar and VTrackBar

HTrackBars and VTrackBars are bars with sliders that move in discrete increments. Like a scroll bar, you typically use a track bar as a slider control that allows users to specify a value or see a value you have displayed graphically, but on a discrete scale rather than a continuous scale. Clicking on the slider moves it in discrete increments instead of continuously.

Typically a horizontal trackbar has a series of tick marks along the bottom of the channel and a vertical trackbar has tick marks on the right.
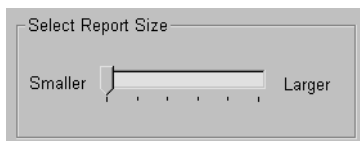
Use a trackbar when you want the user to select a discrete value. For example, you might use a trackbar to enable a user to select a timer interval or the size of a window.

You can set properties such as minimum and maximum values, the frequency of tick marks, and the location where tick marks display.

You can highlight a range of values in the trackbar with the SelectionRange function. The range you select is indicated by a black fill in the channel and an arrow at each end of the range. This is useful if you want to indicate a range of preferred values. In a scheduling application, the selection range could indicate a block of time that is unavailable. Setting a selection range does not prevent the user from selecting a value either inside or outside the range.
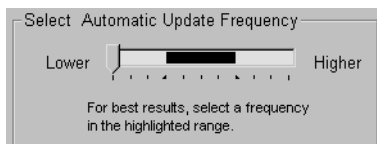
You can see an example of a window with a trackbar in the PowerBuilder Code Examples sample application in the Examples subdirectory in your PowerBuilder directory. See the w_trackbars window in *PBEXAMW3.PBL*.

HProgressBar and VProgressBar

HProgressBars and VProgressBars are rectangles that indicate the progress of a lengthy operation, such as an installation program that copies a large number of files. The progress bar gradually fills with the system highlight color as the operation progresses.

You can set the range and current position of the progress bar in the Properties view using the MinPosition, MaxPosition, and Position properties. To specify the size of each increment, set the SetStep property.

You can see an example of a window with a progress bar in the PowerBuilder Code Examples sample application in the Examples subdirectory in your PowerBuilder directory. See the w_progressbars window in *PBEXAMW3.PBL*.

DropDownListBox

DropDownListBoxes combine the features of a SingleLineEdit and a ListBox.
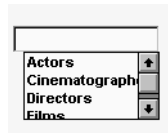
There are two types of DropDownListBoxes:

• Noneditable

• Editable

Noneditable boxes

If you want your user to choose only from a fixed set of choices, make the DropDownListBox noneditable.

In these boxes, the only valid values are those in the list.

There are several ways for users to pick an item from a noneditable DropDownListBox:

• Use the arrow keys to scroll through the list.

• Type a character. The ListBox scrolls to the first entry in the list that begins with the typed character. Typing the character again scrolls to the next entry that begins with the character, unless the character can be combined with the first to match an entry.

• Click the down arrow to the right of the edit control to display the list, then select the one you want.

Editable boxes	If you want to give users the option of specifying a value that is not in the list, make the DropDownListBox editable by selecting the AllowEdit check box on the General tab page.
	With editable DropDownListBoxes, you can choose to have the list always display or not. For the latter type, the user can display the list by clicking the down arrow.
Populating the list	You specify the list in a DropDownListBox the same way as for a ListBox. For information, see "ListBox" on page 285.
Specifying the size of the drop-down box	To indicate the size of the box that drops down, size the control in the Window painter using the mouse. When the control is selected in the painter, the full size—including the drop-down box—is shown.
Other properties	As with ListBoxes, you can specify whether the list is sorted and whether the edit control is scrollable.
	For more information, right-click in any tab page in the Properties view and select Help from the pop-up menu.

DropDownPictureListBox

DropDownPictureListBoxes are similar to DropDownListBoxes in the way they present information. They differ in that DropDownListBoxes use only text to present information, whereas DropDownPictureListBoxes add images to the information.

Everything that you can do with DropDownListBoxes you can do with DropDownPictureListBoxes. For more information, see "DropDownListBox" on page 283.

| Adding images to a DropDownPictureList Box | You can choose from a group of stock images provided by PowerBuilder, or use any bitmap (BMP) file, icon (ICO) file, GIF file, or JPEG file when you add images to a DropDownPictureListBox. You use the same technique that you use to add pictures to a PictureListBox. For more information, see "Adding images to a PictureListBox" on page 286. |

ListBox

A ListBox displays available choices. You can specify that ListBoxes have scroll bars if more choices exist than can be displayed in the ListBox at one time.

ListBoxes are an exception to the rule that a control should either invoke an action or be used for viewing and entering data. ListBoxes can do both. ListBoxes display data, but can also invoke actions. Typically in Windows applications, clicking an item in the ListBox selects the item. Double-clicking an item acts upon the item.

For example, in the PowerBuilder Open dialog box, clicking an object name in a ListBox selects the object. Double-clicking a name opens the object's painter.

PowerBuilder automatically selects (highlights) an item when a user selects it at runtime. If you want something to happen when users double-click an item, you must code a script for the control's DoubleClicked event. The Clicked event is always triggered before the DoubleClicked event.

Populating the list

To add items to a ListBox, select the ListBox to display its properties in the Properties view, select the Items tab, and enter the values for the list. Press tab to go to the next line.

In the Items tab page, you can work with rows in this way:

To do this	Do this
Select a row	Click the row button on the left or with the cursor in the edit box, press Shift+Space
Delete a row	Select the row and press Delete
Move a row	Click the row button and drag the row where you want it or press Shift+Space to select the row and then press Ctrl+Up Arrow or Ctrl+Down Arrow to move the row
Delete text	Click the text and select Delete from the pop-up menu

Changing the list at runtime
To change the items in the list at runtime, use the functions AddItem, DeleteItem, and InsertItem.

Setting tab stops

You can set tab stops for text in ListBoxes (and in MultiLineEdits) by setting the TabStop property on the General property page. You can define up to 16 tab stops. The default is a tab stop every eight characters.

You can also define tab stops in a script. Here is an example that defines two tab stops and populates a ListBox:

```
// lb_1 is the name of the ListBox.
string f1, f2, f3
f1 = "1"
f2 = "Emily"
f3 = "Foulkes"
// Define 1st tab stop at character 5.
lb_1.tabstop[1] = 5
// Define 2nd tab stop 10 characters after the 1st.
lb_1.tabstop[2] = 10
// Add an item, separated by tabs.
// Note that the ~t must have a space on either side
// and must be lowercase.
lb_1.AddItem(f1 + " ~t " + f2 + " ~t " + f3)
```

Note that this script will not work if it is in the window's Open event, because the controls have not yet been created. The best way to specify this is in a user event that is posted in the window's Open event using the PostEvent function.

Other properties

For ListBoxes, you can specify whether:

• Items in the ListBox are displayed in sorted order

• The ListBox allows the user to select multiple items

• The ListBox displays scroll bars if needed

For more information, right-click in any tab page in the Properties view for a ListBox and select Help from the pop-up menu.

PictureListBox

A PictureListBox, like a ListBox, displays available choices in both text and images. You can specify that PictureListBoxes have scroll bars if more choices exist than can be displayed in the PictureListBox at one time.

Adding images to a PictureListBox

You can choose from a group of stock images provided by PowerBuilder, or use any bitmap (BMP) file, icon (ICO) file, GIF file, or JPEG file when you add images to a PictureListBox.

Keep in mind, however, that the images should add meaning to the list of choices. If you use a large number of images in a list, they become meaningless.

You could, for example, use images in a long list of employees to show the department to which each employee belongs, so you might have a list with 20 or 30 employees, each associated with one of five images.

❖ **To add an image to a PictureListBox:**

1 Select the PictureListBox control to display its properties in the Properties view, and then select the Pictures tab.

The Pictures property page displays.

2 Use the PictureName drop-down ListBox to select stock pictures to add to the PictureListBox
or
Use the Browse button to select a bitmap (BMP) file, icon (ICO) file, GIF file, or JPEG file to include in the PictureListBox.

About cursor files
To use a cursor file, you must type the file name. You cannot select it.

---

3 Specify a picture mask color (the color that will be transparent for the picture).

4 Specify the height and width for the image in pixels or accept the defaults.
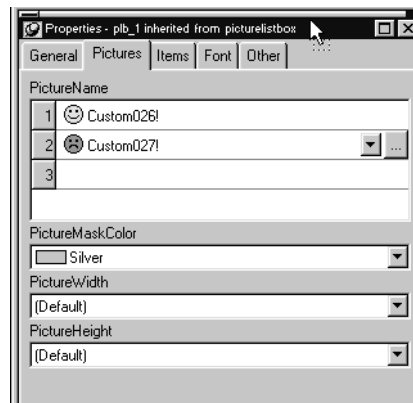
5 Repeat the procedure for the number of images you plan to use in your PictureListBox.

6   Select the Items tab and change the Picture Index for each item to the
     appropriate number.
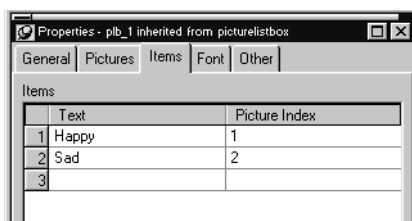
7   Click OK.

On the Items tab page, you can work with rows in this way:

To	Do this
Select a row	Click the row button on the left, or with the cursor in the edit box, press Shift+Space
Delete a row	Select the row and press Delete
Move a row	Click the row button and drag the row where you want it or press Shift+Space to select the row and then press Ctrl+Up Arrow or Ctrl+Down Arrow to move the row
Delete text	Click the text and select Delete from the pop-up menu

On the Pictures tab page, you can work with rows in the same way, and also:

To	Do this
Browse for a picture	Select the row and click the Browse button or press F2

For information about other properties, right-click in any tab page in the
Properties view and select Help from the pop-up menu.

ListView

A ListView control lets you display items and icons in a variety of
arrangements. You can display large or small icons in free-form lists. You can
add columns, pictures, and items to the ListView, and modify column
properties, using PowerScript functions such as AddColumn, AddLargePicture,
SetItem, SetColumn, and so on. For information about ListView functions, see
the online Help.

The following illustration from the Code Examples application shows a ListView control used in a sales order application.
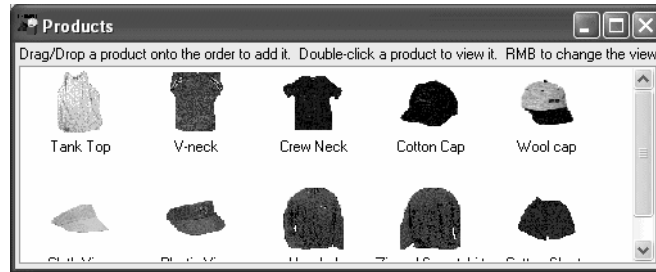
Adding ListView items and pictures

Adding images to a ListView control is the same as adding images to a PictureListBox. The ListView control's Properties view has two tab pages for adding pictures: Large Picture (default size 32 by 32 pixels) and Small Picture (16 by 16 pixels).

For more information, see "Adding images to a PictureListBox" on page 286.

❖ **To add ListView items:**

1   Select the ListView control to display its properties in the Properties view and then select the Items tab.

2   Enter the name of the ListView item and the picture index you want to associate with it. This picture index corresponds to the images you select on the Large Picture, Small Picture, and State property pages.

On the Items tab page, you can work with rows in this way:

To	Do this
Select a row	Click the row button on the left, or with the cursor in the edit box, press Shift+Space
Delete a row	Select the row and press Delete
Move a row	Click the row button and drag the row where you want it, or press Shift+Space to select the row and then press Ctrl+Up Arrow or Ctrl+Down Arrow to move the row
Delete text	Click the text and select Delete from the pop-up menu

Note Setting the picture index for the first item to zero clears all the settings on the tab page.

3   Set properties for the item on the Large Picture, Small Picture, and/or State tab pages as you did on the Items tab page.

On these pages, you can also browse for a picture. To do so, click the browse button or press F2.

4   Repeat until all the items are added to the ListView.

Choosing a ListView style

You can display a ListView in four styles:

- Large icon

- Small icon

- List

- Report

❖ **To select a ListView style:**

1   Select the ListView control to display its properties in the Properties view and then select the General tab.

2   Select the type of view you want from the View drop-down list.

For more information about other properties, right-click in any tab page in the Properties view and select Help from the pop-up menu.

Setting other properties

You can set other ListView properties.

❖ **To specify other ListView properties:**

1   Select the ListView control to display its properties in the Properties view.

2   Choose the tab appropriate to the property you want to specify:

To specify	Choose this tab
The border style	General
Whether the user can delete items	General
The images for ListView items in Large Icon view	Large Picture
The images for ListView items in Small Icon, list, and report views	Small Picture
The state images for ListView items	State
The names and associated picture index for ListView items	Items
The font size, family, and color for ListView items	Font
The size and position of the ListView	Other

To specify	Choose this tab
The icon for the mouse pointer in the ListView	Other
The icon for a drag item, and whether the drag-and-drop must be performed programmatically	Other

For more information on the ListView control, see *Application Techniques*. For information about its properties, see *Objects and Controls*.

TreeView

You can use TreeView controls in your application to represent relationships among hierarchical data. An example of a TreeView implementation is PowerBuilder's Browser. The tab pages in the Browser contain TreeView controls.
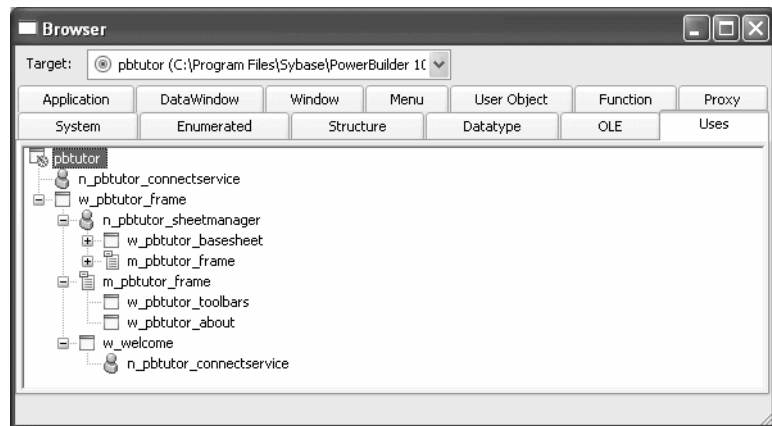
Adding TreeView items and pictures

A TreeView consists of TreeView items that are associated with one or more pictures. You add images to a TreeView in the same way that you add images to a PictureListBox.

For more information, see "Adding images to a PictureListBox" on page 286.

Dynamically changing image size
The image size can be changed at runtime by setting the PictureHeight and PictureWidth properties when you create a TreeView.

For more information about PictureHeight and PictureWidth, see the *PowerScript Reference*.

❖ **To add items to a TreeView:**

• Write a script in the TreeView constructor event to create TreeView items.

For more information about populating a TreeView, see *Application Techniques* and *Objects and Controls.*

Adding state pictures to TreeView items

A *state* picture is an image that appears to the left of the TreeView item indicating that the item is not in its normal mode. A state picture could indicate that a TreeView item is being changed, or that it is performing a process and is unavailable for action.
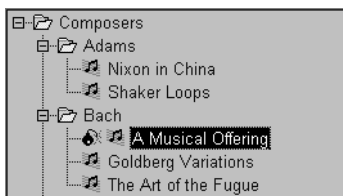
❖ **To specify a state picture for a TreeView item:**

1 Select the TreeView control to display its properties in the Properties view and then select the State tab.

2 Do one of the following:

• Use the StatePictureName drop-down list to select stock pictures to add to the TreeView.

• Use the Browse button to select any bitmap (BMP) file, icon (ICO) file, GIF file, or JPEG file.

To specify a cursor file
To use a cursor file, you must type the file name. You cannot select it.

Working in the Properties view with the rows in the State or Pictures tab page is the same as working with them in a ListView control. For information, see "ListView" on page 288.

❖ **To activate a state picture for a TreeView item:**

• Write a script that changes the image when appropriate.

For example, the following script gets the current TreeView item and displays the state picture for it.

```
long          ll_tvi
treeviewitem  tvi
```

```
ll_tvi = tv_foo.finditem(currenttreeitem! , 0)
tv_foo.getitem(ll_tvi , tvi)
tvi.statepictureindex = 1
tv_foo.setitem(ll_tvi, tvi)
```

For more information on the TreeView control, see *Application Techniques*.

Setting other properties

❖   **To specify other TreeView properties:**

1   Select the TreeView control to display its properties in the Properties view and then select the General tab.

2   Enter a name for the TreeView in the Name text box and specify other properties as appropriate. Among the properties you can specify on the General property page are:

   •   The border style

   •   Whether the TreeView has lines showing the item hierarchy

   •   Whether the TreeView includes collapse and expand buttons

   •   Whether the user can delete items

   •   Whether the user can drag and drop items into the TreeView

   For more information, right-click in any tab page in the Properties view and select Help from the pop-up menu.

3   For other options, choose the tab appropriate to the property you want to specify:

To specify	Choose this tab
The images used to represent TreeView items	Pictures
The state images for the TreeView items	State
The font size, family, and color for TreeView items	Font
The size and position of the TreeView	Other
The icon for the mouse pointer in the TreeView	Other
The icon for a drag item, and whether the drag-and-drop must be performed programmatically	Other

For more information on the TreeView control, see *Application Techniques*. For information about its properties, see *Objects and Controls*.

Tab

A Tab control is a container for tab pages that display other controls. You can add a Tab control to a window in your application to present information that can logically be grouped together but may also be divided into distinct categories. An example is the use of tab pages in the Properties view for objects in PowerBuilder. Each tab page has a tab that displays the label for the tab page and is always visible, whichever tab page is selected.

When you add a Tab control to a window, PowerBuilder creates a Tab control with one tab page labeled "none". The control is rectangular.

Selecting Tab controls and tab pages

You may find that you select the control when you want to select the page and vice versa. This Tab control has three tab pages. The TabPosition setting is tabsontopandbottom!, so that the tab for the selected tab page and pages that precede it in the tab order display at the top of the Tab control.
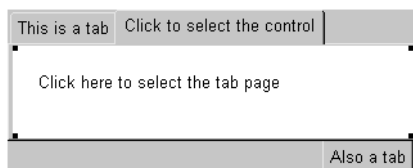
To select the Tab control, click any of the tabs where the label displays, or in the area adjacent to the tabs, shown in gray here.

To select a tab page, click its tab and then click anywhere on the tab page except the tab itself. The handles at the corners of the white area indicate that the tab page is selected, not the Tab control.

Adding tab pages to a Tab control

To add a new Tab control to a window, select Insert>Control>Tab and click in the window. The control has one tab page when it is created. Use the following procedure to add additional tab pages to the tab control.

❖ **To create a new tab page within a Tab control:**

1 Select the Tab control by clicking on the tab of the tab page or in the area to its right.

The handles that indicate that the Tab control is selected display at the corners of the Tab control. If you selected the tab page, the handles display at the corners of the area under the tab.

2   Choose Insert TabPage from the pop-up menu.
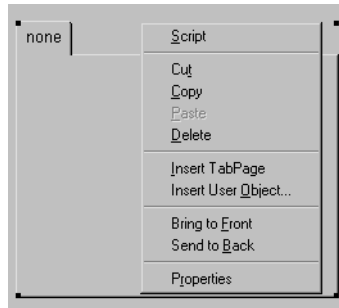
3   Add controls to the new tab page.

Creating a reusable
tab page

You can create reusable tab pages in the User Object painter by defining a tab page with controls on it that is independent of a Tab control. Then you can add that tab page to one or more Tab controls.

❖   **To define a tab page that is independent of a Tab control:**

1   Click the New button on the PowerBar and use the Custom Visual icon on the Object tab page to create a custom visual user object.

2   Size the user object to match the size of the Tab controls in which you will use it.

3   Add the controls that you want to have appear on the tab page to the user object.

4   Select the user object (not one of the controls you added) and specify the information to be used by the tab page on the TabPage page in the Properties view:

•   Text—the text to be displayed on the tab

•   PictureName—a picture to appear on the tab with or instead of the text

•   PowerTipText—text for a pop-up message that displays when the user moves the cursor to the tab

•   Colors for the tab and the text on the tab

5   Save and close the user object.

Adding a reusable tab page to a Tab control

Once you have created a user object that can be used as a tab page, you can add it to a Tab control. You cannot add the user object to a Tab control if the user object is open, and, after you have added the user object to the control, you cannot open the user object and the window that contains the Tab control at the same time.

❖ **To add a tab page that exists as an independent user object to a Tab control:**

1   In the Window painter, right-click the Tab control.

2   Choose Insert User Object from the pop-up menu.

3   Select a user object that you have set up as a tab page and click OK.

A tab page, inherited from the user object you selected, is inserted. You can select the tab page, set its tab page properties, and write scripts for the inherited user object just as you do for tab pages defined within the Tab control, but you cannot edit the content of the user object within the Tab control. If you want to edit the controls, close the Window painter and go back to the User Object painter to make changes.

Manipulating the Tab control

❖ **To change the name and properties of the Tab control:**

1   Click any of the tabs in the Tab control to display the Tab control properties in the Properties view.

2   Edit the properties.

For more information, right-click in the Properties view and select Help from the pop-up menu.

❖ **To change the scripts of the Tab control:**

1   With the mouse pointer on one of the tabs, double-click the Tab control, or display the pop-up menu and select Script.

2   Select a script and edit it.

❖ **To resize a Tab control:**

•   Grab a border of the control and drag it to the new size.

The Tab control and all tab pages are sized as a group.

❖ **To move a Tab control:**

- With the mouse pointer on one of the tabs, hold down the left mouse button and drag to move the control to the new position.

  The Tab control and all tab pages are moved as a group.

❖ **To delete a Tab control:**

- With the mouse pointer on one of the tabs, select Cut or Delete from the pop-up menu.

Manipulating the tab pages

❖ **To view a different tab page:**

- Click on the page's tab.

  The selected tab page is brought to the front. The tabs are rearranged according to the TabPosition setting you have chosen.

❖ **To change the name and properties of a tab page:**

1   Select the tab.

    It might move to the position for a selected tab based on the TabPosition setting. For example, if TabPosition is set to tabsonbottomandtop! and a tab displays at the top, it moves to the bottom when you select it.

2   Click anywhere on the tab page except the tab.

3   Edit the properties.

❖ **To change the scripts of the tab page:**

1   Select the tab.

    It may move to the position for a selected tab based on the Tab Position setting.

2   Click anywhere on the tab page except the tab.

3   Select Script from the tab page's pop-up menu.

4   Select a script and edit it.

❖ **To delete a tab page from a Tab control:**

- With the mouse pointer anywhere on the tab page except the tab, select Cut or Delete from the pop-up menu.

Managing controls on
tab pages

❖ **To add a control to a tab page:**

• Choose a control from the toolbar or the Control menu and click on the tab page, just as you would add a control to a window.

You can add controls only to a tab page created within the Tab control. To add controls to an independent tab page, open it in the User Object painter.

❖ **To move a control from one tab page to another:**

• Cut or copy the control and paste it on the destination tab page.

The source and destination tab pages must be embedded tab pages, not independent ones created in the User Object painter.

❖ **To move a control between a tab page and the window containing the Tab control:**

• Cut or copy the control and paste it on the destination window or tab page.

Moving the control between a tab page and the window changes the control's parent, which affects scripts that refer to the control.

For more information on the Tab control, see the chapter on using tabs in a window in *Application Techniques*.

MonthCalendar

A MonthCalendar control lets you display a calendar to your users to make it easy for them to view and set date information. You can size the calendar to show from one to twelve months. The following illustration shows a calendar with three months. Today's date is August 25, 2004, and the date October 8 has been selected.

If a user selects a date or a range of dates in the calendar, you can use the GetSelectedDate or GetSelectedRange functions to obtain them. You use the SetSelectedDate and SetSelectedRange functions to select dates programmatically.

You can also:

• Set and get minimum and maximum dates that can be displayed in the calendar

• Display dates in bold

• Get the number of months currently displayed with the start and end dates

• Set and get the date the calendar uses as the current date

• Use properties to customize the appearance of the calendar

Users can navigate through the calendar using the arrow keys in the top corners. You can specify how many months should scroll for each click using the ScrollRate property. If users click on the name of the month in the title bar, a drop-down list displays, allowing them to navigate to another month in the same year. Clicking on the year in the title bar displays a spin control that lets users navigate quickly to a different year.

DatePicker

The DatePicker control provides an easy way for a user to select a single date. The user can choose to edit the date in the control or to select a date from a drop-down calendar. The calendar is similar to the MonthCalendar control, which can be used to select a range of dates. As an alternative to the drop-down calendar, you can set the ShowUpDown property to display up and down arrows that allow users to specify one element of the date at a time.

The drop-down calendar can only be used to select a date. The up and down arrows lets users specify a time as well as a date. The following illustration shows three DatePicker controls. The controls on the left have the ShowUpDown property set. One uses the standard date format, and the other uses a custom format that displays the date and time. The control on the right uses the default drop-down calendar option and the standard long date format.
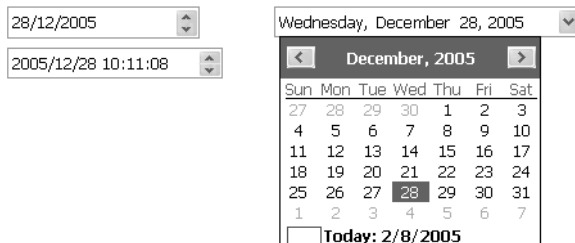
You can set initial properties for the appearance and behavior of the control in the Properties view. Properties that apply to the drop-down calendar portion of the control are similar to the properties that apply to the MonthCalendar control and display on the Calendar page in the Properties view. For example, you can choose which day of the week displays as the first day in the week, whether the current date is circled, and whether a "Today Section" showing the current date displays at the bottom of the calendar.

Specifying a format

You can choose to display the date in the DatePicker control as a long date, a short date, a time, or with a custom format. To set a custom format in the painter, select dtfCustom! from the Format list and type the format in the Custom Format field. For example, the second control on the left in the previous illustration uses the custom format yyyy/MM/dd HH:mm:ss. The uppercase H for the hour format specifies a 24-hour clock. The following statements set the Format property to use a custom format and set the CustomFormat property to show the full name of the month, the day of the month followed by a comma, and the four-digit year:

```
dp_1.Format = dtfCustom!
dp_1.CustomFormat = "MMMM dd, yyyy"
```

For a complete list of formats you can use, see the description of the CustomFormat property in the online Help.

Specifying maximum and minimum dates

The MaxDate and MinDate properties determine the range of dates that a user can enter or pick in the control. If you set a restricted range, the calendar displays only the months in the range you set, and if users type a date outside the range into the control, the date reverts to its previous value.

Editing modes

When a user tabs into a DatePicker control, the control is in normal editing mode and behaves in much the same way as an EditMask control with a Date or DateTime mask. Users can edit the part of the date (year, month, day, hour, minutes, or seconds) that has focus using the up/down arrow keys on the keyboard or, for numeric fields, the number keys. Use the left/right arrow keys to move between parts of the date.

If the control has a drop-down calendar, users can navigate from one month or year to another using the controls in the calendar and click to select a date. If the ShowUpDown option is set, users can change the selected part of the date or time with the up and down keys in the control. To navigate in the drop-down calendar, a user can:

- Click the arrows in the top corners to move from month to month

- Click the month to display a list of months, then click a month to select it

- Click the year to display a spin control, then use the spin control's arrows to select a year

- Click a date to select the date and close the calendar

- Press the Esc key to close the calendar without changing the selection

Allowing users to edit the date directly

You can give users a third way to change the date by setting the AllowEdit property to true. The user can then press F2 or click in the control to select all the text in the control for editing. When the control loses focus, the control returns to normal editing mode and the UserString event is fired. The UserString event lets you test whether the text the user entered in the control is a valid date and set the value to the new date if it is valid. If it is valid, you can use the event's *dtm* by reference argument to set the value to the new date. This code in the UserString event tests whether the date is valid and within range:

```
Date d
DateTime dt

IF IsDate(userstr) THEN
   d = Date(usrstr)
   IF (this.maxdate >= d and this.mindate <= d) THEN
     dtm = DateTime(dt)
   ELSE
     MessageBox("Date is out of range", userstr)
   END IF
ELSE
   MessageBox("Date is invalid", userstr)
END IF
```

The Value property

The Value property contains the date and time to which the control is set. If you do not specify a date, the Value property defaults to the current date and time. You can set the property in the painter or in a script. If you change the value at runtime, the display is updated automatically. The Value property has two parts that can be obtained from the DateValue and TimeValue properties. These properties should be used only to obtain the date and time parts of the Value property; they cannot be used to set a date or time. The Text property and the GetText function return the Value property as a string formatted according to your format property settings.

You can use the SetValue function to set the Value property in a control using separate date and time values or a single DateTime value. This example sets the property control using separate date and time values:

```
date d
time t

d=date("2007/12/27")
t=time("12:00:00")

dp_1.SetValue(d, t)
```

This example sets the Value property using a DateTime value:

```
date d
time t
datetime dt
dt = DateTime(d, t)

dp_1.SetValue(dt)
```

Localizing the DatePicker control

The DatePicker control is designed to work in different locales. The string values in the DatePicker control support non-English charactersand the names of months and days of the week in the calendar display in the local language. You can set the FirstDayOfWeek property on the Calendar page in the Properties view to have the drop-down calendar use Monday or any other day of the week as the first day.

The MaxDate and MinDate properties and the date part of the Value property use the Short Date format specified in the regional settings in the Windows control panel of the local computer, and the time part uses the local computer's time format. The three predefined display formats—long date, short date, and time—also depend on the local computer's regional settings.

Animation

Animation controls can display Audio-Video Interleaved (AVI) clips. An AVI clip is a series of bitmap frames that can be played like a movie. The clip can come from an uncompressed AVI file or from an AVI file compressed using run-length encoding (BI_RLE8). If you use an AVI file that has a sound channel, the sound is not played.

You might display an AVI clip to show the user that some activity is occurring while a lengthy operation such as a search or full build is completing. To specify which AVI clip to use, specify the AVI file name in the control's AnimationName property. If you want the control to display only when an event in your application starts to play the application, set its Border and Visible properties to false and its Transparent property to true.

InkEdit and InkPicture

InkEdit and InkPicture provide the ability to capture ink input from users of Tablet PCs.

The InkEdit control captures and recognizes handwriting and optionally converts it to text. The InkPicture control captures signatures, drawings, and other annotations that do not need to be recognized as text. You can place a background image in an InkPicture control, and capture and save a user's annotations to the picture.

The ink controls are fully functional on Tablet PCs. On other computers, the InkEdit control behaves like a multiline edit control. If the Microsoft Tablet PC Software Development Kit (SDK) 1.7 is installed on the computer, InkPicture controls can accept ink input from the mouse.

InkEdit control

The InkEdit control on a Tablet PC is like a MultiLineEdit control that has the added ability to accept ink input. On other PCs, the InkEdit control behaves as a normal MultiLineEdit control and cannot collect ink.

On a Tablet PC, the InkEdit control collects ink from a user in the form of handwriting and can handle single or multiple lines of text. It also recognizes gestures, which are specific pen strokes that represent a keyboard action such as backspace, space, or tab.The InkEdit control can convert ink to text, or leave it as handwriting.

InkPicture control

The InkPicture control behaves like a Picture control that accepts annotation. The InkPicture control does not convert ink to text. You can associate a picture with the control so that the user can draw annotations on the picture, then save the ink, the picture, or both. If you want to use the control to capture and save signatures, you usually do not associate a picture with it.

You might use an InkPicture control to display an image of a process flow chart or a floor plan of a building, and capture suggested changes that users enter in the form of ink. Using an image of a garden, for example, a user could mark trees and shrubs to be removed and indicate where new plants should be added.

You can save the background image, the ink annotations, or both, to a file or to a blob.

C H A P T E R   1 3 **Understanding Inheritance**

About this chapter   This chapter describes how to use inheritance to build PowerBuilder objects.

Contents

Topic	Page
About inheritance	305
Creating new objects using inheritance	306
The inheritance hierarchy	307
Browsing the class hierarchy	308
Working with inherited objects	310
Using inherited scripts	311

About inheritance

One of the most powerful features of PowerBuilder is inheritance. It enables you to build windows, user objects, and menus that derive from existing objects.

Using inheritance has a number of advantages:

- When you change an ancestor object, the changes are reflected in all the descendants. You do not have to make manual changes in the descendants, as you would in a copy. This saves you coding time and makes the application easier to maintain.

- The descendant inherits the ancestor's scripts, so you do not have to re-enter the code to add to the script.

- You gain consistency in the code and objects in your applications.

This chapter describes how inheritance works in PowerBuilder and how to use it to maximize your productivity.

Opening ancestors and descendants
To enforce consistency, PowerBuilder does not let you open an ancestor object until you have closed any descendants that are open, or open a descendent object when its ancestor is open.

Creating new objects using inheritance

You use the Inherit From Object dialog box to create a new window, user object, or menu using inheritance.

❖ **To create a new object using inheritance:**

1 Click the Inherit button in the PowerBar, or select File>Inherit from the menu.

2 In the Inherit From Object dialog box, select the object type (menu, user object, or window) from the Objects of Type drop-down list, and then select the target as well as the library or libraries you want to look in. Finally, select the object from which you want to inherit the new object.

Displaying objects from many libraries
To find an object more easily, you can select more than one library in the Libraries list. Use Ctrl+Click to toggle selected libraries and Shift+Click to select a range.

---

3 Click OK.

The new object, which is a descendant of the object you chose to inherit from, opens in the appropriate painter.

The inheritance hierarchy

When you build an object that inherits from another object, you are creating a hierarchy (or tree structure) of ancestor objects and descendent objects. Chapter 11, "Working with Windows," uses the example of creating two windows, w_customer and w_employee, that inherit their properties from a common ancestor, w_ancestor. In this example, w_employee and w_customer are the descendants.

The object at the top of the hierarchy is a base class object, and the other objects are descendants of this object. Each descendant inherits information from its ancestor. The base class object typically performs generalized processing, and each descendant modifies the inherited processing as needed.

Multiple descendants

An object can have an unlimited number of descendants, and each descendant can also be an ancestor. For example, if you build three windows that are direct descendants of the w_ancestor window and three windows that are direct descendants of the w_employee window, the hierarchy looks like this:

Figure 13-1: Object hierarchy example
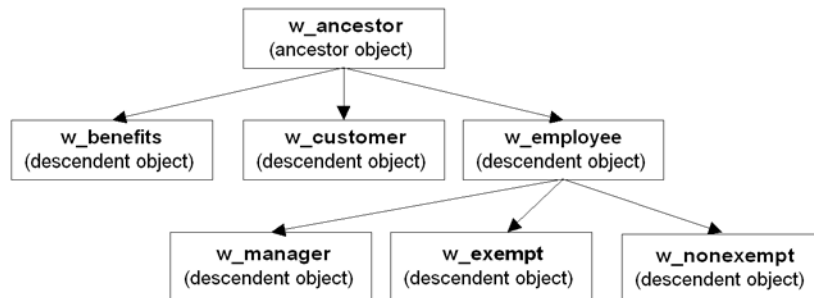
Browsing the class hierarchy

PowerBuilder provides a Browser that can show the hierarchy of the built-in PowerBuilder system objects and the hierarchy of ancestor and descendent windows, menus, and user objects you create. In object-oriented terms, these are called class hierarchies: each PowerBuilder object defines a class.

Regenerating objects
The Browser also provides a convenient way to regenerate objects and their descendants. For more information, see "Regenerating library entries" on page 157.

❖ **To browse the class hierarchy of PowerBuilder system objects:**

1 Click the Browser button in the PowerBar.

2 Choose the System tab to show the built-in PowerBuilder objects.

3 In the left pane, scroll down the object list and select the powerobject.

4 Display the pop-up menu for the powerobject and choose Show Hierarchy.

5 Select Expand All from the pop-up menu and scroll to the top.

The hierarchy for the built-in PowerBuilder objects displays.
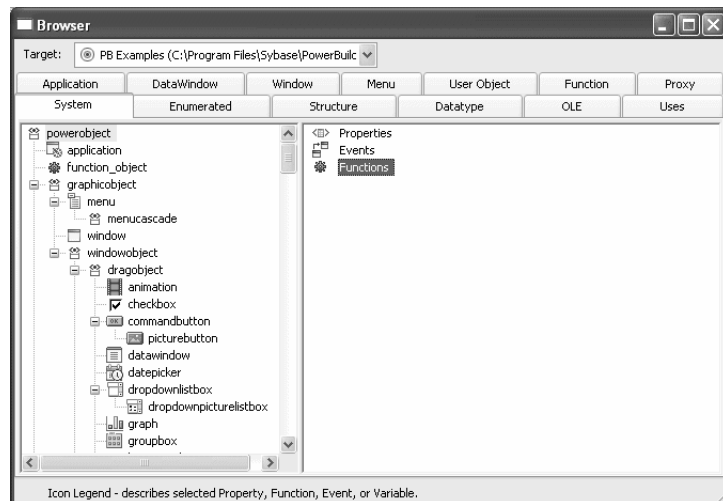
Getting context-sensitive Help in the Browser
To get context-sensitive Help for an object, control, or function, select
Help from its pop-up menu.

❖ **To display the class hierarchy for other object types:**

1   Choose the Menu, Window, or User Object tab.

If you choose any other object type, there is no inheritance for the object
type, so you cannot display a class hierarchy.

2   In the left pane, select an object and choose Show Hierarchy from its
pop-up menu.

3   Select an object and choose Expand All from its pop-up menu.

PowerBuilder shows the selected object in the current application.
Descendent objects are shown indented under their ancestors.

For example, if your application uses the PBDOM PowerBuilder extension
object, the pbdom_object displays on the User Object page. You can select
Show Hierarchy and Expand All from its pop-up menu to display its
descendent objects.
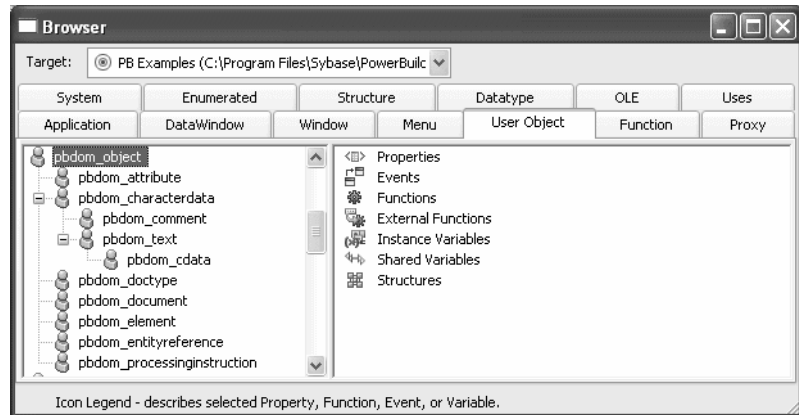
Working with inherited objects

This section describes:

- Working in a descendent object
- Working in an ancestor object
- Resetting properties in a descendant

Working in a
descendent object

You can change descendent objects to meet specialized needs. For example, you can:

- Change properties of the descendent object
- Change properties of inherited controls in the object
- Add controls to a descendent window or user object
- Add menu items to a menu

You cannot copy a control on a descendent window or visual user object if the control inherits from the ancestor object, because the resulting inheritance hierarchy cannot be maintained consistently. You *can* copy a control on a descendent object if the control does *not* inherit from the object's ancestor.

For specifics about what you can do in inherited windows, user objects, and menus, see Chapter 11, "Working with Windows," Chapter 15, "Working with User Objects," and Chapter 14, "Working with Menus and Toolbars."

Working in an
ancestor object

When you use inheritance to build an object, the descendant is dependent on the definition of the ancestor. Therefore you should not delete the ancestor without deleting the descendants. You should also be careful when you change the definition of an ancestor object. You may want to regenerate descendent objects if you do any of the following:

- Delete or change the name of an instance variable in the ancestor
- Modify a user-defined function in the ancestor
- Delete a user event in an ancestor
- Rename or delete a control in an ancestor

When you regenerate the descendants, the compiler will flag any references it cannot resolve so you can fix them. For information about regenerating objects, see Chapter 6, "Working with Libraries."

About local changes
If you change a property in an ancestor object, the property also changes in all descendants—*if* you have not already changed that property in a descendant, in which case the property in the descendant stays the same. In other words, local changes always override inherited properties.

Using inherited scripts

In the hierarchy formed by ancestors and descendants, each descendant inherits its event scripts from its immediate ancestor. If an inherited event does not have a script, you can write a script for the event in the descendant. If the inherited event does have a script, the ancestor script will execute in the descendant unless you extend the script or override it. You can:

•   Extend the ancestor script—build a script that executes *after* the ancestor script

•   Override the ancestor script—build a script that executes *instead* of the ancestor script

You cannot delete or modify an ancestor script from within a descendant.

Extending or overriding a script

The Extend Ancestor Script item on the Edit menu or the pop-up menu in the Script view determines whether the script is extended or overridden. If the menu item is selected (a check mark displays next to it), the ancestor script is extended. If there is no check mark, the ancestor script is overridden.

When there is no script for the descendant, the Extend Ancestor Script menu item is selected and disabled. You cannot clear the menu item unless you add a script to the descendant. When you have added a script, the menu item is enabled and you can choose to override the ancestor script by clearing the menu item, or to extend it by leaving the menu item selected.

If you delete the script in the descendant
If, after adding a script to the descendant and clearing the Extend Ancestor Script menu item, you delete the script, the menu item returns to its default state: selected and disabled. A message displays in the status bar warning you that this has occurred. If you then add a new script, the menu item is reenabled. You must remember to clear the Extend Ancestor Script menu item if you want to override the ancestor script.

Executing code before the ancestor script	To write a script that executes *before* the ancestor script, first override the ancestor script and then in the descendent script explicitly call the ancestor script at the appropriate place. For more information, see "Calling an ancestor script" on page 315.
Getting the return value of the ancestor script	To get the return value of an ancestor script, you can use the AncestorReturnValue variable. This variable is always available in descendent scripts that extend an ancestor script. It is also available if you override the ancestor script and use the CALL syntax to call the ancestor event script. For more information, see *Application Techniques*.

Viewing inherited scripts

If an inherited object or control has a script defined only in an ancestor, no script displays in the Script view.

Script icons in the second drop-down list	The second drop-down list in the Script view indicates which events have scripts written for an ancestor as follows:

- If the event has a script in an ancestor only, the script icon next to the event name in the second drop-down list is displayed in color.

- If the event has a script in an ancestor as well as in the object you are working with, the script icon is displayed half in color.

Script icons in the third drop-down list	The third drop-down list in the Script view shows the current object followed by each of its ancestors in ascending order. The icons next to object names indicate whether the object has a script for the event selected in the second drop-down list as follows:

- If an object is the highest class in the hierarchy to have a script, a transparent script icon displays next to its name. No icon displays next to the names of any of its ancestors.

- If an object does not have a script for the event but it has an ancestor that has a script for the event, the script icon next to its name is displayed in color.

- If an object has a script for the event, and it has an ancestor that also has a script for the event, the script icon next to its name is displayed half in color.

❖ **To view an ancestor script:**

1   In the Script view for an inherited object, select the object itself or a control in the first drop-down list, and the event whose script you want to see in the second drop-down list.

    The Script view does not display the script for the ancestor. No script displays.

2   In the third drop-down list in the Script view, select an ancestor object that has a script for the selected event.

    The Script view displays any script defined in the ancestor object.

3   To climb the inheritance hierarchy, in the third drop-down list, select the script for the grandparent of the current object, great-grandparent, and so on until you display the scripts you want.

    The Script view displays the scripts for each of the ancestor objects. You can traverse the entire inheritance hierarchy using the third drop-down list.

Extending a script

When you extend an ancestor script for an event, PowerBuilder executes the ancestor script, *then* executes the script for the descendant when the event is triggered.

❖ **To extend an ancestor script:**

1   In the first drop-down list in the Script view, select the object or a control, and in the second drop-down list, select the event for which you want to extend the script.

2   Make sure that Extend Ancestor Script on the Edit menu or the pop-up menu in the Script view is selected.

    Extending the ancestor script is the default.

3   In the Script view, enter the appropriate statements.

    You can call the script for any event in any ancestor as well as call any user-defined functions that have been defined for the ancestor. For information about calling an ancestor script or function, see "Calling an ancestor script" on page 315 and "Calling an ancestor function" on page 315.

Example of extending
a script

If the ancestor script for the Clicked event in a button beeps when the user clicks the button without selecting an item in a list, you might extend the script in the descendant to display a message box in addition to beeping.

Overriding a script

❖ **To override an ancestor script:**

1   In the first drop-down list in the Script view, select the object or a control, and in the second drop-down list, select the event for which you want to override the script.

2   Code a script for the event in the descendant.

You can call the script for any event in any ancestor as well as call any user-defined functions that have been defined for the ancestor.

For information about calling an ancestor script or function, see "Calling an ancestor script" on page 315 and "Calling an ancestor function" on page 315.

Override but not execute
To override a script for the ancestor but not execute a script in the descendant, enter only a comment in the Script view.

3   Select Extend Ancestor Script on the Edit menu or the pop-up menu to clear the check mark.

Clearing the Extend Ancestor Script item means that you are overriding the script.

At runtime, PowerBuilder executes the descendent script when the event is triggered. The ancestor script is not executed.

Example of overriding
a script

If the script for the Open event in the ancestor window displays employee files and you want to display customer files in the descendent window, select Override Ancestor Script and create a new script for the Open event in the descendant to display customer files.

Calling an ancestor script

When you write a script for a descendent object or control, you can call scripts written for any ancestor. You can refer by name to any ancestor of the descendent object in a script, not just the immediate ancestor (parent). To reference the immediate ancestor (parent), you can use the Super reserved word.

For more information about calling scripts for an event in an ancestor window, user object, or menu, and about the Super reserved word, see the *PowerScript Reference*.

Calling an ancestor function

When you write a script for a descendent window, user object, or menu, you can call user-defined functions that have been defined for any of its ancestors. To call the first function up the inheritance hierarchy, just call the function as usual:

function (*arguments*)

If there are several versions of the function up the inheritance hierarchy and you do not want to call the first one up, you need to specify the name of the object defining the function you want:

ancestorobject::*function* (*arguments*)

This syntax works only in scripts for the descendent object itself, not in scripts for controls or user objects in the descendent object or in menu item scripts. To call a specific version of an ancestor user-defined function in a script for a control, user object, or menu item in a descendent object, do the following:

1   Define an object-level user-defined function in the descendent object that calls the ancestor function.

2   Call the function you just defined in the descendent script.

For more information about calling an ancestor function, see the *PowerScript Reference*.

Working with Menus and Toolbars

About this chapter

By adding customized menus and toolbars to your applications, you can make it easy and intuitive for your users to select commands and options. This chapter describes how to define and use menus and toolbars.

Contents

Menus and menu items

Usually, all windows in an application have menus except child and response windows. Menus are lists of related commands or options (menu items) that a user can select in the currently active window. Each choice in a menu is called a menu item. Menu items display in a menu bar or in drop-down or cascading menus.

About Menu objects
Each item in a menu is defined as a Menu object in PowerBuilder. You can see the Menu object in the list of objects in the Browser's System tab.

Using menus

You can use menus you build in PowerBuilder in two ways:

- **In the menu bar of windows**   Menu bar menus are associated with a window in the Window painter and display whenever the window is opened.

- **As pop-up menus**   Pop-up menus display only when a script executes the PopMenu function.

Both uses are described in this chapter.

Designing menus

PowerBuilder gives you complete freedom in designing menus, but you should follow conventions to make your applications easy to use. For example, you should keep menus simple and consistent; group related items in a drop-down menu; make sparing use of cascading menus and restrict them to one or two levels.

This chapter describes some guidelines you should follow when designing menus. A full discussion of menu design is beyond the scope of this book. You should acquire a book that specifically addresses design guidelines for graphical applications and apply the rules when you use PowerBuilder to create your menus.

Building menus

When you build a menu, you:

- Specify the appearance and behavior of the menu items by setting their properties.

- Build scripts that determine how to respond to events in the menu items. To support these scripts, you can declare functions, structures, and variables for the menu.

There are two ways to build a menu. You can:

- Build a new menu from scratch. See "Building a new menu" on page 323.

- Build a menu that inherits its style, functions, structures, variables, and scripts from an existing menu. You use inheritance to create menus that are derived from existing menus, thereby saving yourself coding and time. See "Using inheritance to build a menu" on page 350.

Using the Menu painter

The Menu painter has several views where you can specify menu items and how they look and behave. For general information about views, how you use them, and how they are related, see "Views in painters that edit objects" on page 116.

In addition to customizing the style of a menu in the Menu painter, you can also customize the style of a toolbar associated with a menu. For information, see "Providing toolbars" on page 337.

Menu painter views

You use two views to specify menu items that display in the menu bar and under menu bar items:

Table 14-1: Views in the Menu painter

This view	Displays
Tree Menu view	All the menu items at the same time when the tree is fully expanded. To fully expand the tree or collapse the expanded tree, press Ctrl+Shift+*.
WYSIWYG Menu view	The menu as you will see it and use it in your application, with the exception of invisible menu items that do display.

The Tree Menu view and the WYSIWYG Menu view are equivalent. You can use either view to insert new menu items on the menu bar or on drop-down or cascading menus, or to modify existing menu items. The menus in both views change when you make a change in either view.

You specify menu properties in two views:

Table 14-2: Views in the Menu painter

This view	Displays
Properties view (for the top-level menu object)	General and Appearance tab pages for setting menu-wide properties
Properties view (for submenu items)	General and Toolbar tab pages for setting properties for submenu items and toolbars

Views for the top level
menu object

This Menu painter layout is for the top level menu object, m_pbapp_frame.
The Tree Menu view is in the top left and the WYSIWYG Menu view is in the
top middle. The General and Appearance tab pages display in the Properties
view on the right. For more information about these properties, see "Setting
menu style properties for contemporary menus" on page 334.
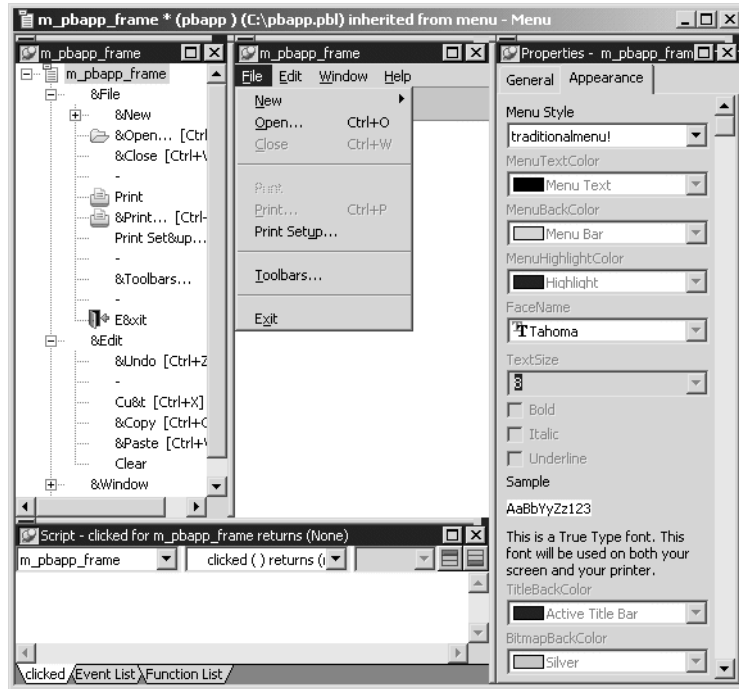
Views for submenu items

This Menu painter layout is for a menu item under the top level, in this case the Open menu item. The Tree Menu view is in the top left and the WYSIWYG Menu view is in the top middle. The General and Toolbar tab pages display in the Properties view on the right. For more information about these properties, see "Setting General properties for menu items" on page 331.
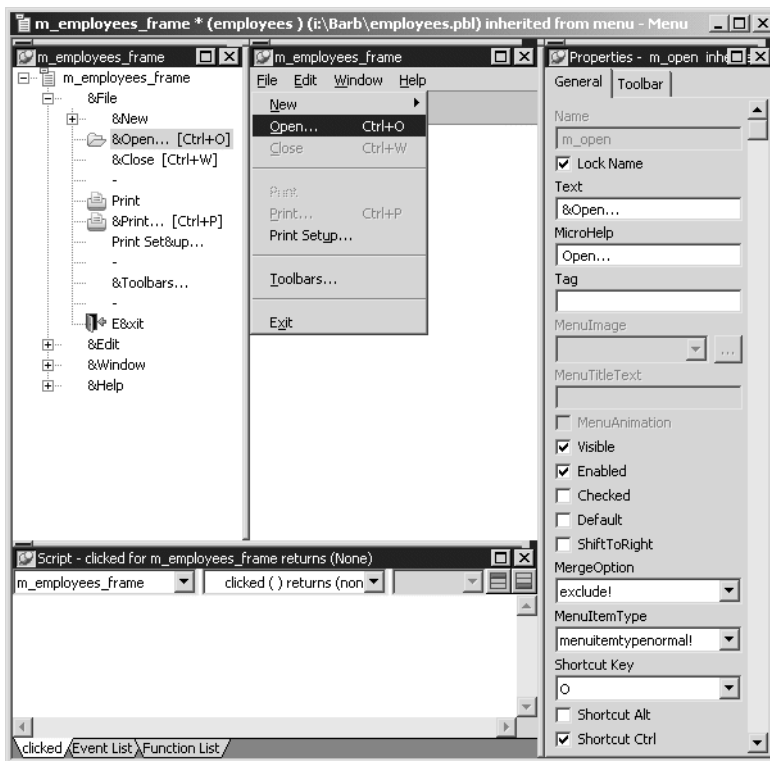
Menu styles

A menu can have a contemporary or traditional style.

Menu style	Description
Contemporary	A 3D-style menu similar to Microsoft Office 2003 and Visual Studio 2005 menus
Traditional	Window default menu style which has a flat appearance

Menus that you import or migrate from earlier versions of PowerBuilder have the traditional style, and new menus use the traditional menu style by default. The new contemporary menu style has a three-dimensional menu appearance that can include images and menu title bands. With a contemporary menu, you can set the MenuAnimation, MenuImage, and MenuTitleText at runtime using scripts.

You select a menu style on the Appearance tab of the Properties view for the top-level menu object in the Menu painter. You must select the top-level menu object in the Tree Menu view of the Menu painter to display its Properties view.

❖ **To specify the menu style:**

1 Select the top-level menu object.

2 In the Appearance tab page, select the menu style you want, contemporarymenu! or traditionalmenu!
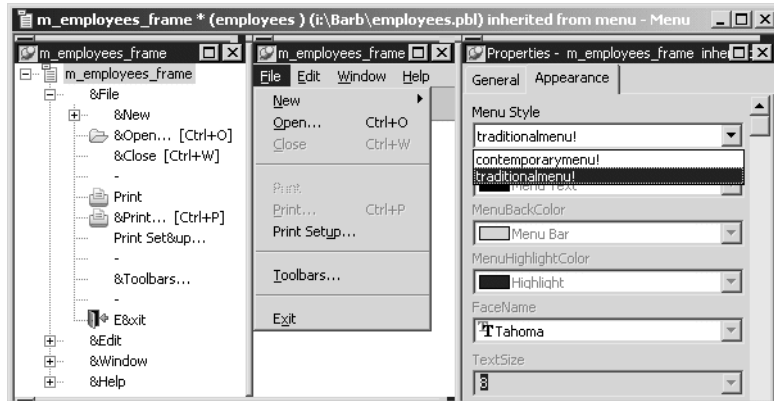
If you select contemporarymenu! in the Menu Style drop-down list, you can customize the display properties for that style and have them apply to all menu items in the current menu. If you select traditionalmenu! the rest of the menu style properties are grayed.

Images for menus and toolbars

Contemporary menus can include images. You can use icons, bitmaps, GIF files, and JPEG files for both contemporary menus and traditional and contemporary toolbars.

All stock icons have a transparent background. Other icon and GIF files with transparent backgrounds are always displayed with a transparent background. If you want a bitmap to display with a transparent background, the bitmap must use button face as its background color. This applies whatever the background color of the menu or toolbar is set to. There is currently no property that allows you to specify that an image has a transparent background.

When an icon file includes several images

With the contemporary menu style and toolbar style, when an icon file includes several images, PowerBuilder uses the following image selection rules:

If the images . . .	PowerBuilder displays . . .
Are all the same size	8 bit, 16 bit, 24 bit, 32 bit, and 4 bit images in that order.
Include 16 bit*16 bit images and also other sized images	16 bit*16 bit images.
Do not include 16 bit*16 bit images	The image with the image size closest to 16 bit. For example, if one icon file contains 24*24 bit images and another icon file contains 32*32 bit images, then PowerBuilder selects the 24*24 bit images.
Are greater than 16 bit images	The image as 16*16 bit or 32*32 bit. If the icon image is 16*16, then it displays as 16*16. If the icon image is larger than 16*16, it will be displayed as 32*32.

Building a new menu

This section describes how to build menus that are not based on existing menus. To create a new menu using inheritance, see "Using inheritance to build a menu" on page 350.

Creating a new menu

You build a new menu by creating a new Menu object and then working on it in the Menu painter.

❖ **To create a new menu:**

1   Click the New button in the PowerBar.

2   On the PB Object tab page, select Menu.

3   Click OK.

The Menu painter opens and displays the Tree Menu view and the WYSIWYG view for defining the menu, and the General and Appearance tab pages for setting menu and toolbar properties. For information about menu and toolbar properties, see "Defining the appearance and behavior of menu items" on page 331.

Because you are creating a new menu and have not added menu items yet, the only content in the Tree Menu view and the WYSIWYG view is an untitled top-level tree view item in the TreeMenu view.

Font size of the menu bar and menu text
You can change the value of the TextSize property for submenu items, but not for the main menu bar. The main menu bar has a fixed height that you cannot change.

Working with menu items

A menu consists of at least one menu item on the menu bar and menu items in a drop-down menu. You can add menu items in three places:

- To the menu bar

- To a drop-down menu

- To a cascading menu

Using the pop-up menu
The procedures in this section use the Insert and Edit menus on the PowerBuilder main menu to insert and edit menu items. You can also use the equivalent items on the selected object's pop-up menu.

How menu items are named

When you add a menu item, PowerBuilder gives it a default name, which displays in the Name box in the Properties view. This is the name by which you refer to a menu item in a script.

About the default menu item names

The default name is a concatenation of the default prefix for menus, m_, and the valid PowerBuilder characters and symbols in the text you typed for the menu item. If there are no valid characters or symbols in the text you typed for the menu item, PowerBuilder creates a unique name m_*n*, where *n* is the lowest number that can be combined with the prefix to create a unique name.

Prefix might be different
The default prefix is different if it has been changed in the Design>Options dialog box.

The complete menu item name (prefix and suffix) can be up to 40 characters. If the prefix and suffix exceed this size, PowerBuilder uses only the first 40 characters without displaying a warning message.

Duplicate menu item names

Menu items in the Tree Menu view and WYSIWYG Menu view can have the same names, but they cannot have the same name in the Properties view. If you try to add a menu item using the same name as an existing menu item, PowerBuilder displays a dialog box that suggests a unique name for the menu item. For example, you might already have an Options item on the Edit menu with the default name m_options. If you add an Options item to another menu, PowerBuilder cannot give it the name m_options.

Menu item names are locked by default

After you add a menu item, the name that PowerBuilder assigns to the menu item is locked. Even if you later change the text that displays for the menu item, PowerBuilder does not rename the menu item. This allows you to change the text that displays in a menu without having to revise all your scripts that reference the menu item. (Remember, you reference a menu item through the name that PowerBuilder assigns to it.)

To rename a menu item after changing the text that displays for it, you can unlock the name.

❖   **To have PowerBuilder rename a menu item:**

1   On the General property page in the Properties view, clear the Lock Name check box.

2   Change the text that displays for the menu item.

Inserting menu items

There are three choices on the Insert menu: Menu Item, Menu Item At End, and Submenu Item. Use the first two to insert menu items in the same menu as the selected item, and use Insert>Submenu Item to create a new drop-down or cascading menu for the selected item.

For example, suppose you have created a File menu on the menu bar with two menu items: Open and Exit. Here are the results of some insert operations:

• Select File and select Insert>Menu Item At End

  A new item is added to the menu bar after the File menu.

• Select Open and select Insert>Menu Item

  A new item is added to the File menu above Open.

- Select Open and select Insert>Menu Item At End

  A new item is added to the File menu after Exit.

- Select Open and select Insert>Submenu Item

  A new cascading menu is added to the Open menu item.

Getting the menu started

The first thing you do with a new menu is add the first item to the menu bar. After doing so, you can continue adding new items to the menu bar or to the menu bar item you just created. As you work, the changes you make display in both the WYSIWYG and Tree Menu views.

The first procedure in this section describes how to add a single first item to the menu bar. Use this procedure if you want to add the menu bar item and then work on its drop-down menu. Use the second procedure to add multiple items to the menu bar quickly.

❖ **To insert the first menu bar item in a new menu:**

1   Select the first menu item, and then select Insert>Submenu Item from the PowerBuilder menu bar.

    PowerBuilder displays an empty box on the menu bar in the WYSIWYG Menu view and as a sub-item in the Tree Menu view.

2   Type the text you want for the menu item and press Enter.

❖ **To insert multiple menu bar items in a new menu:**

1   Select Insert>Submenu Item.

    PowerBuilder displays an empty box on the menu bar in the WYSIWYG Menu view and as a submenu item in the Tree Menu view.

2   Type the text you want for the menu item and press Tab.

    PowerBuilder displays a new empty box on the menu bar in the WYSIWYG Menu view and as a submenu item in the Tree Menu view.

3   Repeat step 2 until you have added all the menu bar items you need.

4   Press Enter to save the last menu bar item.

Adding additional menu items

After you have created the first menu bar item, you can add more items to the menu bar or start building drop-down and cascading menus.

❖ **To insert additional menu items on the menu bar:**

1   Do one of the following:

- With any menu bar item selected, select Insert>Menu Item At End to add an item to the end of the menu bar.

- Select a menu bar item and select Insert>Menu Item to add a menu bar item before the selected menu bar item.

2   Type the text you want for the menu bar item, and then press Enter.

❖ **To add a drop-down menu to an item on the menu bar:**

1   Select the item in the menu bar for which you want to create a drop-down menu.

2   Select Insert>Submenu Item.

PowerBuilder displays an empty box.

3   Type the text you want for the menu item, and then press Tab.

4   Repeat Step 3 until you have added all the items you want on the drop-down menu.

5   Press Enter to save the last drop-down menu item.

❖ **To add a cascading menu to an item in a drop-down menu:**

1   Select the item in a drop-down menu for which you want to create a cascading menu.

2   Select Insert>Submenu Item.

PowerBuilder displays an empty box.

3   Type the text you want for the menu item, and then press Tab.

4   Repeat step 3 until you have added all the items you want on the cascading menu.

5   Press Enter to save the last cascading menu item.

❖ **To add an item to the end of any menu:**

1   Select any item on the menu.

2   Select Insert>Menu Item At End.

PowerBuilder displays an empty box.

3   Type the text you want for the second menu item in the box and press Enter.

❖ **To insert an item in any existing menu:**

1 Select the item that should follow the new menu item.

2 Select Insert>Menu Item.

An empty box displays above the item you selected.

3 Type the text you want for the menu item and press Enter.

Creating separation lines in menus

You should separate groups of related menu items with lines.

❖ **To create a line between items on a menu:**

1 Insert a new menu item where you want the separation line to display.

2 Type a single dash (-) as the menu item text and press Enter.

A separation line displays.

Duplicating menu items

You might save time creating new menu items if you duplicate existing menu items. A duplicate menu item has the same properties and script as the original menu item. You might be able to modify a long script slightly to make it work for your duplicate menu item.

❖ **To duplicate a menu item or a submenu item:**

1 Select the menu item or the submenu item to duplicate.

2 Select Edit>Duplicate or press Ctrl+T.

The duplicate item displays at the same level of the menu, following the item you selected. The name of the duplicate menu item is unique.

3 Change the text of the duplicate menu item.

4 Modify the properties and script associated with the duplicate item as needed.

Changing menu item text

It is often necessary to change the text of a menu item, and if you duplicate a menu item, you need to change the text of the duplicate item.

❖ **To change the text of a menu item:**

1   Do one of the following:

- Click the item to select it, then click it again.

- Select the item and select Edit>Menu Item Text.

- Select the item and open the general page in the Properties view.

2   Type the new text for the menu item in the box in the WYSIWYG Menu or Tree Menu view or in the Text box in the Properties view.

Selecting menu items

You can select multiple menu items to move them, delete them, or change their common properties.

❖ **To select multiple individual menu items:**

- Press Ctrl and select each item you want.

❖ **To select a range of menu items at the same level in the menu:**

- Select the first item, press Shift, and select the last item.

Navigating in the menu

As you work in a menu, you can move to another menu item by selecting it. You can also use the Right Arrow, Left Arrow, Up Arrow, and Down Arrow keys on the keyboard to navigate.

Moving menu items

The easiest way to change the order of items in the menu bar or in a drop-down or cascading menu is to drag the item you want to move and drop it where you want it to be. You can drag items at the same level in a menu structure or to another level. For example, you can drag an item in the menu bar to a drop-down menu or an item in a cascading menu to the menu bar.

WYSIWYG Menu and Tree Menu views
You can use drag and drop within each view. You can also drag from one view and drop in another.

❖ **To move a menu item or submenu item using drag and drop:**

1   Select the item.

2   Press and hold the left mouse button and drag the item to a new location.

    A feedback line appears at the new location to indicate where to drop the item.

3   Release the mouse button to drop the menu item.

    The menu item displays in the new location.

Dragging to copy
To copy a menu item by dragging it, press and hold the Ctrl key while you drag and drop the item. A copied menu item has the same properties and scripts as the original menu item.

You can also copy or move a menu item by selecting the item and using the Cut, Copy, and Paste items on the Edit menu or the pop-up menu.

Deleting menu items

❖ **To delete a menu item:**

1   Select the menu item you want to delete.

2   Click the Delete button in the PainterBar or select Edit>Delete from the menu bar.

Saving the menu

You can save the menu you are working on at any time. When you save a menu, PowerBuilder saves the compiled menu items and scripts in the library you specify.

❖ **To save a menu:**

1 Select File>Save from the menu bar.

If you have previously saved the menu, PowerBuilder saves the new version in the same library and returns you to the Menu painter. If you have not previously saved the menu, PowerBuilder displays the Save Menu dialog box.

2 Name the menu in the Menus box (see "Naming the menu" next).

3 Write comments to describe the menu.

These comments display in the Select Menu dialog box and in the Library painter. It is a good idea to use comments so you and others can easily remember the purpose of the menu later.

4 Specify the library in which to save the menu and click OK.

Naming the menu

The menu name can be any valid PowerBuilder identifier of up to 40 characters. For information about PowerBuilder identifiers, see the *PowerScript Reference*.

A common convention is to use m_ as a standard prefix, and a suffix that helps you identify the particular menu. For example, you might name a menu used in a sales application m_sales.

Defining the appearance and behavior of menu items

By setting menu properties, you can customize the display of menus in applications that you create with PowerBuilder. You use the Menu painter to change the appearance and behavior of your menu and menu items by choosing different settings in the tab pages in the Properties view. For a list of all menu item properties, see *Objects and Controls*.

Setting General properties for menu items

This section describes the properties you can set when you select a menu item and then select the General tab page in the Properties view.

Creating MicroHelp
and tags

MicroHelp is a brief text description of the menu item that displays on the status bar at the bottom of a Multiple Document Interface (MDI) application window. Type the text you want to display in the MicroHelp box. For examples of MicroHelp text, select an item from a menu in PowerBuilder and look at the text that displays in the status bar.

A tag is a text string that you can associate with an object and use in any way you want.

For information about defining MicroHelp text and tag properties, see the chapter on building MDI applications in *Application Techniques*.

Setting the
appearance of a menu
item

On the General tab page in the Properties view, you can also specify how a menu item appears at runtime.

Table 14-3: Setting display properties for menu items

Property	Meaning
Visible	Whether the menu item is visible. An invisible menu item still displays in the WYSIWYG and Tree Menu views, but at runtime, it will not display. In WYSIWYG Menu view, an invisible item has faded and dotted text.
Enabled	Whether the menu item can be selected.
Checked	Whether the menu item displays with a check mark next to it.
Default	Whether the menu item text is bold. In a pop-up menu, Default indicates what action occurs if the user double-clicks instead of right-clicks on an item. In dragging, Default indicates what happens when an item is dragged with the left mouse button instead of the right mouse button.
ShiftToRight	Whether the menu item shifts to the right (or down for a drop-down or cascading menu) when you add menu items in a menu that is inherited from this menu. Selecting this property allows you to insert menu items in descendent menus, instead of being able to add them only to the end.
	For more information, see "Inserting menu items in a descendent menu" on page 352.
MergeOption	The way menus are modified when an OLE object is activated. Options are: File, Edit, Window, Help, Merge, Exclude.
	For more information, see the chapter about using OLE in an application in *Application Techniques.*
MenuItemType	Whether the menu item you are creating is Normal, About, Exit, or Help type.

The settings you specify here determine how the menu items display by default. You can change the values of the properties in scripts at runtime.

Assigning accelerator and shortcut keys

Every menu item should have an accelerator key, also called a mnemonic access key, which allows users to select the item from the keyboard by pressing Alt+*key* when the menu is displayed. Accelerator keys display with an underline in the menu item text.

You can also define shortcut keys, which are combinations of keys that a user can press to select a menu item whether or not the menu is displayed.

For example, in the following menu all menu items have accelerator keys: the accelerator key is N for New, O for Open, and so on. New, Open, Close, and Print each have shortcut keys: the Ctrl key in combination with another key or keys.

You should adopt conventions for using accelerator and shortcut keys in your applications. All menu items should have accelerator keys, and commonly used menu items should have shortcut keys.

If you specify the same shortcut for more than one MenuItem, the command that occurs later in the menu hierarchy is executed.

Some shortcut key combinations, such as Ctrl+C, Ctrl+V, and Ctrl+X, are commonly used by many applications. Avoid using these combinations when you assign shortcut keys for your application.

❖ **To assign an accelerator key:**

• Type an ampersand (&) before the letter in the menu item text that you want to designate as the accelerator key.

For example, &File designates the F in File as an accelerator key and Ma&ximize designates the x in Maximize as an accelerator key.

Displaying an ampersand in the text
If you want an ampersand to display in the menu text, type two ampersands. For example, `Fish&&Chips` displays as Fish&Chips with no accelerator key. To display Fish&Chips as the menu text with the C underlined as the accelerator, type `Fish&&&Chips`.

❖ **To assign a shortcut key:**

1 Select the menu item to which you want to assign a shortcut key.

2 Select the General tab in the Properties view.

3 Select a key from the Shortcut Key drop-down list.
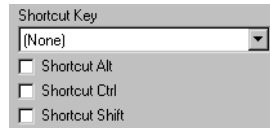
4 Select Shortcut Alt, Shortcut Ctrl, and/or Shortcut Shift to create a key combination.

PowerBuilder displays the shortcut key next to the menu item name.

Setting menu style properties for contemporary menus

Menus with a contemporary style have a three-dimensional menu appearance and can include bitmap and menu title bands. The following figure shows a contemporary style menu:
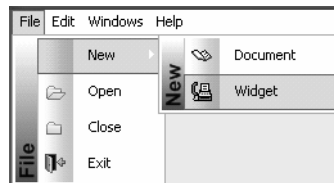
After you select the contemporary style, you can modify other menu style properties on the top-level menu object and on all lower-level menu items. Since it is important to maintain a consistent look across each menu and toolbar, very few style properties are modifiable at the menu item level.

If you select the traditional menu style
If you select traditionalmenu! for the top-level menu object, you cannot modify any of the menu style properties.

You can modify menu style properties only at design time. After you select the contemporary menu style for a top-level menu object, you can select values for other style properties to manipulate a menu's visual appearance. The following properties are modifiable for the top-level menu object only; you cannot modify them for individual menu items:

Property	Datatype	Use to assign
MenuStyle	Enumerated	Overall menu style. Values are: contemporarymenu! and traditionalmenu!
MenuTextColor	Long	Menu text color. (Default is the Windows menu text color.)
MenuBackColor	Long	Background color of the menu.
MenuHighlightColor	Long	Menu highlight color. (Default is the default Windows highlight color.)
FaceName	String	Font face name.
TextSize	Integer	Font character size in points for menu items. (Does not apply to the main menu bar which has a fixed height.)
Bold	Boolean	Bold font.
Italic	Boolean	Italic font.
Underline	Boolean	Underline font.
TitleBackColor	Long	Background color of the title panel.
BitmapBackColor	Long	Background color of the bitmap band of the menu. (Default is silver.)
MenuBitmaps	Boolean	Bitmap band for the menu.
BitmapGradient	Boolean	Background of the bitmap band to a gradient style.
MenuTitles	Boolean	Menu title band.
TitleGradient	Boolean	Background gradient style for the title panel.

Setting menu item style properties

Menu items have style properties that you set at design time. You cannot use these style properties with a traditional style menu. Unlike the style properties on the Menu object that display on the Appearance tab of the Properties view, the fields where you set these properties are located on the General tab of the Properties view for each menu item.

Property	Datatype	Use to assign
MenuAnimation	Boolean	Visual sizing cue to the menu item bitmap when the associated menu item is selected. This property is ignored if the MenuImage property is not assigned.
MenuImage	String	Bitmap image to be used with the menu item. This property is ignored if the MenuBitmaps property for the menu object is not selected or is set to false.
MenuTitleText	String	Label for menu item that has a cascading submenu. The label text is set vertically in a column to the left of the submenu items and the bitmaps for submenu items, if any. If the vertical label text is longer than the height of all the submenu items, the label text is not used. The label text is cut from two ends. This property is ignored if the MenuTitles property for the menu object is not selected.

You select or enter values for the menu item style properties on the General tab of the Properties view for each menu item. You can make selections for the MenuAnimation and MenuImage properties only if the MenuBitmaps check box for the current menu object is selected. The MenuBitmaps check box is selected by default for the contemporary menu style.

You can enter text for the MenuTitleText property only if the MenuTitles check box for the current menu object is selected.

Providing toolbars

To make your application easier to use, you can add toolbars with buttons that users can click as a shortcut for choosing an item from a menu. In PowerBuilder, you can associate a toolbar with the window types listed in Table 14-4.

Table 14-4: Window types that can use menus

Window type	Description
Main window	A main window is a standalone window that can be independent of all other windows. You use the main window as the anchor for your application. The first window your application opens is a main window (unless you are building an MDI application, in which case the first window is an MDI frame window).
MDI window	A window in a Multiple Document Interface application, an application in which users work within a frame window that lets them perform activities on multiple sheets of information. This is useful in applications where users require the ability to do several different things at a time. An MDI frame window has a menu bar, a client area, sheets, and (usually) a toolbar. An MDI sheet window is a window that can be opened in the client area of an MDI frame.
MDI Help window	An MDI window with a status area that can display MicroHelp.

Creating windows in PowerBuilder

You can create a main window, an MDI window, or an MDI Help window in PowerBuilder by clicking the New button in the PowerBar and selecting Window on the PB Object tab page. The new window's type is Main by default. To change it to MDI or MDI Help, select the window type on the General page in the Properties view.

In MDI windows, you can associate a toolbar with the MDI frame and a toolbar with the active sheet. This screen shows New, Print, and Exit buttons on the toolbar associated with the MDI Frame, and window management buttons on the toolbar associated with the sheet. The toolbar associated with the MDI frame is called the **FrameBar**. The toolbar associated with the active sheet is called the **SheetBar**.
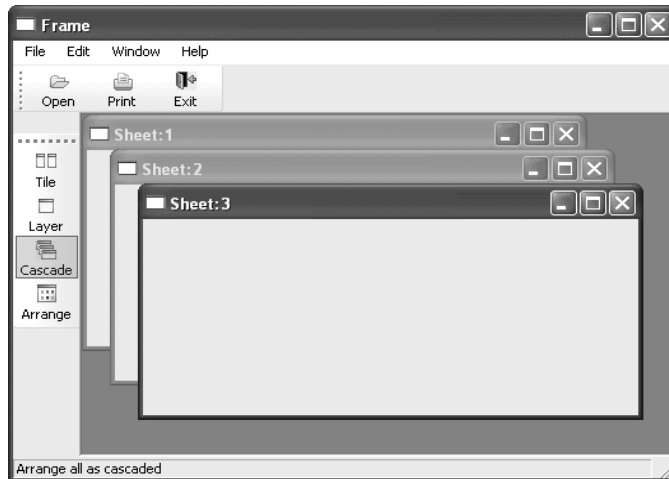
This section provides you with the information you need to create and use toolbars. For information about customizing toolbar behavior and saving and restoring toolbar settings, see *Application Techniques*.

How toolbars work

Toolbars you add to a window behave like the toolbars provided in the PowerBuilder development environment:

- Users can choose whether or not to display text in a toolbar, use PowerTips, float the toolbar, move the toolbar around the frame, and dock it underneath or beside any other toolbar. No coding is required to support these basic toolbar operations.

- Toolbar buttons map directly to menu items. Clicking a toolbar button is the same as clicking its corresponding menu item (or pressing the accelerator key for that item).

- Toolbars work only in MDI frame, MDI sheet, and Main windows. If you open a pop-up window with a menu that has a toolbar, the toolbar does not display.

- If both the MDI sheet and the frame have toolbars and the sheet is open, then the menu that is displayed is the menu for the sheet, but both toolbars appear and are operative.

- If the currently active sheet does not have a menu, then the menu and toolbar for the frame remain in place and are operative. This can be confusing to your user, because the displayed menu is not for the active sheet. If any sheet has a menu, then all sheets should probably have menus.

Menus with multiple toolbars

A single menu can have more than one toolbar. When you associate a menu that has multiple toolbars with a window, PowerBuilder displays all the toolbars when you open the window. This screen shows a sheet open in an MDI frame, with one FrameBar and two SheetBars:
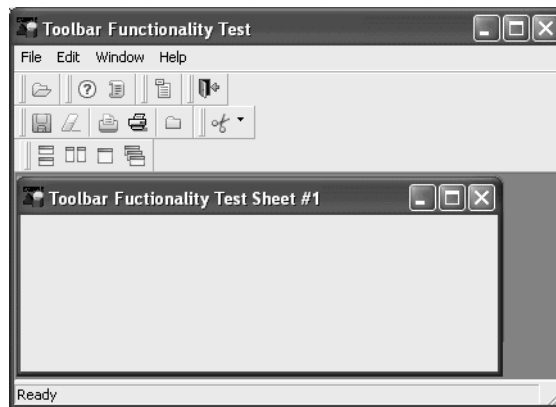
You can work with the toolbars independently. For example, you can float any of the toolbars, move them around the window, and dock them at different locations within the window.

The button associated with a menu item can appear on only one toolbar at a time. To indicate which toolbar a menu item's button belongs to, you set the ToolbarItemBarIndex property for the menu item. All items that have the same index number appear on the same toolbar.

Adding toolbars to a window

PowerBuilder provides an easy way to add toolbars to a window: when you are defining an item in the Menu painter for a menu that will be associated with an MDI frame window, an MDI sheet, or a main window, you simply specify that you want the menu item to display in the toolbar with a specific picture. At runtime, PowerBuilder automatically generates a toolbar for the window containing the menu.

❖ **To add toolbars to a window:**

1 In the Menu painter, specify the display characteristics of the menu items you want to display in the toolbar.

For details, see "Toolbar item display characteristics" next.

2 (Optional) In the Menu painter, specify drop-down toolbars for menu items.

3 In the Window painter, associate the menu with the window and turn on the display of the toolbar.

4 (Optional) In the Window painter, specify other properties, such as the size and location of a floating toolbar, on the Toolbar property page.

Selecting a toolbar style

You select a toolbar style on the Appearance tab of the Properties view for the top-level menu object in the Menu painter.

A toolbar can have a contemporary or traditional style.

Menu style	Description
Contemporary	A 3D-style toolbar similar to Microsoft Office 2003 and Visual Studio 2005 toolbars
Traditional	A more traditional and older toolbar style

Toolbars that you import or migrate from earlier versions of PowerBuilder have the traditional style, and new toolbars use the traditional toolbar style by default.

❖ **To specify the toolbar style:**

1   Select the top-level menu object.

2   At the bottom of the Appearance tab page, select the toolbar style you want, contemporarytoolbar! or traditionaltoolbar!

If you select `traditionaltoolbar!` in the Toolbar Style drop-down list, the rest of the toolbar style properties are grayed. If you select `contemporarytoolbar!` style, you can customize the display properties for that style and have them apply to all menu items with associated toolbar buttons in the current menu.

Selecting the toolbar button style property   Unless you are using the traditional toolbar style for the current menu object, you can select the ToolbarAnimation check box on the Toolbar tab or the Properties view for each menu item. If you do not select an image for the ToolbarItemName property of a menu item, the selection you make for the ToolbarAnimation property is ignored.

Setting toolbar properties

You can customize the display of toolbars in applications that you create with PowerBuilder by setting toolbar properties.

Toolbar style properties

In addition to customizing the style of a menu, you can customize the style of a toolbar associated with the menu. For example, the following picture shows a contemporary style toolbar with an expanded toolbar cascade and a highlighted Exit button:
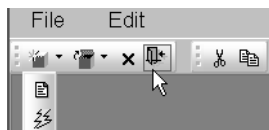
Toolbar style properties   Toolbars have style properties that you can change at design time on the top-level menu object. You can modify these properties only if you select `contemporarytoolbar!` as the toolbar style for the top-level menu object.

Property	Datatype	Use to assign
ToolbarBackColor	Long	Background color of the menu toolbar.
ToolbarGradient	Boolean	Gradient of the menu toolbar background.
ToolbarHighlightColor	Long	Highlight color for the toolbar buttons when they are selected.
ToolbarStyle	Enumerated	Overall style of the menu toolbar. Values are: contemporarytoolbar! and traditionaltoolbar!
ToolbarTextColor	Long	Color of the text in the menu toolbar.

Toolbar item style property   You can select the ToolbarAnimation property for a menu item toolbar button. This property offsets the button image by two pixels to the upper left when a user positions the cursor over the button. You cannot assign this property at the menu object or toolbar level. You must assign it to individual toolbar items (buttons) at design time. This property has a Boolean datatype. You can select it on the Toolbar tab for each menu item below the top-level menu object. With a contemporary menu, you can set the ToolbarAnimation property at runtime at runtime using scripts.

The customizable menu and toolbar styles can be used for MDI and main windows. Pop-up menus can also use menu style properties. The styles do not affect existing PowerBuilder applications that use a traditional style. You can, however, update an existing PowerBuilder application to use the new style properties.

Toolbar item display characteristics

In the Menu painter, you specify the menu items you want to display in the toolbar, the text for the toolbar button and tip, the pictures to use to represent the menu items, and other characteristics of the toolbar.

❖ **To specify the display characteristics of a toolbar item:**

1   Open the menu in the Menu painter and select the menu item you want to display in the toolbar.

2   Select the Toolbar property page and set properties of the toolbar item as shown in Table 14-5.

Table 14-5: Toolbar properties in the Menu painter

Property	What to specify
ToolbarItemText	Specify two different text values for the toolbar button and PowerTip with a comma-delimited string, as follows: *Text in button, PowerTip*
ToolbarItemName	Choose a stock picture from the drop-down list or browse to select a bitmap, GIF, JPEG or icon file. If you choose a stock picture, PowerBuilder uses the up version when the item is not clicked and the down version when the item is clicked. (The up version appears raised and the down version appears indented.) If you are specifying a file, the picture should be 16 pixels wide and 15 pixels high. For the best result, use 16*16 icons on a toolbar.
ToolbarItemDownName	Specify a different picture to display when the item is clicked (optional).
ToolbarItemVisible	Clear if you want the toolbar button to be hidden. The default is visible.
ToolbarItemDown	Check if you want the down version of the button to display initially.
ToolbarAnimation	If you want the toolbar image to be animated when selected, select the check box.
ToolbarItemSpace	Specify any integer if you want to leave space before the picture in the toolbar. Experiment with values to get the spacing the way you want it. If you leave the value at 0, there will be no spacing before the picture. (Spacing is used only when the toolbar is not displaying text.)
ToolbarItemOrder	Specify the order in which the item displays in the toolbar. If you leave the value 0, PowerBuilder places the item in the order in which it goes through the menu to build the toolbar.

Property	What to specify
ToolbarItemBarIndex	Specify a number other than 1 if you want multiple toolbars to display for this menu. The number you specify determines which toolbar the current menu item appears on. All items that have the same index number appear on the same toolbar.
ObjectType	Specify Menu or MenuCascade.
Columns	(This property displays only if you choose MenuCascade in the ObjectType drop-down list.) Indicate the number of columns you want to display in the cascading toolbar.
Drop Down	(This property displays only if you choose MenuCascade in the ObjectType drop-down list.) If you want the button to be a drop-down toolbar button, select the check box.

Menu items can have drop-down toolbars

A menu item can have a toolbar button associated with it that displays a drop-down toolbar. When the user clicks on the button, PowerBuilder displays a drop-down toolbar that shows all of the toolbar buttons for menu items defined at the next level. For example, if you define a drop-down toolbar for the File menu item, the drop-down toolbar will show the buttons defined for the items on the File menu (such as New, Open, Close, and Exit).

PowerBuilder displays a drop-down toolbar at runtime by default if the Object Type of the menu item is MenuCascade. You can specify programmatically whether submenu items display in a drop-down toolbar or as normal toolbar items by setting the DropDown property of the menu item. For example, if you want a descendent menu item to have a drop-down toolbar, but not its ancestor, clear the DropDown check box on the ancestor's Toolbar property page, and set the DropDown property of the descendent menu item to TRUE in a script.

❖ **To specify a drop-down toolbar for a menu item:**

1   In the Menu painter, select the menu item for which you want to add a drop-down toolbar.

2   On the Toolbar property page, change the Object Type to MenuCascade.

3   (Optional) Specify the number of columns you want to display in the Columns box.

The default is a single column. If there are many items on the submenu, as there are on the toolbar item for inserting controls in the Window painter, you will probably want to specify three or four columns.

Setting toolbar properties in the Window painter

In the Window painter, you associate the menu with the window on the window's General property page. The window displays the toolbar by default. If you do not want the toolbar to display, clear the ToolbarVisible check box on the window's Toolbar property page. You can also specify the toolbar's alignment and position on this property page.

Setting toolbar properties in the Application painter

You can specify global properties for all toolbars in your application on the Toolbar property page in the Application painter or by setting properties of the Application object in a script. Typically you set these in the application's Open event, but you can set them anywhere.

Table 14-6: Toolbar properties in the Application painter

Property	Meaning
ToolbarFrameTitle	The text that displays as the title for the FrameBar when it is floating.
ToolbarSheetTitle	The text that displays as the title for the SheetBar when it is floating.
ToolbarPopMenuText	(String) Text to display on the pop-up menu for toolbars (see below).
ToolbarUserControl	(Boolean) If TRUE (default), users can use the toolbar pop-up menu to hide or show the toolbars, move toolbars, or show text. If FALSE, users cannot manipulate the toolbar.
ToolbarText	(Boolean) If TRUE, text displays in the buttons. If FALSE (the default), text does not display.
ToolbarTips	(Boolean) If TRUE (default), PowerTips display when text is not displayed in the buttons. If FALSE, PowerTips do not display.

Specifying the text in the toolbar's pop-up menu

By default, PowerBuilder provides a pop-up menu for the toolbar, which users can use to manipulate the toolbar. It is similar to the pop-up menu you use to manipulate the PowerBar and PainterBar.

You can change the text that displays in this menu, but you cannot change the functionality of the menu items in the menu. Typically, you do this when you are building an application in a language other than English.

You change the text as follows:

- The first two items in the pop-up menu display the titles set in ToolbarFrameTitle and ToolbarSheetTitle (defaults: FrameBar and SheetBar).

- The remaining text items are specified by the property ToolbarPopMenuText. To specify values for this property, use a comma-delimited list of values to replace the text "Left," "Top," "Right," "Bottom," "Floating," "Show Text," and "Show PowerTips":

      ToolbarPopMenuText = "*left, top, right, bottom, floating, showText, showPowerTips*"

For example, to change the text for the toolbar pop-up menu to German and have hot keys underlined for each, you would specify the following:

```
ToolbarPopMenuText = "&Links, &Oben, &Rechts, " +   &
   "&Unten, &Frei positionierbar, &Text anzeigen, " &
   + "&PowerTips anzeigen"
```

Writing scripts for menu items

You write scripts for menu items in the Script view. The scripts specify what happens when users select a menu item.

❖ **To write a script for a menu item:**

- Double-click the menu item or select Script from the menu item's pop-up menu.

  The Script view displays for the clicked event, which is the default event for a menu item.

Menu item events

Menu items have the following events:

- **Clicked**   Typically, your application will contain Clicked scripts for each menu item in a drop-down or cascading menu. For example, the script for the Clicked event for the Open menu item on the File menu opens a file.

- **Help**   You can provide Help on a menu item when a user presses the F1 key, or when the user clicks the context Help button [?] on the title bar of the window with which the menu is associated, and then clicks on a menu item.

- **Selected**   You will probably use few Selected scripts since users do not expect things to happen when they simply highlight a menu item. One use of Selected scripts is to change MicroHelp displayed in an MDI application as the user scrolls through a menu.

About the Clicked event

The Clicked event is triggered whenever:

- The user clicks the menu item

- The user selects (highlights) the menu item using the keyboard and then presses ENTER

- The user presses the shortcut key for the menu item

- The menu containing the menu item is displayed and the user presses the accelerator key Alt+*key*.

- When a script executes the PopMenu function and displays a pop-up menu

A menu item responds to a mouse-click or the keyboard only if both its Visible and Enabled properties are set to "true".

If the menu item has a drop-down or cascading menu under it, the script for its Clicked event (if any) is executed when the mouse button is pressed, and then the drop-down or cascading menu displays. If the menu item does not have a menu under it, the script for the Clicked event is executed when the mouse button is released.

Using the Clicked event to specify menu item properties

When the user clicks an item on the menu bar to display a drop-down menu, the Clicked event for the menu item on the menu bar is triggered and then the drop-down menu is displayed.

You can use the menu bar's Clicked event to specify the properties of the menu items in the drop-down menu. For example, if you want to disable items in a drop-down menu, you can disable them in the script for the Clicked event for the menu item in the menu bar.

About the Help event

The Help event is triggered when the user presses F1or clicks the context Help button [?] on a window's title bar and then points and clicks on a menu item.

About the Selected event

The Selected event is triggered when the user selects a menu item.

Using functions and variables

You can use functions and variables in your scripts.

Using functions

PowerBuilder provides built-in functions that act on menu items. You can use these functions in scripts to manipulate menu items at runtime. For example, to hide a menu, you can use the built-in Hide function.

For a complete list of the menu-level built-in functions, look at the Function List view or use the Browser.

Defining menu-level functions
You can define your own menu-level functions to make it easier to manipulate your menus. One way you can do this is in the Function List view, by selecting Add from the pop-up menu.

For more information, see Chapter 8, "Working with User-Defined Functions."

Using variables

Scripts for menu items have access to all global variables declared for the application. You can also declare local variables, which are accessible only in the script where they are declared.

You can declare instance variables for the menu when you have data that needs to be accessible to scripts in several menu items in a menu. Instance variables are accessible to all menu items in the menu.

For a complete description of variables and how to declare them, see the *PowerScript Reference*.

Defining menu-level structures

If you need to manipulate a collection of related variables, you can define menu-level structures using the Structure view. You do this by displaying the Structure List view and then selecting Add from the pop-up menu. The Structure and Structure List views are not part of the default layout.

For more information, see Chapter 10, "Working with Structures."

Referring to objects in your application

You can refer to any object in the application in scripts for menu items. You must fully qualify the reference, using the object name, as follows.

Referring to windows

When referring to a window, you simply name the window. When referring to a property in a window, you must always qualify the property with the window's name:

> *window.property*

For example, this statement moves the window w_cust from within a menu item script:

```
w_cust.Move(300, 300)
```

This statement minimizes w_cust:

```
w_cust.WindowState = Minimized!
```

You can use the reserved word ParentWindow to refer to the window that the menu is associated with at runtime. For example, the following statement closes the window the menu is associated with:

```
Close(ParentWindow)
```

You can also use ParentWindow to refer to properties of the window a menu is associated with, but not to refer to properties of controls or user objects in the window.

For example, the following statement is valid, because it refers to properties of the window itself:

```
ParentWindow.Height = ParentWindow.Height/2
```

But the following statement is invalid, because it refers to a control in the window:

```
ParentWindow.sle_result.Text = "Statement invalid"
```

Referring to controls and user objects in windows

When referring to a control or user object, you must always qualify the control or user object with the name of the window:

> *window.control.property*

> *window.userobject.property*

For example, this statement enables a CommandButton in window w_cust from a menu item script:

```
w_cust.cb_print.Enabled = TRUE
```

Referring to menu
items

When referring to a menu item, use this syntax:

menu.menu item

menu.menu item.property

Reference within the same menu
When referring to a menu item within the same menu, you do not have to
qualify the reference with the menu name.

When referring to a menu item in a drop-down or cascading menu, you must
specify each menu item on the path to the menu item you are referencing,
separating the names with periods.

For example, to place a check mark next to the menu item m_bold, which is on
a drop-down menu under m_text in the menu saved in the library as m_menu,
use this statement:

```
m_menu.m_text.m_bold.Check( )
```

If the previous script is for a menu item in the same menu (m_menu), you do
not need to qualify the menu item with the name of the menu:

```
m_text.m_bold.Check( )
```

Using inheritance to build a menu

When you build a menu that inherits its style, events, functions, structures,
variables, and scripts from an existing menu, you save coding time. All you
have to do is modify the descendent object to meet the requirements of the
current situation.

❖ **To use inheritance to build a descendent menu:**

1   Click the Inherit button on the PowerBar.

2   In the Inherit From Object dialog box, select Menus from the Object Type
    drop-down list, the library or libraries you want to look in, and the menu
    you want to use to create the descendant, and click OK.

Displaying menus from many libraries
To find a menu more easily, you can select more than one library in the
Application Libraries list. Use Ctrl+Click to toggle selected libraries and
Shift+Click to select a range.

The selected menu displays in the WYSIWYG Menu view and the Tree Menu view in the Menu painter. The title in the painter's title bar indicates that the menu is a descendant.

3   Make the changes you want to the descendent menu as described in the next section.

4   Save the menu under a new name.

Using the inherited information

When you build and save a menu, PowerBuilder treats the menu as a unit that includes:

•   All menu items and their scripts

•   Any variables, functions, and structures declared for the menu

When you use inheritance to build a menu, everything in the ancestor menu is inherited in all of its descendants.

What you can do   You can do the following in a descendent menu:

•   Add menu items to the end of a menu

•   Insert menu items in a menu (with some restrictions)

    For more information, see "Where you can insert menu items in a descendent menu" on page 353.

•   Modify existing menu items

    For example, you can change the text displayed for a menu item or change its initial appearance, such as making it disabled or invisible.

•   Build scripts for menu items that do not have scripts in the ancestor menu

•   Extend or override inherited scripts

•   Declare functions, structures, and variables for the menu

What you cannot do   You cannot do the following in a descendent menu:

•   Change the order of inherited menu items

•   Delete an inherited menu item

•   Insert menu items between inherited menu items that do not have the ShiftToRight property set (see "Modifying the ShiftToRight property" on page 352)

- Change the name of an inherited menu item

- Change the type of an inherited menu item

Hiding a menu item
If you do not need a menu item in a descendent menu, you can hide it by clearing the visible property in the Properties view or by using the Hide function.

About menu item names in a descendant

PowerBuilder uses the following syntax to show names of inherited menu items:

AncestorMenuName::MenuItemName

For example, in a menu inherited from m_update_file, you see m_update_file::m_file for the m_file menu item, which is defined in m_update_file.

The inherited menu item name is also locked, so you cannot change it.

Understanding inheritance

The issues concerning inheritance with menus are similar to the issues concerning inheritance with windows and user objects. For information, see Chapter 13, "Understanding Inheritance."

Inserting menu items in a descendent menu

Modifying the ShiftToRight property

When defining a descendent menu, you might want to insert menu items in the middle of the menu bar or in the middle of a drop-down or cascading menu. To do this, you set the ShiftToRight property in a menu item's Properties view on the General property page.

If the ancestor menu has no menu items with ShiftToRight set, you can add a new menu item to the end of the descendent menu. To add new menu items elsewhere in the menu, set the ShiftToRight property for the descendent menu items that will follow the new menu item.

The ShiftToRight property is used for menu items on the menu bar (where items need to shift right if a new item is inserted) and for menu items in a drop-down or cascading menu (where items might need to shift *down* if a new item is inserted). The property name is ShiftToRight, but it means shift down in drop-down or cascading menus.

Where you set the
ShiftToRight property

You set the ShiftToRight property in an ancestor menu only if you know that you will always want a group of menu items to shift right (or down) when you inherit from the menu and add a new menu item. For example, if you have File, Edit, Window, and Help menus on the menu bar, set the ShiftToRight property for the Window and Help menu items if you are going to inherit from this menu, because Window and Help are usually the last items on a menu bar.

Where you can insert
menu items in a
descendent menu

In a descendent menu, a group of menu items can be one of four types. Each type has an insertion rule.

Table 14-7: Insertion rules for groups of menu items

Type of group	Insertion rule
Inherited menu items without ShiftToRight set	You cannot insert a new menu item before any of these menu items
Inherited menu items with ShiftToRight set in ancestor	You can insert before the first menu item in the group but not before the others
New items without ShiftToRight set	You can insert a new menu item before any of these menu items
New items with ShiftToRight set	You can insert a new menu item before any of these menu items

The "Example with no ShiftToRight in ancestor" on page 354 and the "Example with ShiftToRight in ancestor" on page 355 demonstrate some of these rules.

How to insert menu
items in a descendent
menu

If you can insert a menu item in a descendent menu, the Insert Menu Item option on the Insert menu and the pop-up menu is enabled. The Insert Menu Item is enabled if ShiftToRight is set in the selected item that will follow the item you are inserting and all menu items following it.

To insert a menu item in a descendant, you use the same method you use to insert an item in a new menu, whether the menu item is on the menu bar or on a drop-down or cascading menu. For information about inserting menu items, see "Working with menu items" on page 324.

The following examples illustrate where you can insert menu items in a descendent menu and demonstrate the rules that govern where you can insert them.

Example with no
ShiftToRight in
ancestor

Suppose you have a menu with File, Edit, Window, and Help items on the menu bar. The menu is inherited from an ancestor frame menu with no items set as ShiftToRight in the ancestor.
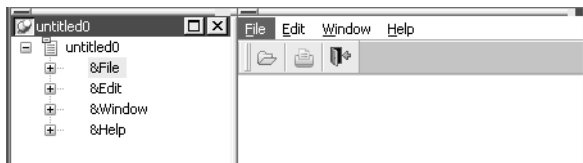
Here is how you might add some new menu items. Since ShiftToRight is not set anywhere at first, you can add a menu item only to the end.

1   Select any item in the menu bar and select Insert>Menu Item At End.

2   Name the new menu item New1 and press Enter.

The New1 menu item is added to the right of the Help menu.

Now add a new Menu item before the New1 menu item. You can do this without setting ShiftToRight on New1, because New1 is a new menu item in the inherited menu.

3   Select Insert Menu Item from the pop-up menu for New1.

4   Name the new menu item New2 and press Enter.

Now add a new Menu item before the Help menu item. You cannot do this unless you set ShiftToRight on the Help menu item, the New2 Menu item, and the New1 menu item, because Help is an inherited menu item without ShiftToRight set in the ancestor menu. For Help to shift right, New2 and New1 must also be able to shift right.

5   Select the Help menu item and in the Properties view, select the ShiftToRight property, and then do the same for New1 and New2.

Order for setting ShiftToRight for the three menu items
You can set ShiftToRight in any order, but you see the items shifting only if you set ShiftToRight from left to right.

Now you can add a new menu item before the Help menu item.

6   Select the Help menu item, then select Insert New Item from the pop-up
menu, name the new item New3, and then press Enter.

File   Edit   Window   New3   Help   New2   New1

If you want to add a new Menu item before the New3 menu item, you can
do it without setting ShiftToRight on New3, because New3 is a new menu
item and ShiftToRight is set in all items that follow.

However, if you want to add a new menu item before the Window menu
item, you cannot do this by working only in the descendent menu because
the Window menu item is an ancestor menu item and ShiftToRight is not
set in the ancestor. To be able to do this, you must set Window as
ShiftToRight in the ancestor.

**Example with
ShiftToRight in
ancestor**

In this example, the inherited menu has the same four menu bar items, but
ShiftToRight has been set in the Window and Help menu items in the ancestor
menu. Suppose you want to insert a new menu item before the Help menu item
and the Window menu item.

1   Select the Help menu item and display the pop-up menu.

The Insert Menu Item option is disabled because the Help item *is not* the
first item in a group of ancestor menu items (Window and Help) with
ShiftToRight set in the ancestor.

2   Select the Window menu item and display the pop-up menu.

The Insert Menu Item option is enabled because the Window item *is* the
first item in a group of ancestor menu items with ShiftToRight set in the
ancestor.

3   Select Insert Menu Item At End from the pop-up menu to insert a new
menu item after Help, name it New1, and press Enter.

The New1 item's ShiftToRight property is set automatically.

Now the Window, Help, and New1 items are set ShiftToRight. You can
insert a new item before Window and New1, but not before Help. This is
because the Window and Help menu items are a group for which
ShiftToRight is set in the ancestor.

You cannot insert a new item before the Edit menu item because Edit is in
a group (File and Edit) that are inherited items with no ShiftToRight set in
the ancestor.

4   Select the Edit menu item, select ShiftToRight in the Properties view, and then add a new menu item.

File **New2** Edit Window Help New1

You could also have set the ShiftToRight property in the ancestor menu, but it is easier to work just in the descendant.

Using menus in your applications

You can use menus in two ways:

- Place them in the menu bar of a window

- Display a menu as a pop-up menu

Adding a menu bar to a window

To have a menu bar display when a window is opened by a user, you associate a menu with the window in the Window painter.

❖   **To associate a menu with a window:**

1   Click the Open button in the PowerBar, select the window with which you want to associate the menu, and open the window.

2   Do one of the following:

- In the Properties view for the window, enter the name of the menu in the MenuName text box on the General tab page.

- Click the Browse button and select the menu from the Select Object dialog box, which lists all menus available to the application.

In the Select Object dialog box, you can search for a menu by clicking the Browse button.

3   Click Save to associate the selected menu with the window.

| Identifying menu items in window scripts | You reference menu items in scripts in windows and controls using the following syntax: |

menu.menu item

You must always fully qualify the menu item with the name of the menu.

When referring to a menu item in a drop-down or cascading menu, you must specify each menu item on the path to the menu item you are referencing, separating the names with periods.

For example, to refer to the Enabled property of menu item m_open, which is under the menu bar item m_file in the menu saved in the library as m_menu, use:

```
m_menu.m_file.m_open.Enabled
```

| Changing a window's menu at runtime | You can use the ChangeMenu function in a script to change the menu associated with a window at runtime. |

Displaying pop-up menus

To display a pop-up menu in a window, use the PopMenu function to identify the menu and the location at which you want to display the menu.

| If the menu is associated with the window | If the menu is currently associated with the window, you can simply call the PopMenu function. |

The following statement in a CommandButton script displays m_appl.m_help as a pop-up menu at the current pointer position, assuming menu m_appl is already associated with the window:

```
m_appl.m_help.PopMenu(PointerX(), PointerY())
```

| If the menu is not associated with the window | If the menu is not already associated with the window, you must create an instance of the menu before you can display it as a pop-up menu. |

The following statements create an instance of the menu m_new, then pop up the menu mymenu.m_file at the pointer location, assuming m_new is not associated with the window containing the script:

```
m_new    mymenu
mymenu = create m_new
mymenu.m_file.PopMenu(PointerX(), PointerY())
```

Working with User Objects

About this chapter

One of the features of object-oriented programming is reusability: you define a component once, then reuse it as many times as you need to without any additional work. User objects are one of the best ways to take advantage of reusability in PowerBuilder. This chapter describes how to define and use user objects.

Contents

About user objects

Applications often have features in common. For example, you might often reuse features like the following:

- A processing package that calculates commissions or performs statistical analysis

- A Close button that performs a certain set of operations and then closes the window

- DataWindow controls that perform standard error checking

- A list that includes all departments

- A predefined file viewer that you plug into a window

If you find yourself using the same application feature repeatedly, you should define a user object: you define the user object once in the User Object painter and use it as many times as you need.

There are two main types of user objects: class and visual. Class user objects are also called nonvisual objects.

Examples of user objects
The PowerBuilder Code Examples contain many interesting user objects in *PBEXAMUO.PBL*. Take a look at them to get an appreciation for the power of user objects.

Class user objects

A class user object lets you reuse a set of business rules or other processing that acts as a unit but has no visual component. For example, you might define a class that calculates sales commissions or performs statistical analysis. Whenever you need to do this type of processing, you instantiate the user object in a script and call its functions.

You build class user objects in the User Object painter, specifying instance variables and object-level functions. Then you create an instance of the class user object in your application, thereby making its processing available.

There are two kinds of class user objects:

• Custom class

• Standard class

Custom class user objects

Custom class user objects are objects of your own design that encapsulate properties and functions not visible to the user. They are not derived from PowerBuilder objects. You define them to create units of processing that have no visual component.

For example, to calculate commissions in an application, you can define an n_CalculateCommission custom class user object that contains properties and user-defined functions that do the processing to calculate commissions.

Whenever you need to use this processing, you create an instance of the user object in a script, which then has access to the logic in the user object.

When you build components that you will deploy to a transaction server, you use custom class user objects. For more information, see *Application Techniques*.

Standard class user objects

A standard class user object inherits its definition from one built-in, nonvisual PowerBuilder object, such as the Transaction object or Error object. You modify the definition to make the object specific to your application, and optionally add instance variables and functions to enhance the behavior of the built-in object. Once you define a standard class user object, you can go to the Application painter and specify that you want to use it instead of the corresponding built-in system object in your application.

One important use of a standard class user object is employing one inherited from the built-in Transaction object to do database remote procedure calls from within an application.

Visual user objects

A visual user object is a reusable control or set of controls that has a certain behavior. You define it in the User Object painter, where you place controls in the user object and write scripts for those controls. Then you can place the user object in windows you build in your applications as often as needed.

There are three types of visual user objects:

- **Custom visual**   Most useful if you frequently group controls together in a window and always use the controls to perform the same processing.

- **External visual**   Useful when you have a custom DLL.

- **Standard visual**   Most useful if you frequently use a PowerBuilder control to perform the same processing.

Custom visual user objects

Custom visual user objects are objects that have several controls that function as a unit. You can think of a custom visual user object as a window that is a single unit and is used as a control.

Assume you frequently use a group of buttons, each of which performs standard processing. If you build a custom user object that contains all the buttons, you can place the buttons in the window as a unit when you place the user object in a window.

External visual user objects

External visual user objects contain controls from objects in the underlying windowing system that were created outside PowerBuilder. You can use a custom DLL in PowerBuilder to create an external user object.

You must know what classes the DLL supports, the messages or events the DLL responds to, and the style bits that you can set in the DLL.

Standard visual user objects

A standard visual user object inherits its definition from one standard PowerBuilder control. You modify the definition to make the control specific to your applications.

Assume you frequently use a CommandButton named Close to display a message box and then close the parent window. If you build a standard visual user object that derives from a CommandButton to perform this processing, you can use the user object whenever you want to display a message box and then close a window.

Building user objects

You can build a user object from scratch, or you can create a user object that inherits its style, events, functions, structures, variables, and scripts from an existing user object.

For information on building a user object from scratch, see "Building a new user object" on page 364. To find out more about creating a user object based on an existing PowerBuilder object, see "Using inheritance to build user objects" on page 372.

About the User Object painter

The User Object painter has five implementations, depending on the type of user object you are working with. It has several views where you specify how the user object behaves and, for custom visual and standard visual user objects, how it looks. For details about the views, how you use them, and how they are related, see "Views in painters that edit objects" on page 116.

Views for visual user objects

In this User Object painter for a custom visual user object, the Layout view and Script view have been arranged to display at the same time:
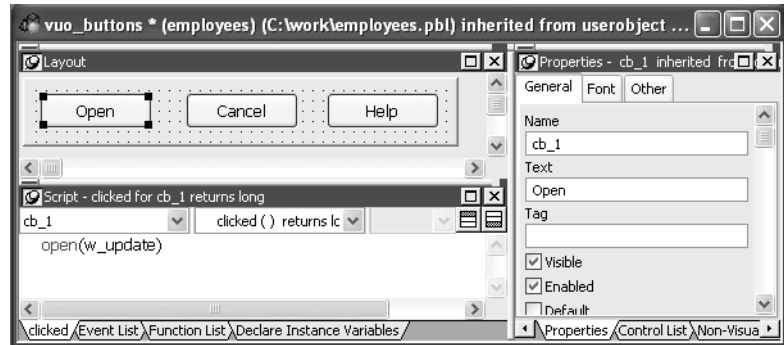
Most of your work in the User Object painter for visual objects is done in three views:

- The Layout view, where you design the appearance of the user object

- The Properties view, where you set user object properties and control properties

- The Script view, where you modify behavior by coding user object and control scripts

In the Layout view, you add controls to a visual user object in the same way you add controls to a window.

For information about specifying user object properties, see "Building a new user object" on page 364. For information about using the Script view, see Chapter 7, "Writing Scripts."

Views for nonvisual user objects

You do not need the Layout and Control List views for nonvisual user objects, but otherwise, you use all the views that you use for visual objects.

Nonvisual user objects require no layout design work, but working in the User Object painter on the behavior of a nonvisual object is otherwise similar to working on the behavior of a visual user object.

Building a new user object

This section describes how to build a user object from scratch. You use this technique to create user objects that are not based on existing user objects.

Creating a new user object

❖ **To create a new user object:**

1   Open the New dialog box.

2   On PB Object tab page, select the kind of user object you want to create.

The five user object choices display at the top of the tab page:
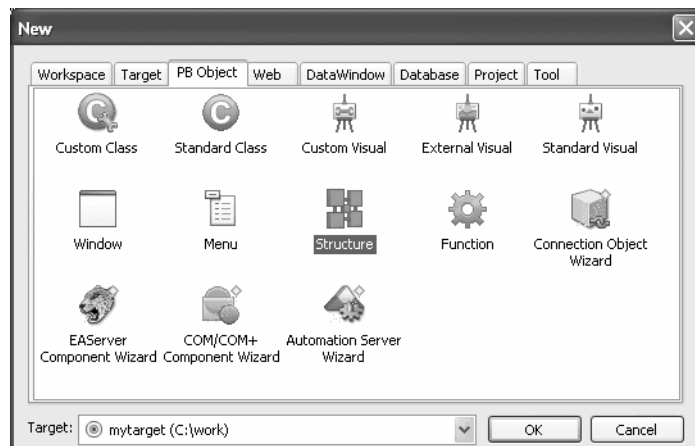
3   Click OK.

What you do next depends on the type of user object you selected. For all user objects except Standard Class and Standard Visual, the User Object painter opens.

The remainder of this section describes how to build each type of user object.

Building a custom class user object

On the PB Object tab page of the New dialog box, if you select Custom Class and click OK, the User Object painter for custom class user objects opens.

❖ **To build the custom class user object:**

1   Declare functions, structures, or variables you need for the user object.

2   Create and compile scripts for the user object.

Custom class user objects have built-in constructor and destructor events.

3   Save the user object.

See "Saving a user object" on page 370.

Using AutoInstantiate

You can create custom class user objects that are autoinstantiated, which provides you with the ability to define methods.

Autoinstantiated user objects do not require explicit CREATE or DESTROY statements when you use them. They are instantiated when you call them in a script and destroyed automatically.

❖ **To define an autoinstantiated custom class user object:**

•   In the Properties view, select the AutoInstantiate check box.

For more information about autoinstantiation, see the *PowerScript Reference.*

Using the EAServer
Project property

In the Properties view, if you specify an EAServer project that will be used to generate an EAServer component (a custom class user object with the characteristics needed for deployment to EAServer), live editing is enabled. When live editing is enabled in the User Object painter, PowerBuilder builds the project for an EAServer component each time you save the user object.

For information about EAServer components and live editing, see *Application Techniques*.

Building a standard class user object

On the PB Object tab page of the New dialog box, if you select Standard Class and click OK, the Select Standard Class Type dialog box displays.

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

1 In the Select Standard Class Type dialog box, select the built-in system object that you want your user object to inherit from and click OK.

2 Declare functions, structures, or variables you need for the user object.

For a list of properties and functions
Use the Browser to list the built-in properties inherited from the selected system object. Use the Function List view or the Browser to list the functions inherited from the selected system object.

---

3 Declare any user events needed for the user object.

For information about user events, see "Communicating between a window and a user object" on page 379.

4 In the Script view, create and compile scripts for the user object.

Class user objects have built-in constructor and destructor events.

5 Save the user object.

See "Saving a user object" on page 370.

Building a custom visual user object

On the PB Object tab page of the New dialog box, if you select Custom Visual and click OK, the User Object painter for custom visual user objects opens. It looks like the Window painter, but the empty box that displays in the Layout view is the new custom visual user object.

Building a custom visual user object is similar to building a window, described in Chapter 11, "Working with Windows." The views available in the Window painter and the User Object painter for custom visual user objects are the same.

❖ **To build the custom visual user object:**

1 Place the controls you want in the custom visual user object.

2 Work with the custom visual user object as you would with a window in the Window painter:

   • Define the properties of the controls

   • Declare functions, structures, or variables as necessary

•   Declare any events needed for the user object or its controls

For information about user events, see "Communicating between a window and a user object" on page 379.

•   In the Script view, create and compile the scripts for the user object or its controls

You can write scripts for each control in a custom visual user object.

For more information on events associated with custom visual user objects, see "Events in user objects" on page 369.

3   Save the user object.

See "Saving a user object" on page 370.

Building an external visual user object

On the PB Object tab page of the New dialog box, if you select External Visual and click OK, the User Object painter for external visual user objects opens.

❖   **To build an external visual user object:**

1   In the Properties view, click the Browse button next to the LibraryName box.

2   In the Select Custom Control DLL dialog box, select the DLL that defines the user object and click OK.

3   In the Properties view, enter the following information, as necessary, and click OK:

•   The class name registered in the DLL

Information about the class name is usually provided by the vendor of the purchased DLL.

•   Text in the Text box

This will be displayed only if the object has a text style property.

•   Display properties (border and scrollbars)

•   Decimal values for the style bits associated with the class

Information about style bits is usually provided by the vendor of the purchased DLL. PowerBuilder will OR these values with the values selected in the display properties for the control.

4   Declare any functions, structures, or variables you need to declare for the user object.

You can declare functions, structures, and variables for the user object in the Script view. Information about functions is usually provided by the vendor of the purchased DLL.

5   Declare any needed events for the user object.

For information about user events, see "Communicating between a window and a user object" on page 379.

6   In the Script view, create and compile the scripts for the user object.

For more information on events associated with external visual user objects, see "Events in user objects" on page 369.

7   Save the user object.

See "Saving a user object" on page 370.

Building a standard visual user object

On the PB Object tab page of the New dialog box, if you select Standard Visual and click OK, the Select Standard Visual Type dialog box displays.

❖ **To build a standard visual user object:**

1   In the Select Standard Visual Type dialog box, select the PowerBuilder control you want to use to build your standard visual user object and click OK.

The selected control displays in the workspace. Your visual user object will have the properties and events associated with the PowerBuilder control you are modifying.

2   Work with the control as you do in the Window painter:

•   Review the default properties and make any necessary changes

•   Declare functions, structures, or variables as necessary

You can declare these in the Script view.

- Declare any user events needed for the user object

    For information about user events, see "Communicating between a window and a user object" on page 379.

- Create and compile the scripts for the user object

    Standard visual user objects have the same events as the PowerBuilder control you modified to create the object.

3   Save the user object.

    See "Saving a user object" on page 370.

Events in user objects

When you build a user object, you can write scripts for any event associated with that user object.

Events in class user objects

Most custom class user objects have only constructor and destructor events. Activate and deactivate events are created for the EAServer and COM+ custom class user objects that you create using the wizards at the bottom of the PB Object page in the New dialog box. For more information, see *Application Techniques*.

Table 15-1: Events for custom class user objects

Event	Occurs when
Constructor	The user object is created
Destructor	The user object is destroyed
Activate	The user object is activated in an application server (EAServer and COM+ component wizards only)
Deactivate	The user object is deactivated in an application server (EAServer and COM+ component wizards only)

Standard class user objects have the same events as the PowerBuilder system object from which they inherit.

Events in visual user objects

Standard visual user objects have the same events as the PowerBuilder control from which they inherit. Custom and external visual user objects have a common set of events.

Table 15-2: Events for custom and external visual user objects

Event	Occurs when
Constructor	Immediately before the Open event of the window and when the user object is dynamically placed in a window
Destructor	Immediately after the Close event of the window and when the user object is dynamically removed from a window
DragDrop	A dragged object is dropped on the user object
DragEnter	A dragged object enters the user object
DragLeave	A dragged object leaves the user object
DragWithin	A dragged object is moved within the user object
Help	A user presses the F1 key or clicks the context Help button [?] on the title bar of the window with which the menu is associated and then points and clicks on a menu item
Other	A Windows message occurs that is not a PowerBuilder event
RButtonDown	The right mouse button is pressed

For more about drag and drop, see *Application Techniques*.

Saving a user object

❖ **To save a user object:**

1   In the User Object painter, select File>Save from the menu bar or click the Save button in the painter bar.

If you have previously saved the user object, PowerBuilder saves the new version in the same library and returns you to the User Object painter.

If you have not previously saved the user object, PowerBuilder displays the Save User Object dialog box.

2   Enter a name in the User Objects box.

For naming considerations, see "Naming the user object" next.

3   Enter comments to describe the user object.

These display in the Select User Object dialog box and in the Library painter, and will document the purpose of the user object.

4   Specify the library in which to save the user object.

To make a user object available to all applications, save it in a common library and include the library in the library search path for each application.

5   Click OK to save the user object.

Validation for server components
In the User Object painter for a custom class user object, the Design menu has EAServer Validation and COM/COM+ Validation menu items. If you are saving an EAServer component or a COM/COM+ component and you check the validation menu item for the component to enable validation, a check displays next to the menu item. When you save the object, you might see some error messages.

For information about EAServer validation and COM/COM+ validation, see *Application Techniques*.

Naming the user object

A user object name can be any valid PowerBuilder identifier up to 40 characters. For information about PowerBuilder identifiers, see the *PowerScript Reference*.

Naming conventions   You should adopt naming conventions to make it easy to understand a user object's type and purpose.

One convention you could follow is to use u_ as the prefix for visual user objects and n_ as the prefix for class (nonvisual) user objects. For standard classes, include the standard prefix for the object or control from which the class inherits in the name. For external user objects, include ex_ in the name, and for custom class user objects, include cst_ in the name.

Table 15-3 shows some examples of this convention.

Table 15-3: Suggested naming conventions for user objects

Type of user object	Format	Example
Standard visual	u_*control_purpose*	u_cb_close, a CommandButton that closes a window
Custom visual	u_*purpose*	u_toolbar, a toolbar
External visual	u_ex_*purpose*	u_ex_sound, outputs sound
Standard class	n_*systemobject_purpose*	n_trans_test, derived from the Transaction object and used for testing
Custom class	n_cst_*purpose*	n_cst_commission, calculates commissions

For a list of naming conventions, see "Naming conventions" in Chapter 5, "Working with PowerScript Targets."

Using inheritance to build user objects

When you build a user object that inherits its definition (properties, events, functions, structures, variables, controls, and scripts) from an existing user object, you save coding time. All you must do is modify the inherited definition to meet the requirements of the current application.

For example, suppose your application has a user object u_file_view that has three CommandButtons:

- List—displays a list of files in a list

- Open—opens the selected file and displays the file in a MultiLineEdit control

- Close—displays a message box and then closes the window

If you want to build another user object that is exactly like the existing u_file_view except that it has a fourth CommandButton, you can use inheritance to build the new user object, and then all you need to do is add the fourth CommandButton.

❖ **To use inheritance to build a descendent user object:**

1   Click the Inherit button in the PowerBar, or select File>Inherit from the menu bar.

2   In the Inherit From Object dialog box, select User Objects from the Objects of Type drop-down list.

3   Select the target as well as the library or libraries you want to look in.

Displaying user objects from many libraries
To find a user object more easily, you can select more than one library in the Libraries list. Use Ctrl+Click to toggle selected libraries and Shift+Click to select a range.

---

4   Select the user object you want to use to create the descendant, and click OK.

The selected object displays in the User Object painter and the title bar indicates that the object is a descendant.

5   Make any changes you want to the user object.

6   Save the user object with a new name.

Using the inherited information

When you build and save a user object, PowerBuilder treats the object as a unit that includes:

•   The object (and any controls within the object if it is a custom visual user object)

•   The object's properties, events, and scripts

•   Any variables, functions, or structures declared for the object

When you use inheritance to build a new user object, everything in the ancestor user object is inherited in the direct descendant and in its descendants in turn.

Ancestor's instance variables display

If you create a user object by inheriting it from a custom class or standard class user object that has public or protected instance variables with simple datatypes, the instance variables display and can be modified in the descendent user object's Properties view.

All public instance variables with simple datatypes such as integer, boolean, character, date, string, and so on display in the descendant. Instance variables with the any or blob datatype or instance variables that are objects or arrays do not display.

What you can do in the descendant

You can do the following in a descendent user object:

- Change the values of the properties and the variables

- Build scripts for events that do not have scripts in the ancestor

- Extend or override the inherited scripts

- Add controls (in custom visual user objects)

- Reference the ancestor's functions and events

- Reference the ancestor's structures if the ancestor contains a public or protected instance variable of the structure datatype

- Access ancestor properties, such as instance variables, if the scope of the property is public or protected

- Declare variables, events, functions, and structures for the descendant

What you cannot do in the descendant

In a descendent user object, you cannot delete controls inherited from a custom visual user object. If you do not need a control in a descendent user object, you can make it invisible.

Understanding inheritance

The issues concerning inheritance with user objects are the same as the issues concerning inheritance with windows and menus. See Chapter 13, "Understanding Inheritance," for more information.

Using user objects

Once you have built a user object, you are ready to use it in an application. This section describes how to use:

- Visual user objects

- Class user objects

Using visual user objects

You use visual user objects by placing them in a window or in a custom visual user object. The techniques are similar whether you are working in the Window painter or the User Object painter.

❖ **To place a user object:**

1 Open the window or custom visual user object in which you want to place the visual user object.

2 Click the User Object button in the PainterBar, or select Insert>Control from the menu bar and then select User Object.

3 Select the user object you want to use and click the location where you want the user object to display.

PowerBuilder creates a descendent user object that inherits its definition from the selected user object and places it in the window or user object.

Dragging the user object from the System Tree
You can drag a user object from the System Tree to the Layout view in the Window painter.

What you can do

After you place a user object in a window or a custom visual user object, you can name it, size it, position it, write scripts for it, and do anything else you can do with a control.

When you place the user object in a window, PowerBuilder assigns it a unique name, just as it does when you place a control. The name is a concatenation of the default prefix for a user object control (initially, uo_) and a default suffix, which is a number that makes the name unique.

You should change the default suffix to a suffix that has meaning for the user object in your application.

For more information about naming, see "Naming controls" on page 254.

Writing scripts

When you place a user object in a window or a custom user object, you are actually creating a descendant of the user object. All scripts defined for the ancestor user object are inherited. You can choose to override or extend those scripts.

For more information, see "Using inherited scripts" on page 311.

You place a user object *as a unit* in a window (or another user object). You cannot write scripts for individual controls in a custom user object after placing it in a window or custom user object; you do that only when you are defining the user object itself.

Placing a user object at runtime

You can add a user object to a window at runtime using the PowerScript functions OpenUserObject and OpenUserObjectWithParm in a script. You can remove a user object from a window using the CloseUserObject function.

Using class user objects

How you insert a nonvisual object

There are two ways to use a class user object when the user object is not autoinstantiating: you can create an instance of it in a script, or you can insert the user object in a window or user object using the Insert menu.

For more information on autoinstantiation, see "Using AutoInstantiate" on page 365.

The nonvisual object you insert can be a custom class user object or a standard class user object of most types.

❖ **To instantiate a class user object:**

1 In the window or user object in which you want to use the class user object, declare a variable of the user object type and create an instance of it using the CREATE statement. For example:

```
// declared instance variable:
// n_myobject invo_myobject
invo_myobject = CREATE n_myobject
```

2 Use the user object's properties and functions to do the required processing.

3 When you have finished using the user object, destroy it using the DESTROY statement.

If you select Autoinstantiate in the properties of the class user object, you cannot use the CREATE and DESTROY statements.

❖ **To insert a class user object:**

1 Open the window or user object in which you want to insert the class user object.

2 Select Insert>Object from the menu bar.

3   Select User Object (at the bottom of the list) and then select the class user object you want to insert.

PowerBuilder inserts the selected class user object.

4   Modify the properties and code the events of the nonvisual object as needed.

When the user object is created in an application, the nonvisual object it contains is created automatically. When the user object is destroyed, the nonvisual object is destroyed automatically.

Using the Non-Visual Object List view

You can use the same technique to insert standard class user objects. Since all class user objects are nonvisual, you cannot see them, but if you look at the Non-Visual Object List view, you see all the class user objects that exist in your user object.

Using the Non-Visual Object List view's pop-up menu, you can display a class user object's properties in the Properties view, display the Script view for the object to code its behavior, or delete the object.

Using global standard class user objects

Five of the standard class user object types are inherited from predefined global objects used in all PowerBuilder applications:

Transaction (SQLCA)
DynamicDescriptionArea (SQLDA)
DynamicStagingArea (SQLSA)
Error
Message

Replacing the built-in global object

If you want your standard class user object to *replace* the built-in global object, you tell PowerBuilder to use your user object *instead of* the built-in system object that it inherits from. You will probably use this technique if you have built a user object inheriting from the Error or Message object.

❖ **To replace the built-in global object with a standard class user object:**

1   Open the Application object.

2   In the Properties view, click the Additional Properties button on the General tab page.

3   In the Application properties dialog box, select the Variable Types tab.

4   Specify the standard class user object you defined in the corresponding
     field and click OK.
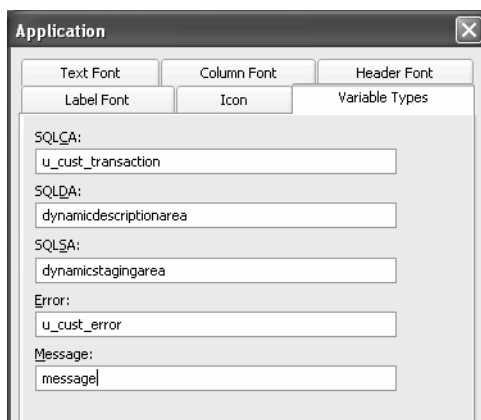
After you have specified your user object as the default global object, it
replaces the built-in object and is created automatically when the application
starts up. You do not create it (or destroy it) yourself.

The properties and functions defined in the user object are available anywhere
in the application. Reference them using dot notation, just as you access those
of other PowerBuilder objects such as windows.

Supplementing the
built-in global object

You can use a user object inherited from one of these global objects by
inserting one in your user object as described in "Using class user objects" on
page 376. If you do, your user object is used *in addition to* the built-in global
object variable. Typically you use this technique with user objects inherited
from the Transaction object. You now have access to two Transaction objects:
the built-in SQLCA and the one you defined.

For more information

For more information about using the Error object, see "Using the Error
object" on page 884.

For information about using the Message object, and about creating your own
Transaction object to support database remote procedure calls, see *Application
Techniques*.

For more information about the DynamicDescriptionArea and
DynamicStagingArea objects used in dynamic SQL, see the *PowerScript
Reference*.

Communicating between a window and a user object

Often you need to exchange information between a window and a visual user object in the window. Consider these situations:

*   You have a set of buttons in a custom user object. Each of the buttons acts upon a file that is listed in a SingleLineEdit control in the window (but not in the user object).

    You need to pass the contents of the SingleLineEdit control from the window to the user object.

*   You have a user object color toolbar. When the user clicks one of the colors in the user object, a control in the window changes to that color.

    You need to pass the color from the user object to the window control.

This section discusses two techniques for handling this communication and presents a simple example.

Table 15-4: Techniques for communicating information in a window

Technique	Advantages	Disadvantages
Functions	Easy to use	Creates overhead, might be unnecessary for simple operations
	Supports parameters and return types, so is not prone to errors	
	Supports data encapsulation and information hiding	
	Best for complex operations	
User events	Very flexible and powerful	Uses no type checking, so is prone to error

Communication with both techniques can be either synchronous (using Send for functions and the EVENT keyword for events) or asynchronous (using Post for functions and the POST keyword for events).

Directly referencing properties

Instead of using functions or user events, it is possible to reference properties of a user object directly. If you have a user object control, uo_1, associated with a custom user object that has a SingleLineEdit, sle_1, you can use the following in a script for the window:

```
uo_1.sle_1.Text = "new text"
```

However, it is better to communicate with user objects through functions and user events, as described below, in order to maintain a clean interface between your user object and the rest of your application.

The functions technique	Exchanging information using functions is straightforward. After a user object calls a function, any return value is available to any control within that object.

For how to use this technique, see "Example 1: using functions" on page 382.

❖ **To pass information from a window to a user object:**

1   Define a public, user object-level function that takes as arguments the information needed from the window.

2   Place the user object in the window.

3   When appropriate, call the function from a script in the window, passing the needed information as arguments.

❖ **To pass information from a user object to a window:**

1   Define a public, window-level function that takes as parameters the information needed from the user object.

2   Place the user object in the window.

3   When appropriate, call the function from a script in the user object, passing the needed information as parameters.

The user events technique	You can define user-defined events, also called user events, to communicate between a window and a user object. You can declare user events for any PowerBuilder object or control.

A custom visual user object often requires a user event. After you place a custom visual user object in a window or in another custom user object, you can write scripts only for events that occur in the user object itself. You cannot write scripts for events in the controls in the user object.

You can, however, define user events for the user object, and trigger those events in scripts for the controls contained in that user object. In the Window painter, you write scripts for the user events, referencing components of the window as needed.

For more information about user events, see Chapter 9, "Working with User Events," and *Application Techniques*. For instructions for using this technique, see "Example 2: using user events" on page 383.

❖ **To define and trigger a user event in a visual user object:**

1   In the User Object painter, select the user object.

Make sure no control in the user object is selected.

2   In the Event List view, select Add from the pop-up menu.

3   In the Prototype window that displays, define the user event.

For how to do so, see "Defining user events" on page 208.

4   Use the Event keyword in scripts for a control to trigger the user event in the user object:

> *userobject*.**Event** *eventname* ( )

For example, the following statement in the Clicked event of a CommandButton contained in a custom visual user object triggers the Max_requested event in the user object:

```
Parent.Event Max_requested()
```

This statement uses the pronoun Parent, referring to the custom visual user object itself, to trigger the Max_requested event in that user object.

5   Implement these user events in the Window painter.

❖   **To implement the user event in the window:**

1   Open the window.

2   In the Window painter, select Insert>Control from the menu bar and place the custom visual user object in the window.

3   Double-click the user object and then in the Script view, write scripts for the user events you defined in the User Object painter.

Examples of user object controls affecting a window

To illustrate these techniques, consider a simple custom visual user object, uo_minmax, that contains two buttons, Maximize and Minimize.

If the user clicks the Maximize button in an application window containing this user object, the current window becomes maximized. If the user clicks Minimize, the window closes to an icon.

Because the user object can be associated with any window, the scripts for the buttons cannot reference the window that has the user object. The user object must get the name of the window so that the buttons can reference the window.

"Example 1: using functions" next shows how PowerBuilder uses functions to pass a window name to a user object, allowing controls in the user object to affect the window the user object is in.

"Example 2: using user events" on page 383 shows how PowerBuilder uses unmapped user events to allow controls in a user object to affect the window the user object is in.

Example 1: using functions

1   In the Script view in the User Object painter, define an instance variable, mywin, of type window.

```
window mywin
```

This variable will hold the name of the window that has the user object.

2   Define a user object-level function, f_setwin, with:

- Public access

- No return value

- One argument, win_param, of type window and passed by value

3   Type the following script for the function:

```
mywin = win_param
```

When f_setwin is called, the window name passed in win_param will be assigned to mywin, where user object controls can reference the window that has the user object.

4   Write scripts for the two buttons:

- **cb_max:** `mywin.WindowState = Maximized!`

- **cb_min:** `mywin.WindowState = Minimized!`

5   Save the user object as uo_minmax and close the User Object painter.

6   Open the window, drag uo_minmax onto the window in the Layout view, and name it uo_func in the Properties view.

7   In the Open event for the window, call the user object-level function, passing the name of the window:

```
uo_func.f_setwin(This)
```

The pronoun This refers to the window's name, which will be passed to the user object's f_setwin function.

What happens   When the window opens, it calls the user object-level function f_setwin, which passes the window name to the user object. The user object stores the name in its instance variable mywin. When the user clicks a button control in the user object, the control references the window through mywin.

Example 2: using user events

1   In the Script view in the User Object painter, define two unmapped user events for the user object: Max_requested and Min_requested.

Leave the Event ID fields blank to define them as unmapped.

2   Trigger user events of the user object in the scripts for the Clicked event of each CommandButton:

- cb_max: `Parent.Event Max_requested()`

- cb_min: `Parent.Event Min_requested()`

3   Save the user object and name it uo_event and close the User Object painter.

4   Open the window and in the Window painter, select Insert>Object from the menu bar and then place uo_event in the window.

5   Double-click uo_event to display its Script view.

The two new user events display in the second drop-down list in the Script view.

6   Write scripts for the two user events:

- max_requested: `Parent.WindowState = Maximized!`

- min_requested: `Parent.WindowState = Minimized!`

These scripts reference the window containing the user object with the pronoun Parent.

What happens   When a user clicks a button, the Clicked event script for that button triggers a user event in its parent, the user object. The user object script for that event modifies its parent, the window.

P A R T 5   # Working with Databases

This part describes how to use PowerBuilder to manage your database and how to use the Data Pipeline painter to copy data from one database to another.

Managing the Database

About this chapter   This chapter describes how to manage the database from within
PowerBuilder.

Contents

Before you begin   You work with relational databases in PowerBuilder. If you are not
familiar with relational databases, you might want to consult an
introductory text, such as *A Database Primer* by C. J. Date.

Working with database components

A database is an electronic storage place for data. Databases are designed
to ensure that data is valid and consistent and that it can be accessed,
modified, and shared.

A database management system (DBMS) governs the activities of a
database and enforces rules that ensure data integrity. A *relational* DBMS
stores and organizes data in tables.

How you work with databases in PowerBuilder

You can use PowerBuilder to work with the following database components:

- Tables and columns

- Keys

- Indexes

- Database views

- Extended attributes

- Additional database components

Tables and columns

A database usually has many tables, each of which contains rows and columns of data. Each row in a table has the same columns, but a column's value for a particular row could be empty or NULL if the column's definition allows it.

Tables often have relationships with other tables. For example, in the EAS Demo DB included with PowerBuilder, the Department table has a Dept_id column, and the Employee table also has a Dept_id column that identifies the department in which the employee works. When you work with the Department table and the Employee table, the relationship between them is specified by a join of the two tables.

Keys

Relational databases use keys to ensure database integrity.

Primary keys   A primary key is a column or set of columns that uniquely identifies each row in a table. For example, two employees may have the same first and last names, but they have unique ID numbers. The Emp_id column in the Employee table is the primary key column.

Foreign keys   A foreign key is a column or set of columns that contains primary key values from another table. For example, the Dept_id column is the primary key column in the Department table and a foreign key in the Employee table.

Key icons   In PowerBuilder, columns defined as keys are displayed with key icons that use different shapes and colors for primary and foreign. PowerBuilder automatically joins tables that have a primary/foreign key relationship, with the join on the key columns.

In the following illustration there is a join on the dept_id column, which is a primary key for the department table and a foreign key for the employee table:
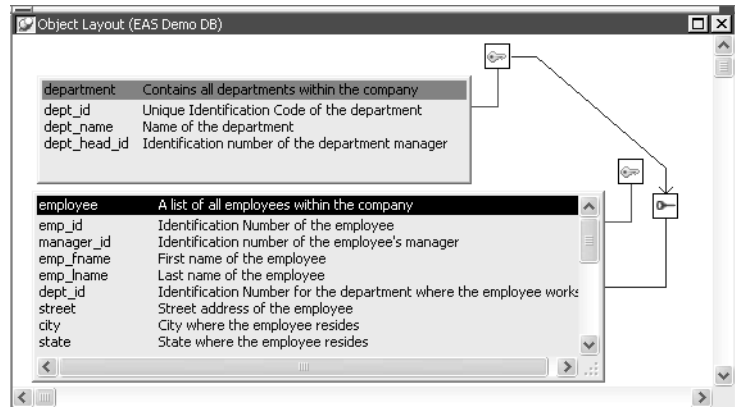
For more information, see "Working with keys" on page 411.

Indexes

An index is a column or set of columns you identify to improve database performance when searching for data specified by the index. You index a column that contains information you will need frequently. Primary and foreign keys are special examples of indexes.

You specify a column or set of columns with unique values as a unique index, represented by an icon with a single key.

You specify a column or set of columns that has values that are not unique as a duplicate index, represented by an icon with two keys.

For more information, see "Working with indexes" on page 416.

Database views

If you often select data from the same tables and columns, you can create a database view of the tables. You give the database view a name, and each time you refer to it the associated SELECT command executes to find the data.

Database views are listed in the Objects view of the Database painter and can be displayed in the Object Layout view, but a database view does not physically exist in the database in the same way that a table does. Only its definition is stored in the database, and the view is re-created whenever the definition is used.

Database administrators often create database views for security purposes. For example, a database view of an Employee table that is available to users who are not in Human Resources might show all columns except Salary.

For more information, see "Working with database views" on page 417.

Extended attributes

Extended attributes enable you to store information about a table's columns in special system tables. Unlike tables, keys, indexes, and database views (which are DBMS-specific), extended attributes are PowerBuilder-specific. The most powerful extended attributes determine the edit style, display format, and validation rules for the column.

For more information about extended attributes, see "Specifying column extended attributes" on page 401. For more information about the extended attribute system tables, see Appendix A, "The Extended Attribute System Tables."

Additional database components

Depending on the database to which you are connected and on your user privileges, you might be able to view or work with a variety of additional database components through PowerBuilder. These components might include:

> Driver information
> Groups
> Metadata types
> Procedures and functions
> Users
> Logins

For example, driver information is relevant to ODBC connections. It lists all the ODBC options associated with the ODBC driver, allowing you to determine how the ODBC interface will behave for a given connection. Login information is listed for Adaptive Server Enterprise (SYC) database connections. Information about groups and users is listed for several of the databases and allows you to add new users and groups and maintain passwords for existing users.

Managing databases

PowerBuilder supports many database management systems (DBMSs). For the most part, you work the same way in PowerBuilder for each DBMS, but because each DBMS provides some unique features (which PowerBuilder makes use of), there are some issues that are specific to a particular DBMS. For complete information about using your DBMS, see *Connecting to Your Database*.

What you can do

Using the Database painter, you can do the following in any DBMS to which you have been given access by the database administrator:

- Modify local table and column properties
- Retrieve, change, and insert data
- Create new local tables or modify existing tables

Setting the database connection

When you open a painter that communicates with the database (such as the Database painter or DataWindow painter), PowerBuilder connects you to the database you used last if you are not already connected. If the connection to the default database fails, the painter still opens.

If you do not want to connect to the database you used last, you can deselect the Connect to Default Profile option in the Database Preferences dialog box.

Changing the database connection

You can change to a different database at any time. You can have several database connections open at a time, although only one connection can be active. The database components for each open connection are listed in the Objects view.

The Database painter title bar displays the number of open connections and shows which is active. The title bar for each view displays the connection with which it is currently associated. You can change the connection associated with a view by dragging the profile name for a different connection onto the view.

For more about changing the database you are connected to, see *Connecting to Your Database*.

Creating and deleting databases

When you are connected to Adaptive Server Anywhere, you can create a new database or delete an existing database using the Database painter.

For all other DBMSs, creating and deleting a database is an administrative task that you cannot do within PowerBuilder.

Using the Database painter

To open the Database painter, click the Database button in the PowerBar.

About the painter

Like the other PowerBuilder painters, the Database painter contains a menu bar, customizable PainterBars, and several views. All database-related tasks that you can do in PowerBuilder can be done in the Database painter.
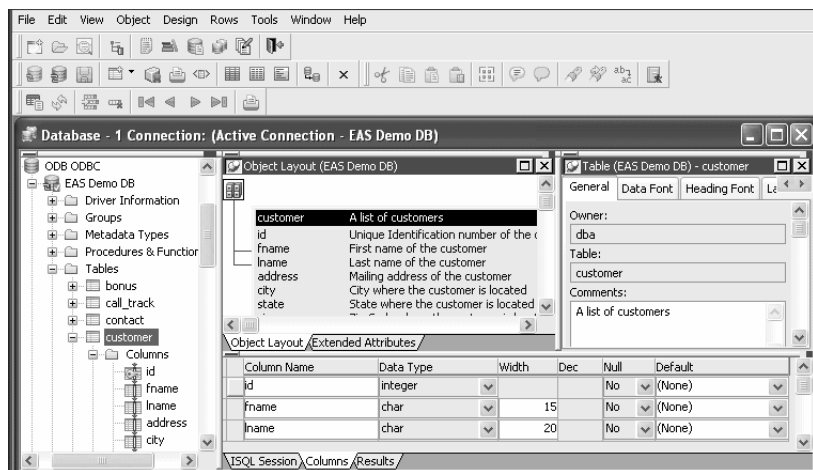
Views in the Database painter

Table 16-1 lists the views available in the Database painter.

Table 16-1: Database painter views

View	Description
Activity Log	Displays the SQL syntax generated by the actions you execute.
Columns	Used to create and/or modify a table's columns (formerly done in the Table painter).
Extended Attributes	Lists the display formats, edit styles, and validation rules defined for the selected database connection.
Interactive SQL	Used to build, execute or explain SQL. (In PowerBuilder 6 and earlier, some of this functionality was in the Database Administration painter.)
Object Details	Displays an object's properties. For some objects, properties are read-only; for others, properties can be modified. This view is analogous to the Properties view in other painters.

View	Description
Object Layout	Displays a graphical representation of tables.
Objects	Lists the database interfaces and profiles. For an active database connection, might also list all or some of the following objects associated with that database: groups, metadata types, procedures and functions, tables, columns, primary and foreign keys, indexes, users, views, driver information, and utilities (the database components listed depend on the database and your user privileges).
Results	Displays data in a grid, table, or freeform format.

Dragging and dropping

You can select certain database objects from the Objects view and drag them to the Object Details, Object Layout, Columns, and/or ISQL views. Just position the pointer on the database object's icon and drag it to the appropriate view.

Table 16-2: Using drag and drop in the Database painter

Object	Can be dragged to
Driver, group, metadata type, procedure or function, table, column, user, primary or foreign key, index	Object Details view
Table or view	Object Layout view
Table or column	Columns view
Procedure or view	ISQL view

Database painter tasks

Table 16-3 describes how to do some basic tasks in the Database painter. Most of these tasks begin in the Objects view. Many can be accomplished by dragging and dropping objects into different views. If you prefer, you can use buttons or menu selections from the main bar or from pop-up menus.

Table 16-3: Common tasks in the Database painter

To	Do this
Modify a database profile	Highlight a database profile and select Properties from the Object or pop-up menu or use the Properties button. Alternatively, use the Import and Export Profiles menu selections to copy profiles.
	For more information, see the section on importing and exporting database profiles in *Connecting to Your Database*.
Connect to a database	Highlight a database profile and then select Connect from the File or pop-up menu or use the Connect button. With File>Recent Connections, you can review and return to earlier connections. You can also make database connections using the Database Profile button.

To	Do this
Create new profiles, tables, views, columns, keys, indexes, or groups	Highlight the database object and select New from the Object or pop-up menu or use the Create button.
Modify database objects	Drag the object to the Object Details view.
Graphically display tables	Drag the table icon from the list in the Objects view to the Object Layout view, or highlight the table and select Add To Layout from the Object or pop-up menu.
Manipulate data	Highlight the table and select Grid, Tabular, or Freeform from the Object>Data menu or the pop-up menu Edit Data item, or use the appropriate Data Manipulation button.
Build, execute or explain SQL	Use the ISQL view to build SQL statements. Use the Paste SQL button to paste SELECT, INSERT, UPDATE, and DELETE statements or type them directly into the view's workspace. To execute or explain SQL, select Execute SQL and Explain SQL from the Design or pop-up menu.
Define or modify extended attributes	Select from the Object>Insert menu the type of extended attribute you want to define or modify, or highlight the extended attribute from the list in the Extended Attributes view and select New or Properties from the pop-up menu.
Specify extended attributes for a column	Drag the column to the Object Details view and select the Extended Attributes tab.
Access database utilities	Double-click a utility in the Objects view to launch it.
Log your work	Select Design>Start Log from the menu bar. To see the SQL syntax generated, display the Activity Log view.

Modifying database preferences

To modify database preferences, select Design>Options from the menu bar. Some preferences are specific to the database connection; others are specific to the Database painter.

Preferences on the General property page

The Connect To Default Profile, Shared Database Profiles, Keep Connection Open, Use Extended Attributes, and Read Only preferences are specific to the database connection.

The remaining preferences are specific to the Database painter. For information about modifying these preferences, see *Connecting to Your Database*.

Table 16-4: Database painter preferences

Database preference	What PowerBuilder does with the specified preference
Columns in the Table List	When PowerBuilder displays tables graphically, eight table columns display unless you change the number of columns.
SQL Terminator Character	PowerBuilder uses the semicolon as the SQL statement terminator unless you enter a different terminator character in the box. Make sure that the character you choose is not reserved for another use by your database vendor. For example, using the slash character (/) causes compilation errors with some DBMSs.
Refresh Table List	When PowerBuilder first displays a table list, PowerBuilder retrieves the table list from the database and displays it. To save time, PowerBuilder saves this list internally for reuse to avoid regeneration of very large table lists. The table list is refreshed every 30 minutes (1800 seconds) unless you specify a different refresh rate.

Preferences on the Object Colors property page

You can set colors separately for each component of the Database painter's graphical table representation: the table header, columns, indexes, primary key, foreign keys, and joins. Set a color preference by selecting a color from a drop-down list.

You can design custom colors that you can use when you select color preferences. To design custom colors, select Design>Custom Colors from the menu bar and work in the Custom Colors dialog box.

Logging your work

As you work with your database, you generate SQL statements. As you define a new table, for example, PowerBuilder builds a SQL CREATE TABLE statement internally. When you click the Create button, PowerBuilder sends the SQL statement to the DBMS to create the table. Similarly, when you add an index, PowerBuilder builds a CREATE INDEX statement.

You can see all SQL generated in a Database painter session in the Activity Log view. You can also save this information to a file. This allows you to have a record of your work and makes it easy to duplicate the work if you need to create the same or similar tables in another database.

❖ **To start logging your work:**

1   Open the Database painter.

2   Select Start Log from the Design menu or the pop-up menu in the Activity Log view.

   PowerBuilder begins sending all generated syntax to the Activity Log view.

❖ **To stop the log:**

•   Select Stop Log from the Design menu or the pop-up menu in the Activity Log view.

   PowerBuilder stops sending the generated syntax to the Activity Log view. Your work is no longer logged.

❖ **To save the log to a permanent text file:**

1   Select Save or Save As from the File menu.

2   Name the file and click Save. The default file extension is *SQL*, but you can change that if you want to.

Submitting the log to your DBMS
You can open a saved log file and submit it to your DBMS in the ISQL view. For more information, see "Building and executing SQL statements" on page 429.

Creating and deleting an Adaptive Server Anywhere database

In PowerBuilder you work within an existing database. With one exception, creating or deleting a database is an administrative task that is not performed directly in PowerBuilder. The one exception is that you can create and delete a local Adaptive Server Anywhere database from within PowerBuilder.

For information about creating and deleting other databases, see your DBMS documentation.

❖ **To create a local Adaptive Server Anywhere database:**

1   From the Objects view, launch the Create ASA Database utility included with the ODBC interface.

The Create Adaptive Server Anywhere Database dialog box displays.

2   In the Database Name box, specify the file name and path of the database you are creating.

If you do not provide a file extension, the database file name is given the extension *DB*.

3   Define other properties of the database as needed.

If you are using a non-English database, you can specify a code page in the Collation Sequence box.

For complete information about filling in the dialog box, click the Help button in the dialog box.

4   Click OK.

When you click OK, PowerBuilder does the following:

- Creates a database with the specified name in the specified directory or folder. If a database with the same name exists, you are asked whether you want to replace it.

- Adds a data source to the *ODBC.INI* key in the registry. The data source has the same name as the database unless one with the same name already exists, in which case a suffix is appended.

- Creates a database profile and adds it to the registry. The profile has the same name as the database unless one with the same name already exists, in which case a suffix is appended.

- Connects to the new database.

❖ **To delete a local Adaptive Server Anywhere database:**

1   Open the Database painter.

2   From the Objects view, launch the Delete ASA Database utility included with the ODBC interface.

3   Select the database you want to delete and select Open.

4   Click Yes to delete the database.

When you click Yes, PowerBuilder deletes the specified database.

Working with tables

When you open the Database painter, the Object view lists all tables in the current database that you have access to (including tables that were not created using PowerBuilder). You can create a new table or alter an existing table. You can also modify table properties and work with indexes and keys.

Creating a new table from scratch

In PowerBuilder, you can create a new table in any database to which PowerBuilder is connected.

❖ **To create a table in the current database:**

1   Do one of the following:

   • Click the Create Table button.

   • Right-click in the Columns view and select New Table from the pop-up menu.

   • Right-click Tables in the Objects view and select New Table from the pop-up menu.

   • Select Insert>Table from the Object menu.

   The new table template displays in the Columns view. What you see in the view is DBMS-dependent. You use this template to specify each column in the table. The insertion point is in the Column Name box for the first column.

2   Enter the required information for this column.

   For what to enter in each field, see "Specifying column definitions" on page 400.

   As you enter information, use the Tab key to move from place to place in the column definition. After defining the last item in the column definition, press the Tab key to display the work area for the next column.

3   Repeat step 2 for each additional column in your table.

4   (Optional) Select Object>Pending SQL from the menu bar or select Pending SQL from the pop-up menu to see the pending SQL syntax.

   If you have not already named the table, you must provide a name in the dialog box that displays. To hide the SQL syntax and return to the table columns, select Object>Pending Syntax from the menu bar.

5   Click the Save button or select Save from the File or pop-up menu, then enter a name for the table in the Create New Table dialog box.

PowerBuilder submits the pending SQL syntax statements it generated to the DBMS, and the table is created. The new table is displayed in the Object Layout view.

About saving the table
If you make changes after you save the table and before you close it, you see the pending changes when you select Pending SQL again. When you click Save again, PowerBuilder submits a DROP TABLE statement to the DBMS, recreates the table, and applies all changes that are pending. Clicking Save many times can be time consuming when you are working with large tables, so you might want to save only when you have finished.

---

6   Specify extended attributes for the columns.

For what to enter in each field, see "Specifying column extended attributes" on page 401.

Creating a new table from an existing table

You can create a new table that is similar to an existing table very quickly by using the Save Table As menu option.

❖   **To create a new table from an existing table:**

1   Open the existing table in the Columns view by dragging and dropping it or selecting Alter Table from the pop-up menu.

2   Right-click in the Columns view and select Save Table As from the pop-up menu.

3   Enter a name for the new table and then the owner's name, and click OK.

The new table appears in the Object Layout view and the Columns view.

4   Make whatever changes you want to the table definition.

5   Save the table.

6   Make changes to the table's properties in the Object Details view.

For more information about modifying table properties, see "Specifying table and column properties" on page 400.

Specifying column definitions

When you create a new table, you must specify a definition for each column. The fields that display for each column in the Columns view depend on your DBMS. You might not see all of the following fields, and the values that you can enter are dependent on the DBMS.

For more information, see your DBMS documentation.

Table 16-5: Defining columns in the Columns view in the Database painter

Field	What you enter
Column Name	(Required) The name by which the column will be identified.
Data Type	(Required) Select a datatype from the drop-down list. All datatypes supported by the current DBMS are displayed in the list.
Width	For datatypes with variable widths, the number of characters in the field.
Dec	For numeric datatypes, the number of decimal places to display.
Null	Select Yes or No from the Null drop-down list to specify whether NULLs are allowed in the column. Specifying No means the column cannot have NULL values; users must supply a value. No is the default in a new table.
Default	The value that will be placed in a column in a row that you insert into a DataWindow object. The drop-down list has built-in choices, but you can type any other value. For an explanation of the built-in choices, see your DBMS documentation.

Specifying table and column properties

After you create and save a table, you can specify the properties of the table and of any or its columns. Table properties include the fonts used for headers, labels, and data, and a comment that you can associate with the table. Column properties include the text used for headers and labels, display formats, validation rules, and edit styles used for data (also known as a column's extended attributes), and a comment you can associate with the column.

Specifying table properties

In addition to adding a comment to associate with the table, you can choose the fonts that will be used to display information from the table in a DataWindow object. You can specify the font, point size, color, and style.

❖   **To specify table properties:**

1   Do one of the following:

  •   Highlight the table in either the Objects view or the Object Layout view and select Properties from the Object or pop-up menu.

  •   Click the Properties button.

  •   Drag and drop the table to the Object Details view.

  The properties for the table display in the Object Details view.

2   Select a tab and specify properties:

Select this tab	To modify this property
General	Comments associated with the table
Data Font	Font for data retrieved from the database and displayed in the Results view by clicking a Data Manipulation button
Heading Font	Font for column identifiers used in grid, tabular, and n-up DataWindow objects displayed in the Results view by clicking a Data Manipulation button
Label Font	Font for column identifiers used in freeform DataWindow objects displayed in the Results view by clicking a Data Manipulation button

3   Right-click on the Object Details view and select Save Changes from the pop-up menu.

  Any changes you made in the Object Details view are immediately saved to the table definition.

Specifying column extended attributes

In addition to adding a comment to associate with a column, you can specify extended attributes for each column. An extended attribute is information specific to PowerBuilder that enhances the definition of the column.

❖ **To specify extended attributes:**

1 Do one of the following:

- Highlight the column in either the Objects view or the Object Layout view and select Properties from the Object or pop-up menu.

- Click the Properties button.

- Drag and drop the column to the Object Details view.

2 Select a tab and specify extended attribute values:

Select this tab	To modify these extended attributes
General	Column comments.
Headers	Label text used in free-form DataWindow objects
	Header text used in tabular, grid, or n-up DataWindow objects
Display	How the data is formatted in a DataWindow object as well as display height, width, and position. For example, you can associate a display format with a Revenue column so that its data displays with a leading dollar sign and negative numbers display in parentheses.
Validation	Criteria that a value must pass to be accepted in a DataWindow object. For example, you can associate a validation rule with a Salary column so that you can enter a value only within a particular range.
	The initial value for the column. You can select a value from the drop-down list. The initial value must be the same datatype as the column, must pass validation, and can be NULL only if NULL is allowed for the column.
Edit Style	How the column is presented in a DataWindow object. For example, you can display column values as radio buttons or in a drop-down list.

3 Right-click on the Column property sheet and select Save Changes from the pop-up menu.

Any changes you made in the property sheet are immediately saved to the table definition.

Overriding definitions
In the DataWindow painter, you can override the extended attributes specified in the Database painter for a particular DataWindow object.

How the information is stored	Extended attributes are stored in the PowerBuilder system tables in the database. PowerBuilder uses the information to display, present, and validate data in the Database painter and in DataWindow objects. When you create a view in the Database painter, the extended attributes of the table columns used in the view are used by default.
About display formats, edit styles, and validation rules	In the Database painter, you create display formats, edit styles, and validation rules. Whatever you create is then available for use with columns in tables in the database. You can see all the display formats, edit styles, and validation rules defined for the database in the Extended Attributes view.

For more information about defining, maintaining, and using these extended attributes, see Chapter 22, "Displaying and Validating Data." |
| About headings and labels | By default, PowerBuilder uses the column names as labels and headings, replacing any underscore characters with spaces and capitalizing each word in the name. For example, the default heading for the column Dept_name is Dept Name. To define multiple-line headings, press Ctrl+Enter to begin a new line. |

Specifying additional properties for character columns

You can also set two additional properties for character columns on the Display property page: Case and Picture.

Specifying the displayed case	You can specify whether PowerBuilder converts the case of characters for a column in a DataWindow object.

❖ **To specify how character data should be displayed:**

• On the Display property page, select a value in the Case drop-down list:

Value	Meaning
Any	Characters are displayed as they are entered
UPPER	Characters are converted to uppercase
lower	Characters are converted to lowercase

Specifying a column as a picture	You can specify that a character column can contain names of picture files.

❖ **To specify that column values are names of picture files:**

1 On the Display property page, select the Picture check box.

When the Picture check box is selected, PowerBuilder expects to find picture file names in the column and displays the contents of the picture file—not the name of the file—in reports and DataWindow objects.

Because PowerBuilder cannot determine the size of the image until runtime, it sets both display height and display width to 0 when you select the Picture check box.

2 Enter the size and the justification for the picture (optional).

Altering a table

After a table is created, how you can alter the table depends on your DBMS.

You can always:

- Add or modify PowerBuilder-specific extended attributes for columns
- Delete an index and create a new index

You can never:

- Insert a column between two existing columns
- Prohibit null values for an appended column
- Alter an existing index

Some DBMSs let you do the following, but others do not:

- Append columns that allow null values
- Increase or decrease the number of characters allowed for data in an existing column
- Allow null values
- Prohibit null values in a column that allowed null values

Database painter is DBMS aware
The Database painter grays out or notifies you about actions that your DBMS prohibits.

For complete information about what you can and cannot do when you modify a table in your DBMS, see your DBMS documentation.

❖   **To alter a table:**

1   Highlight the table and select Alter Table from the pop-up menu.

Opening multiple instances of tables
You can open another instance of a table by selecting Columns from the
View menu. Doing this is helpful when you want to use the Database
painter's cut, copy, and paste features to cut or copy and paste between
tables.

The table definition displays in the Columns view (this screen shows the
Employee table).

2   Make the changes you want in the Columns view or in the Object Details
view.

3   Select Save Table or Save Changes.

PowerBuilder submits the pending SQL syntax statements it generated to
the DBMS, and the table is modified.

Cutting, copying, and pasting columns

In the Database painter, you can use the Cut, Copy, and Paste buttons in the
PainterBar (or Cut, Copy, and Paste from the Edit or pop-up menu) to cut, copy,
and paste one column at a time within a table or between tables.

❖   **To cut or copy a column within a table:**

1   Put the insertion point anywhere in the column you want to cut or copy.

2   Click the Cut or Copy button in the PainterBar.

❖ **To paste a column within a table:**

1   Put the insertion point in the column you want to paste to.

 If you are changing an existing table, put the insertion point in the last column of the table. If you try to insert a column between two columns, you get an error message. To an existing table, you can only append a column. If you are defining a new table, you can paste a column anywhere.

2   Click the Paste button in the PainterBar.

❖ **To paste a column to a different table:**

1   Open another instance of the Columns view and use Alter Table to display an existing table or click New to create a new table.

2   Put the insertion point in the column you want to paste to.

3   Click the Paste button in the PainterBar.

Closing a table

You can remove a table from a view by selecting Close or Reset View from its pop-up menu. This action only removes the table from the Database painter view. It does not drop (remove) the table from the database.

Dropping a table

Dropping removes the table from the database.

❖ **To drop a table:**

1   Select Drop Table from the table's pop-up menu or select Object>Delete from the menu bar.

2   Click Yes.

Deleting orphaned table information

If you drop a table outside PowerBuilder, information remains in the system tables about the table, including extended attributes for the columns.

❖ **To delete orphaned table information from the extended attribute system tables:**

•   Select Design>Synch Extended Attributes from the menu bar and click Yes.

 If you try to delete orphaned table information and there is none, a message tells you that synchronization is not necessary.
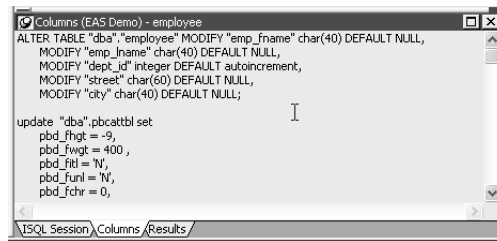
Viewing pending SQL changes

As you create or alter a table definition, you can view the pending SQL syntax changes that will be made when you save the table definition.

❖ **To view pending SQL syntax changes:**

• Right-click the table definition in the Columns view and select Pending Syntax from the pop-up menu.

   PowerBuilder displays the pending changes to the table definition in SQL syntax:

The SQL statements execute only when you save the table definition or reset the view and then tell PowerBuilder to save changes.

Copying, saving, and printing pending SQL changes

When you are viewing pending SQL changes, you can:

• Copy pending changes to the clipboard

• Save pending changes to a file

• Print pending changes

To copy, save, or print only part of the SQL syntax
Select the part of the SQL syntax you want before you copy, save, or print.

❖ **To copy the SQL syntax to the clipboard:**

• In the Pending Syntax view, click the Copy button or select Copy from the pop-up menu.

❖ **To save SQL syntax for execution at a later time:**

1 In the Pending Syntax view, Select File>Save As.

   The Save Syntax to File dialog box displays.

2 Navigate to the folder where you want to save SQL, name the file, and then click the Save button.

At a later time, you can import the SQL file into the Database painter and execute it.

❖ **To print pending table changes:**

• While viewing the pending SQL syntax, click the Print button or select Print from the File menu.

❖ **To display columns in the Columns view:**

• Select Object>Pending Syntax from the menu bar.

Printing the table definition

You can print a report of the table's definition at any time, whether or not the table has been saved. The Table Definition Report contains information about the table and each column in the table, including the extended attributes for each column.

❖ **To print the table definition:**

• Select Print or Print Definition from the File or pop-up menu or click the Print button.

Exporting table syntax

You can export the syntax for a table to the log. This feature is useful when you want to create a backup definition of the table before you alter it or when you want to create the same table in another DBMS.

To export to another DBMS, you must have the PowerBuilder interface for that DBMS.

❖ **To export the syntax of an existing table to a log:**

1 Select the table in the painter workspace.

2 Select Export Syntax from the Object menu or the pop-up menu.

   If you selected a table and have more than one DBMS interface installed, the DBMS dialog box displays. If you selected a view, PowerBuilder immediately exports the syntax to the log.

3 Select the DBMS to which you want to export the syntax.

4   If you selected ODBC, specify a data source in the Data Sources dialog box.

5   Supply any information you are prompted for.

PowerBuilder exports the syntax to the log. Extended attribute information (such as validation rules used) for the selected table is also exported. The syntax is in the format required by the DBMS you selected.

For more information about the log, see "Logging your work" on page 395.

About system tables

Two kinds of system tables exist in the database:

• System tables provided by your DBMS (for more information, see your DBMS documentation)

• PowerBuilder extended attribute system tables

About PowerBuilder system tables

PowerBuilder stores extended attribute information you provide when you create or modify a table (such as the text to use for labels and headings for the columns, validation rules, display formats, and edit styles) in system tables. These system tables contain information about database tables and columns. Extended attribute information extends database definitions.

In the Employee table, for example, one column name is Emp_lname. A label and a heading for the column are defined for PowerBuilder to use in DataWindow objects. The column label is defined as `Last Name:`. The column heading is defined as `Last Name`. The label and heading are stored in the PBCatCol table in the extended attribute system tables.

The extended attribute system tables are maintained by PowerBuilder, and only PowerBuilder users can enter information into them. Table 16-6 lists the extended attribute system tables. For more information, see Appendix A, "The Extended Attribute System Tables."

Table 16-6: Extended attribute system tables

This system table	Stores this extended attribute information
PBCatCol	Column data such as name, header and label for reports and DataWindow objects, and header and label positions
PBCatEdt	Edit style names and definitions
PBCatFmt	Display format names and definitions
PBCatTbl	Table data such as name, fonts, and comments

This system table	Stores this extended attribute information
PBCatVld	Validation rule names and definitions

Opening and displaying system tables

You can open system tables like other tables, in the Database painter.

By default, PowerBuilder shows only user-created tables in the Objects view. If you highlight Tables and select Show System Tables from the pop-up menu, PowerBuilder also displays system tables.

Creating and editing Sybase Adaptive Server temporary tables

You can create and edit temporary tables in the Database painter, SQL painter, or DataWindow painter when you use the PowerBuilder SYC native driver to connect to an ASE database. Temporary tables persist for the duration of a database connection, residing in a special database called "tempdb".

You add a temporary table to the tempdb database by assigning a name that starts with the # character when you create a new table in a PowerBuilder painter. (Temporary tables must start with the # character.)

After you create a temporary table, you can create indexes and a primary key for the table. If you define a unique index or primary key, you can execute INSERT, UPDATE, and DELETE statements for a temporary table. Selecting Edit Data from the pop-up menu of a temporary table retrieves data that you store in that table.

Standard catalog query limitations
When you click Refresh from the pop-up menu for the Tables node in the Database painter or the Objects view of the DataWindow painter, the list of tables displays temporary tables even though they exist only in the tempdb database. However, once you refresh table definitions from the database, the Objects view can no longer list the index or primary key information of the temporary tables, and the Layout view can no longer display that information graphically.

You can create DataWindow objects that access temporary tables in a PowerBuilder runtime application, but your application must first explicitly create the temporary tables, along with the appropriate keys and indexes, using the same database transaction object used by the DataWindow.

You can use the EXECUTE IMMEDIATE PowerScript syntax to create temporary tables at runtime:

```
string s1, s2, s3, s4
s1 = 'create table dbo.#temptab1 (id int not null, ' &
     + 'lname char(20) not null) '
s2 = 'alter table dbo.#temptab1 add constraint idkey' &
     + ' primary key clustered (id) '
s3 = 'create nonclustered index nameidx on ' &
     + 'dbo.#temptab1 (lname ) '
s4 = 'insert into #temptab1 select emp_id, ' &
     + 'emp_lname from qadb_emp'
execute immediate :s1 using sqlca;
if sqlca.sqlcode = 0 then
   execute immediate :s2 using sqlca;
   execute immediate :s3 using sqlca;
   execute immediate :s4 using sqlca;
else
   messagebox("Create error", sqlca.sqlerrtext)
end if
```

Working with keys

If your DBMS supports primary and foreign keys, you can work with the keys in PowerBuilder.

Why you should use keys

If your DBMS supports them, you should use primary and foreign keys to enforce the referential integrity of your database. That way you can rely on the DBMS to make sure that only valid values are entered for certain columns instead of having to write code to enforce valid values.

For example, say you have two tables called Department and Employee. The Department table contains the column Dept_Head_ID, which holds the ID of the department's manager. You want to make sure that only valid employee IDs are entered in this column. The only valid values for Dept_Head_ID in the Department table are values for Emp_ID in the Employee table.

To enforce this kind of relationship, you define a foreign key for Dept_Head_ID that points to the Employee table. With this key in place, the DBMS disallows any value for Dept_Head_ID that does not match an Emp_ID in the Employee table.

For more about primary and foreign keys, consult a book about relational database design or your DBMS documentation.

What you can do in the Database painter

You can work with keys in the following ways:

- Look at existing primary and foreign keys

- Open all tables that depend on a particular primary key

- Open the table containing the primary key used by a particular foreign key

- Create, alter, and drop keys

For the most part, you work with keys the same way for each DBMS that supports keys, but there are some DBMS-specific issues. For complete information about using keys with your DBMS, see your DBMS documentation.

Viewing keys

Keys can be viewed in several ways:

- In the expanded tree view of a table in the Objects view

- As icons connected by lines to a table in the Object Layout view

In the following picture, the Department table has two keys:

- A primary key (on dept_id)

- A foreign key (on dept_head_id)
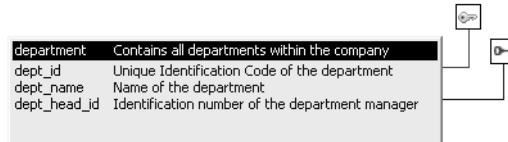
If you cannot see the lines
If the color of your window background makes it hard to see the lines for the keys and indexes, you can set the colors for each component of the Database painter's graphical table representation, including keys and indexes. For information, see "Modifying database preferences" on page 394.

Opening related tables

When working with tables containing keys, you can easily open related tables.

❖ **To open the table that a particular foreign key references:**

1   Display the foreign key pop-up menu.

2   Select Open Referenced Table.

❖ **To open all tables referencing a particular primary key:**

1   Display the primary key pop-up menu.

2   Select Open Dependent Table(s).

PowerBuilder opens and expands all tables in the database containing foreign keys that reference the selected primary key.

Defining primary keys

If your DBMS supports primary keys, you can define them in PowerBuilder.

❖ **To create a primary key:**

1 Do one of the following:

- Highlight the table for which you want to create a primary key and click the Create Primary Key drop-down toolbar button in PainterBar1.

- Select Object>Insert>Primary Key from the main menu or New>Primary Key from the pop-up menu.

- Expand the table's tree view, right-click Primary Key, and select New Primary Key from the pop-up menu.

The Primary Key properties display in the Object Details view.

2 Select one or more columns for the primary key.

> **Columns that are allowed in a primary key**
> Only a column that does not allow null values can be included as a column in a primary key definition. If you choose a column that allows null values, you get a DBMS error when you save the table. In DBMSs that allow rollback for Data Definition Language (DDL), the table definition is rolled back. In DBMSs that do not allow rollback for DDL, the painter is refreshed with the current definition of the table.

3 Specify any information required by your DBMS.

> **Naming a primary key**
> Some DBMSs allow you to name a primary key and specify whether it is clustered or not clustered. For these DBMSs, the Primary Key property page has a way to specify these properties.

For DBMS-specific information, see your DBMS documentation.

4 Right-click on the Object Details view and select Save Changes from the pop-up menu.

Any changes you made in the view are immediately saved to the table definition.

Completing the primary key
Some DBMSs automatically create a unique index when you define a primary key so that you can immediately begin to add data to the table. Others require you to create a unique index separately to support the primary key before populating the table with data.

To find out what your DBMS does, see your DBMS documentation.

Defining foreign keys   If your DBMS supports foreign keys, you can define them in PowerBuilder.

❖ **To create a foreign key:**

1   Do one of the following:

   • Highlight the table and click the Create Foreign Key drop-down toolbar button in PainterBar1.

   • Select Object>Insert>Foreign Key from the main menu or New>Foreign Key from the pop-up menu.

   • Expand the table's tree view and right-click on Foreign Keys and select New Foreign Key from the pop-up menu.

   The Foreign Key properties display in the Object Details view. Some of the information is DBMS-specific.

2   Name the foreign key in the Foreign Key Name box.

3   Select the columns for the foreign key.

4   On the Primary Key tab page, select the table and column containing the Primary key referenced by the foreign key you are defining.

   Key definitions must match exactly
   The definition of the foreign key columns must match the primary key columns, including datatype, precision (width), and scale (decimal specification).

5   On the Rules tab page, specify any information required by your DBMS.

   For example, you might need to specify a delete rule by selecting one of the rules listed for On Delete of Primary Table Row.

   For DBMS-specific information, see your DBMS documentation.

6   Right-click on the Object Details view and select Save Changes from the pop-up menu.

Any changes you make in the view are immediately saved to the table definition.

Modifying keys   You can modify a primary key in PowerBuilder.

❖   **To modify a primary key:**

1   Do one of the following:

- Highlight the primary key listed in the table's expanded tree view and click the Properties button.

- Select Properties from the Object or pop-up menu.

- Drag the primary key icon and drop it in the Object Details view.

2   Select one or more columns for the primary key.

3   Right-click on the Object Details view and select Save Changes from the pop-up menu.

Any changes you make in the view are immediately saved to the table definition.

Dropping a key   You can drop keys (remove them from the database) from within PowerBuilder.

❖   **To drop a key:**

1   Highlight the key in the expanded tree view for the table in the Objects view or right-click the key icon for the table in the Object Layout view.

2   Select Drop Primary Key or Drop Foreign Key from the key's pop-up menu.

3   Click Yes.

Working with indexes

You can create as many single- or multi-valued indexes for a database table as you need, and you can drop indexes that are no longer needed.

Update limitation
You can update a table in a DataWindow object only if it has a unique index or primary key.

Creating an index

In Adaptive Server Anywhere databases
In Adaptive Server Anywhere databases, you should not define an index on a column that is defined as a foreign key, because foreign keys are already optimized for quick reference.

❖ **To create an index:**

1 Do one of the following:

- Highlight the table for which you want to create an index and click the Create Index drop-down toolbar button in PainterBar1.

- Select Object>Insert>Index from the main menu or New>Index from the pop-up menu.

- Expand the table's tree view, right-click on Indexes, and select New Index from the pop-up menu.

The Index's properties display in the Object Details view.

2 Enter a name for the index in the Index box.

3 Select whether or not to allow duplicate values for the index.

4 Specify any other information required for your database

For example, in Adaptive Server Enterprise, specify whether the index is clustered, and in Adaptive Server Anywhere, specify the order of the index.

5 Click the names of the columns that make up the index.

6 Select Save Changes from the pop-up menu.

7 Right-click on the Object Details view and select Save Changes from the pop-up menu.

Any changes you made in the view are immediately saved to the table definition.

Modifying an index   You can modify an index.

❖   **To modify an index:**

1   Do one of the following:

- Highlight the index listed in the table's expanded tree view and click the Properties button.

- Select Properties from the Object or pop-up menu.

- Drag the index icon and drop it in the Object Details view.

2   In the Object Details view, select or deselect columns as needed.

3   Right-click on the Object Details view and select Save Changes from the pop-up menu.

Any changes you made in the view are immediately saved to the table definition.

Dropping an index   Dropping an index removes it from the database.

❖   **To drop an index from a table:**

1   In the Database painter workspace, display the pop-up menu for the index you want to drop.

2   Select Drop Index and click Yes.

Working with database views

A database view gives a different (and usually limited) perspective of the data in one or more tables. Although you see existing database views listed in the Objects view, a database view does not physically exist in the database as a table does. Each time you select a database view and use the view's data, PowerBuilder executes a SQL SELECT statement to retrieve the data and creates the database view.

For more information about using database views, see your DBMS documentation.

Using database views
in PowerBuilder

You can define and manipulate database views in PowerBuilder. Typically you use database views for the following reasons:

• To give names to frequently executed SELECT statements.

• To limit access to data in a table. For example, you can create a database view of all the columns in the Employee table except Salary. Users of the database view can see and update all information except the employee's salary.

• To combine information from multiple tables for easy access.

In PowerBuilder, you can create single- or multiple-table database views. You can also use a database view when you define data to create a new database view.

You define, open, and manipulate database views in the View painter, which is similar to the Select painter. For more information about the Select painter, see "Selecting a data source" on page 476.

Updating database views
Some database views are logically updatable and others are not. Some DBMSs do not allow any updating of views. For the rules your DBMS follows, see your DBMS documentation.

❖ **To open a database view:**

1 In the Objects view, expand the list of Views for your database.

2 Highlight the view you want to open and select Add To Layout from the pop-up menu, or drag the view's icon to the Object Layout view.

❖ **To create a database view:**

1 Click the Create View button, or select View or New View from the Object>Insert or pop-up menu.

The Select Tables dialog box displays, listing all tables and views that you can access in the database.

2 Select the tables and views from which you will create the view by doing one of the following:

• Click the name of each table or view you want to open in the list displayed in the Select Tables dialog box, then click the Open button to open them. The Select Tables dialog box closes.

- Double-click the name of each table or view you want to open. Each object is opened immediately. Then click the Cancel button to close the Select Tables dialog box.

Representations of the selected tables and views display in the View painter workspace:
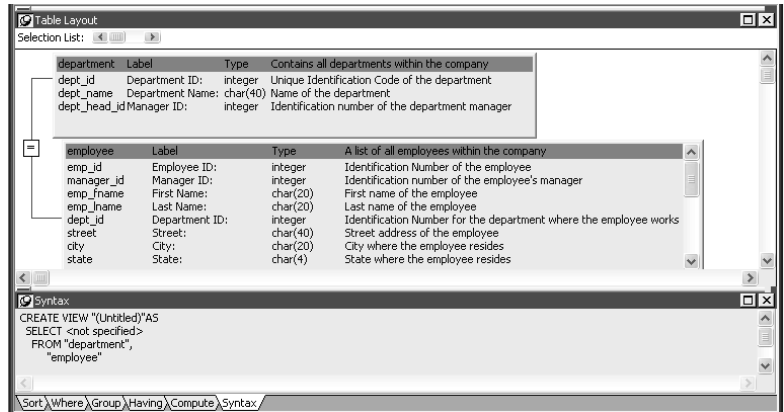
3 Select the columns to include in the view and include computed columns as needed.

4 Join the tables if there is more than one table in the view.

For information, see "Joining tables" on page 420.

5 Specify criteria to limit rows retrieved (Where tab), group retrieved rows (Group tab), and limit the retrieved groups (Having tab), if appropriate.

For information, see the section on using the SQL Select painter in "Selecting a data source" on page 476. The View painter and the SQL Select painter are similar.

6 When you have completed the view, click the Return button.

7 Name the view.

Include *view* or some other identifier in the view's name so that you will be able to distinguish it from a table in the Select Tables dialog box.

8 Click the Create button.

PowerBuilder generates a CREATE VIEW statement and submits it to the DBMS. The view definition is created in the database. You return to the Database painter workspace with the new view displayed in the workspace.

Displaying a database view's SQL statement

You can display the SQL statement that defines a database view. How you do it depends on whether you are creating a new view in the View painter or want to look at the definition of an existing view.

❖ **To display the SQL statement from the View painter:**

• Select the Syntax tab in the View painter.

    PowerBuilder displays the SQL it is generating. The display is updated each time you change the view.

❖ **To display the SQL statement from the Database painter:**

• Highlight the name of the database view in the Objects view and select Properties from the pop-up menu, or drag the view's icon to the Object Details view.

    The completed SELECT statement used to create the database view displays in the Definition field on the General page:
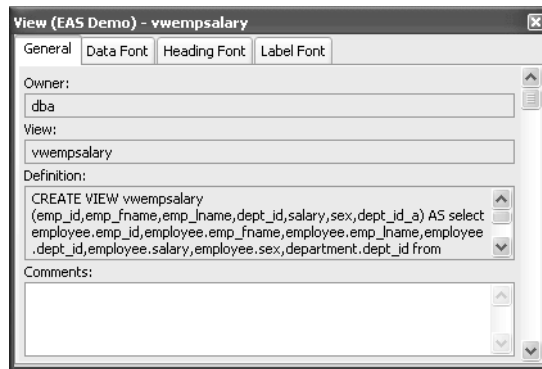
View dialog box is read-only
You cannot alter the view definition in the Object Details view. To alter a view, drop it and create another view.

Joining tables

If the database view contains more than one table, you should join the tables on their common columns. When the View painter is first opened for a database view containing more than one table, PowerBuilder makes its best guess as to the join columns, as follows:

• If there is a primary/foreign key relationship between the tables, PowerBuilder automatically joins them.

• If there are no keys, PowerBuilder tries to join tables based on common column names and types.

❖ **To join tables:**

1   Click the Join button.

2   Click the columns on which you want to join the tables.

In the following screen, the Employee and Department tables are joined on the dept_id column:
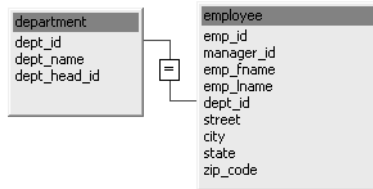
3   To create a join other than the equality join, click the join representation in the workspace.

The Join dialog box displays:
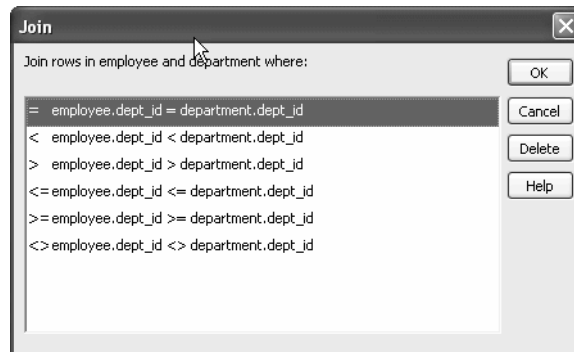
4   Select the join operator you want from the Join dialog box.

If your DBMS supports outer joins, outer join options also display in the Join dialog box. For example, in the preceding dialog box (which uses the Employee and Department tables), you can choose to include rows from the Employee table where there are no matching departments, or rows from the Department table where there are no matching employees.

For more about outer joins, see "Using ANSI outer joins" on page 493.

| Dropping a database view | Dropping a database view removes its definition from the database. |

❖ **To drop a view:**

1   In the Objects view, select the database view you want to drop.

2   Click the Drop Object button or select Drop View from the pop-up menu.

    PowerBuilder prompts you to confirm the drop, then generates a DROP VIEW statement and submits it to the DBMS.

| Exporting view syntax | You can export the syntax for a view to the log. This feature is useful when you want to create a backup definition of the view before you alter it or when you want to create the same view in another DBMS. |

❖ **To export the syntax of an existing view to a log:**

1   Select the view in the painter workspace.

2   Select Export Syntax from the Object menu or the pop-up menu.

    For more information about the log, see "Logging your work" on page 395.

Manipulating data

As you work on the database, you often want to look at existing data or create some data for testing purposes. You might also want to test display formats, validation rules, and edit styles on real data.

PowerBuilder provides data manipulation for such purposes. With data manipulation, you can:

•   Retrieve and manipulate database information

•   Save the contents of the database in a variety of formats (such as Excel, PDF, or XML)

Retrieving data
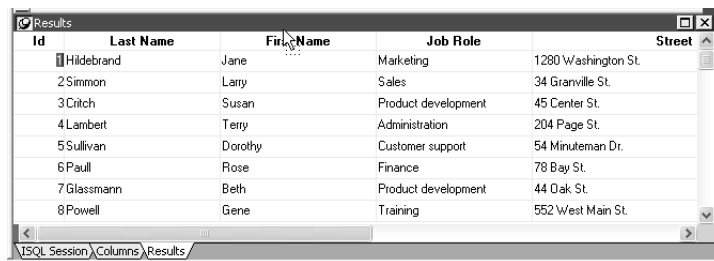
❖   **To retrieve data:**

1   In the Database painter, select the table or database view whose data you want to manipulate.

2   Do one of the following:

- Click one of the three Data Manipulation buttons (Grid, Tabular, or Freeform) in the PainterBar.

- Select Data or Edit Data from the Object or pop-up menu and choose one of the edit options from the cascading menu that displays.

All rows are retrieved and display in the Results view. As the rows are being retrieved, the Retrieve button changes to a Cancel button. You can click the Cancel button to stop the retrieval.

Exactly what you see in the Results view depends on the formatting style you picked. What you see is actually a DataWindow object. The formatting style you picked corresponds to a type of DataWindow object (grid, tabular, or freeform). In a grid display, you can drag the mouse on a column's border to resize the column.

This window is in the grid format:

Only a few rows of data display at a time. You can use the First, Prior, Next, and Last buttons to move from page to page.

Modifying data

You can add, modify, or delete rows. When you have finished manipulating the data, you can apply the changes to the database.

If looking at data from a view
Some views are logically updatable and others are not. Some DBMSs do not allow any updating of views.

For the rules your DBMS follows regarding updating of views, see your DBMS documentation.

❖ **To modify data:**

1   Do one of the following:

- To modify existing data, tab to a field and enter a new value.

- To add a row, click the Insert Row button and enter data in the new row.

- To delete a row, click the delete Row button.

When you add or modify data, the data uses the validation rules, display formats, and edit styles that you or others have defined for the table in the Database painter.

2   Click the Save Changes button or select Rows>Update to apply changes to the database.

Sorting rows

You can sort the data, but any sort criteria you define are for testing only and are not saved with the table or passed to the DataWindow painter.

❖ **To sort the rows:**

1   Select Rows>Sort from the menu bar.

The Specify Sort Columns dialog box displays.

2   Drag the columns you want to sort on from the Source Data box to the
     Columns box:
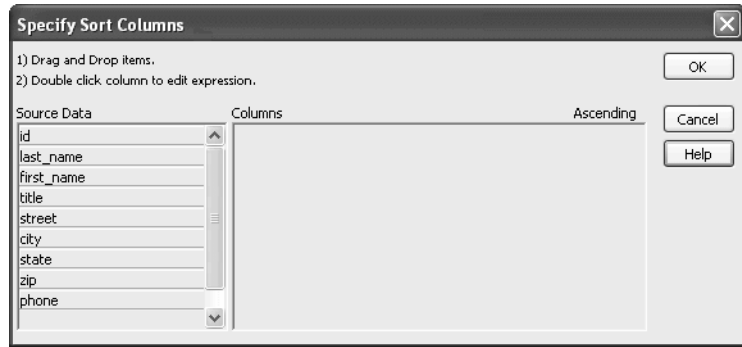
A check box with a check mark in it displays under the Ascending heading
to indicate that the values will be sorted in ascending order. To sort in
descending order, clear the check box.

Precedence of sorting
The order in which the columns display in the Columns box determines the
precedence of the sorting. For example, in the preceding dialog box, rows
would be sorted by department ID. Within department ID, rows would be
sorted by state.

To change the precedence order, drag the column names in the Column
box into the order you want.

---

3   (Optional) Double-click an item in the Columns box to specify an
     expression to sort on.

     The Modify Expression dialog box displays.

4   Specify the expression.

     For example, if you have two columns, Revenues and Expenses, you can
     sort on the expression Revenues – Expenses.

5   Click OK to return to the Specify Sort Columns dialog box with the
     expression displayed.

If you change your mind
You can remove a column or expression from the sorting specification by
simply dragging it and releasing it outside the Columns box.

---

6   When you have specified all the sort columns and expressions, click OK.

Filtering rows

You can limit which rows are displayed by defining a filter.

The filters you define are for testing only and are not saved with the table or passed to the DataWindow painter.

❖ **To filter the rows:**

1   Select Rows>Filter from the menu bar.

    The Specify Filter dialog box displays.

2   Enter a boolean expression that PowerBuilder will test against each row:

    If the expression evaluates to TRUE, the row is displayed. You can paste functions, columns, and operators in the expression.

3   Click OK.

    PowerBuilder filters the data. Only rows meeting the filter criteria are displayed.

❖ **To remove the filter:**

1   Select Rows>Filter from the menu bar.

    The Specify Filter dialog box displays, showing the current filter.

2   Delete the filter expression, then click OK.

Filtered rows and updates
Filtered rows are updated when you update the database.

Viewing row information

You can display information about the data you have retrieved.

❖ **To display row information:**

• Select Rows>Described from the menu bar.

The Describe Rows dialog box displays showing the number of:

• Rows that have been deleted in the painter but not yet deleted from the database

• Rows displayed in Preview

• Rows that have been filtered

• Rows that have been modified in the painter but not yet modified in the database

All row counts are zero until you retrieve the data from the database or add a new row. The count changes when you modify the displayed data or test filter criteria.

Importing data

You can import data from an external source and then save the imported data in the database.

❖ **To import data:**

1 Select Rows>Import from the menu bar.

The Select Import File dialog box displays.

2 Specify the file from which you want to import the data.

The types of files you can import into the painter are shown in the Files of Type drop-down list.

3 Click Open.

PowerBuilder reads the data from the file. You can click the Save Changes button or select Rows>Update to add the new rows to the database.

Printing data

You can print the data displayed by selecting File>Print from the menu bar. Before printing, you can also preview the output on the screen.

❖ **To preview printed output before printing:**

1 Select File>Print Preview from the menu bar.

Preview displays the data as it will print. To display rulers around the page borders in Print Preview, select File>Print Preview Rulers.

2 To change the magnification used in Print Preview, select File>Print Preview Zoom from the menu bar.

The Zoom dialog box displays.

3 Select the magnification you want and click OK.

Preview zooms in or out as appropriate.

4 When you have finished looking at the print layout, select File>Print Preview from the menu bar again.

Saving data

You can save the displayed data in an external file.

❖ **To save the data in an external file:**

1 Select File>Save Rows As from the menu bar.

The Save Rows As dialog box displays.

2 Choose a format for the file.

You can select from several formats, including Powersoft report (PSR), XML, and HTML.

If you want the column headers saved in the file, select a file format that includes headers, such as Excel With Headers. When you select a *with headers* format, the names of the database columns (not the column labels) will also be saved in the file.

For more information, see "Saving data in an external file" on page 533.

3   For TEXT, CSV, SQL, HTML, and DIF formats, select an encoding for the file.

You can select ANSI/DBCS, Unicode LE (Little-Endian), Unicode BE (Big-Endian), or UTF8.

4   Name the file and save it.

PowerBuilder saves all displayed rows in the file; all columns in the displayed rows are saved. Filtered rows are not saved.

Creating and executing SQL statements

The Database painter's Interactive SQL view is a SQL editor in which you can enter and execute SQL statements. The view provides all editing capabilities needed for writing and modifying SQL statements. You can cut, copy, and paste text; search for and replace text; and paint SQL statements. You can also set editing properties to make reading your SQL files easier.

Building and executing SQL statements

You can use the Interactive SQL view to build SQL statements and execute them immediately. The view acts as a notepad in which you can enter SQL statements.

Creating stored procedures

You can use the Interactive SQL view to create stored procedures or triggers, but make sure that the painter's SQL statement terminator character is not the same as the terminator character used in the stored procedure language of your DBMS.

About the statement terminator   By default, PowerBuilder uses the semicolon as the SQL statement terminator. You can override the semicolon by specifying a different terminator character in the Database painter. To change the terminator character, select Design>Options from the Database painter's menu bar.

Make sure that the character you choose is not reserved for another use by your database vendor. For example, using the slash character (/) causes compilation errors with some DBMSs.

Controlling comments

By default, PowerBuilder strips off comments when it sends SQL to the DBMS. You can have comments included by clearing the check mark next to Strip Comments in the pop-up menu of the Interactive SQL view.

Entering SQL

You can enter a SQL statement in four ways:

•   Pasting the statement

•   Typing the statement in the view

•   Opening a text file containing the SQL

•   Dragging a procedure or function from the Objects view

Pasting SQL   You can paste SELECT, INSERT, UPDATE, and DELETE statements to the view. Depending on which kind of statement you want to paste, PowerBuilder displays dialog boxes that guide you through painting the full statement.

❖ **To paste a SQL statement to the workspace:**

1   Click the Paste SQL button in the PainterBar, or select Paste Special>SQL from the Edit or pop-up menu.

    The SQL Statement Type dialog box displays, listing the types of SQL statements you can use.

2   Double-click the appropriate icon to select the statement type.

    The Select Table dialog box displays.

3   Select the table(s) you will reference in the SQL statement.

    You go to the Select, Insert, Update, or Delete painter, depending on the type of SQL statement you are pasting. The Insert, Update, and Delete painters are similar to the Select painter, but only the appropriate tabs display in the SQL toolbox at the bottom of the workspace.

    For more information about the Select painter, see "Selecting a data source" on page 476.

4   Do one of the following:

• For a SELECT statement, define the statement exactly as in the Select painter when building a view.

You choose the columns to select. You can define computed columns, specify sorting and joining criteria, and WHERE, GROUP BY, and HAVING criteria. For more information, see "Working with database views" on page 417.

• For an INSERT statement, type the values to insert into each column. You can insert as many rows as you want.

• For an UPDATE statement, specify the new values for the columns in the Update Column Values dialog box. Then specify the WHERE criteria to indicate which rows to update.

• For a DELETE statement, specify the WHERE criteria to indicate which rows to delete.

5   When you have finished painting the SQL statement, click the Return button in the PainterBar in the Select, Insert, Update, or Delete painter.

You return to the Database painter with the SQL statement pasted into the ISQL view.

Typing SQL

Rather than paste, you can simply type one or more SQL statements directly in the ISQL view.

You can enter most statements supported by your DBMS, including statements you can paint as well as statements you cannot paint, such as a database stored procedure or CREATE TRIGGER statement. You cannot enter certain statements that could destabilize the PowerBuilder development environment. These include the SET statement and the USE *database* statement.

Sybase Adaptive Server Enterprise stored procedures
When you use the Database painter to execute a Sybase Adaptive Server Enterprise system stored procedure, you *must* start the syntax with the keyword EXEC or EXECUTE. For example, enter EXEC SP_LOCK. You cannot execute the stored procedure simply by entering its name.

Importing SQL from a text file

You can import SQL that has been saved in a text file into the Database painter.

❖ **To read SQL from a file:**

1   Put the insertion point where you want to insert the SQL.

2   Select Paste Special>From File from the Edit or pop-up menu.

3   Select the file containing the SQL, and click OK.

Dragging a procedure or function from the Objects view

From the tree view in the Objects view, you can select an existing procedure or function that contains a SQL statement you want to enter, and drag it to the Interactive SQL view.

Explaining SQL

Sometimes there is more than one way to code SQL statements to obtain the results you want. When this is the case, you can use Explain SQL on the Design menu to help you select the most efficient method. Explain SQL displays information about the path that PowerBuilder will use to execute the statements in the SQL Statement Execution Plan dialog box. This is most useful when you are retrieving or updating data in an indexed column or using multiple tables.

DBMS-specific information
The information displayed in the SQL Statement Execution Plan dialog box depends on your DBMS. For more about the SQL execution plan, see your DBMS documentation.

Executing SQL

When you have the SQL statements you want in the workspace, you can submit them to the DBMS.

❖ **To execute the SQL:**

•   Click the Execute button, or select Design>Execute SQL from the menu bar.

If the SQL retrieves data, the data appears in grid format in the Results view. If there is a database error, you see a message box describing the problem.

For a description of what you can do with the data, see "Manipulating data" on page 422.

Customizing the editor

The Interactive SQL view provides the same editing capabilities as the file editor. It also has Script, Font, and Coloring properties that you can change to make SQL files easier to read. With no change in properties, SQL files have black text on a white background and a tab stop setting of 3 for indentation.

Setting Script and Font properties

Select Design>Options from the menu bar to open the Database Preferences dialog box. The Script and Font properties are the same as those you can set for the file editor.

For more information, see "Using the file editor" on page 32.

Editor properties apply elsewhere
When you set Script and Font properties for the Database painter, the settings also apply to the Script view, the file editor, and the Debug window.

Setting Coloring properties

You can set the text color and background color for SQL styles (such as datatypes and keywords) so that the style will stand out and the SQL code will be more readable. You set Coloring properties on the Coloring tab page.

Enabling syntax coloring
Be sure the Enable Syntax Coloring check box is selected before you set colors for SQL styles. You can turn off all Coloring properties by clearing the check box.

Controlling access to the current database

The Database painter's Design menu provides access to a series of dialog boxes you can use to control access to the current database. In some DBMSs, for example, you can assign table access privileges to users and groups.

Which menu items display on the Design menu and which dialog boxes display depend on your DBMS.

For information about support for security options in your DBMS, see *Connecting to Your Database* and your DBMS documentation.

Adding objects for MobiLink synchronization

About MobiLink

MobiLink is a session-based synchronization system that allows two-way synchronization between a main database, called the consolidated database, and multiple remote databases. You use the MobiLink Synchronization for ASA wizard on the Database tab of the New dialog box to create objects that make it easy to control database synchronization from a PowerBuilder application.

This section describes the MobiLink synchronization wizard and the objects it creates. For more detailed information about synchronization from PowerBuilder applications, see the chapter on MobiLink synchronization in *Application Techniques*.

What the wizard generates

You use the MobiLink synchronization wizard to create a nonvisual user object, a global external function, and a structure for holding database connection parameters. These objects make it easier to add database synchronization capabilities to a PowerBuilder target. By default, the wizard also adds two windows and a second global function, but these objects are optional.

Table 16-7 shows objects that can be generated by the wizard, listed by their default names, where *appname* stands for the name of the current application.

Table 16-7: Objects generated by MobiLink Synchronization wizard

Default name	Description
nvo_*appname*_sync	Nonvisual user object that starts synchronization from the remote client.
gf_*appname*_sync	Global function that instantiates nvo_*appname*_sync to start the synchronization. This function includes the logic to start the synchronization with or without a feedback window.
s_*appname*_sync_parms	Global structure used to store runtime information, such as connection parameters entered by the user in the w_*appname*_sync_options window.
w_*appname*_sync or w_sync_default_feedback	Optional feedback window that can be used to display synchronization status to the client instead of the standard MobiLink status window. Even if you do not use the optional status window, a generated feedback window must be instantiated by the gf_*appname*_sync function.

Default name	Description
gf_*appname*_configure_sync	Optional global function that calls the w_*appname*_sync_options window, which allows the user to configure the dbmlsync client.
w_*appname*_sync_options	Window that allows the application user to change connection arguments at runtime.

Using a desktop database profile

Some information that you enter in the wizard is optional, but other information is required. The wizard prompts you for a database profile, which it uses to establish a connection to a remote database on the desktop. If you are not testing a connection on the desktop, you can select the option to proceed without a database connection and ignore the database profile field.

A database profile is required for automatic retrieval of publication names in the database. A publication is a database object describing data to be synchronized. A publication, along with a synchronization user name and a synchronization subscription, is required for MobiLink synchronization.

Selecting publication names

The wizard lets you select multiple publication names if they exist in the remote database defined by the connection profile. There must be subscriptions associated with the publication in order for them to display in the publication selection list.

If you selected the option to proceed without a database connection, the wizard prompts you to type a publication name (or a comma-separated list of publication names) in the MobiLink Client Publication wizard page instead of prompting you to select publication names retrieved from the database.

For more information about publications, see the *MobiLink Synchronization User's Guide* on the Technical Library CD or the SQL Anywhere Studio online Help.

Overriding registry settings on the client machine

By default, information you enter in the wizard is saved in instance variables in the nvo_*appname*_sync user object that the wizard generates. This information includes values that you select for MobiLink logging and command line options and the MobiLink server and port. Prior to synchronization, the values of these instance variables can be modified with values entered by an application user in the nvo_*appname*_sync_options Options window. The first time synchronization is run, values of the instance variables are entered into the client machine registry. The next time the application is run, this information is available for retrieval from the registry.

The MobiLink Synchronization for ASA wizard has an optional Override Registry Settings screen that allows you to override client registry settings. When you enable runtime overrides to the client registry settings, you must assign a build number to the objects generated by the wizard.

The build number you assign can be any positive numeric value. To override the registry settings, the build number you assign must be higher than the build number in the registry, if there is one. Registry settings will be used if the build number in the registry is equal to or lower than the build number in the is_sync_object_bld instance variable of the nvo_*appname*_sync user object that the wizard generates.

The Override Registry Settings screen of the wizard displays only if you do not clear the default option (on the Optional Runtime Configuration Objects screen) to prompt the application user for password and runtime changes. If, however, you select the option to disallow runtime overrides to the synchronization, the wizard does not display the Override Registry Settings screen and does not generate the w_*appname*_sync_options Options window.

Wizard options

MobiLink Synchronization for ASA wizard options are listed in Table 16-8.

Table 16-8: MobiLink Synchronization for ASA wizard options

Option	Description
Destination library	Lets you select the target PBL file where you want to generate the MobiLink synchronization objects.
Desktop database connection	Lets you select a PowerBuilder database profile or proceed without a database connection.
Publication name	Lets you select a publication (or multiple publications) if you specified a database profile for a desktop database connection. If you did not, you can type the name of a publication you want to synchronize.
Override registry settings	Lets you override client registry settings with values that you (or the application user) select for MobiLink logging and command line options, and the MobiLink server and port for the application
Client logging options	Specifies what information gets written to the synchronization log and whether you save the information to a log file.
Additional command line options	Adds the options you specify to the command line for starting the MobiLink synchronization client. You can click the Usage button to see a list of valid options.

Option	Description
Extended options	Adds extended options you specify. You do not need to enter the "-e" switch for extended options in this field. You can click the Usage button to see a list of valid extended options. Single quotes must be used for any extended option values requiring quotation marks. You must separate multiple options with semicolons; for example: `scn=on;adr='host=localhost;port=2439'`
Host	Sets the host information for connecting to the MobiLink Synchronization Server. If you enter a value for this field, it overrides any value set in synchronization subscriptions and in the Extended Options field.
Port	Sets the port for connecting to the MobiLink Synchronization Server. The default port for MobiLink is 2439. The value you enter for this field overrides any value set in synchronization subscriptions and in the Extended Options field.

For more information about a wizard option, click inside the option field and press F1 for online Help.

❖ **To add objects for MobiLink synchronization**

1 Select File>New from the PowerBuilder menu bar.

2 Click the Database tab, select the MobiLink Synchronization for ASA wizard, and click OK.

3 Follow the instructions in the wizard, providing the information the wizard needs.

On the last page of the wizard, make sure the Generate To-Do List check box is selected if you want the wizard to add items to the To-Do List to guide and facilitate your development work.

4 When you are satisfied with your choices in the wizard, click Finish.

The wizard generates objects that you can use for database synchronization.

For information about using the generated objects to synchronize a target database, see the chapter on MobiLink synchronization in *Application Techniques*.

Managing MobiLink synchronization on the server

You can start the MobiLink synchronization server and Sybase Central from the PowerBuilder UI.

Starting the MobiLink synchronization server

Before you synchronize remote databases with the consolidated database, you must start the MobiLink synchronization server. You can start the server from the Database or the Database Profile painter in PowerBuilder.

❖ **To start the MobiLink synchronization server:**

1   From the Objects view of the Database painter or from the Database Profile painter, expand the ODBC Utilities folder and click MobiLink Synchronization server.

The MobiLink Synchronization Server Options dialog box displays.

2   Select the MobiLink version and enter the ODBC connection string for your consolidated database.

The values that populate the MobiLink version drop-down list come from the Adaptive Server Anywhere versions listed in the *hkey_local_machine\software\odbc\odbcinst.ini* registry key.

The ODBC connection string should not contain any blank spaces that are not part of the data source name. The following is an example of an ODBC connection string for the ASA :

```
DSN=ASA 9.0 Sample;UID=dba;PWD=sql
```

3   Define other options as needed.

For information about filling in specific fields in the dialog box, click the Help button in the dialog box. The Usage button opens a dialog box with information about command line options.

4   Click OK.

When you click OK, PowerBuilder starts the MobiLink Synchronization server.

Using Sybase Central

You can use Sybase Central to manage MobiLink synchronization and create synchronization scripts that are held in the consolidated database. You can also use the ASA plug-in to Sybase Central to add publications, synchronization users, and synchronization subscriptions to remote databases.

❖ **To start Sybase Central**

• From the Objects view of the Database painter or from the Database Profile painter, expand the ODBC Utilities folder, and click Sybase Central.

Sybase Central displays.

❖ **To work with the consolidated database in Sybase Central**

• Right-click MobiLink Synchronization in the left pane of Sybase Central, select Connect from the pop-up menu, enter connection parameters in the Connect dialog box, and click OK.

You can use Sybase Central to add scripts for database tables and select synchronization events that cause the script to be executed.

❖ **To work with remote databases in Sybase Central**

• Right-click Adaptive Server Anywhere in the left pane of Sybase Central, select Connect from the pop-up menu, enter connection parameters in the Connect dialog box, and click OK.

If you open the Publications and MobiLink Users folders under the MobiLink Synchronization Client folder for the remote database, you can add publications and synchronization users in the right pane of Sybase Central.

After you add a publication and a synchronization user, you can create a synchronization subscription by right-clicking the publication in the right pane of Sybase Central, selecting Properties from the pop-up menu, clicking the Subscribe button on the Synchronization Subscriptions tab of the Publication Properties dialog box, and clicking OK.

For more information, see the chapter on MobiLink synchronization in the *Application Techniques* and the SQL Anywhere Studio online Help. You can open the online Help by clicking the Help button in dialog boxes of the ASA or MobiLink plug-in for Sybase Central.

Working with Data Pipelines

About this chapter

This chapter describes how to use the Data Pipeline painter to create data pipelines, which let you reproduce database data in various ways.

Contents

About data pipelines

The Data Pipeline painter gives you the ability to reproduce data quickly within a database, across databases, or even across DBMSs. To do that, you create a data pipeline which, when executed, pipes the data as specified in the definition of the data pipeline.

What you can do

With the Data Pipeline painter, you can perform some tasks that would otherwise be very time consuming. For example, you can:

- Pipe data (and extended attributes) from one or more tables to a table in the same DBMS or a different DBMS

- Pipe an entire database, a table at a time, to another DBMS (and if needed, pipe the database's extended attribute system tables)

- Create a table with the same design as an existing table but with no data

- Pipe corporate data from a database server to an Adaptive Server Anywhere database on your computer so you can work on the data and report on it without needing access to the network

- Upload local data that changes daily to a corporate database

- Create a new table when a change (such as allowing or disallowing NULLs or changing primary key or index assignments) is disallowed in the Database painter

---

Piping data in applications
You can also create applications that pipe data. For more information, see *Application Techniques*.

Source and destination databases

You can use the Data Pipeline painter to pipe data from one or more tables in a source database to one table in a destination database.

You can pipe all data or selected data in one or more tables. For example, you can pipe a few columns of data from one table or data selected from a multitable join. You can also pipe from a view or a stored procedure result set to a table.

When you pipe data, the data in the source database remains in the source database and is reproduced in a new or existing table in the destination database.

Although the source and destination can be the same database, they are usually different ones, and they can even have different DBMSs. For example, you can pipe data from an Adaptive Server Enterprise database to an Adaptive Server Anywhere database on your computer.

Defining a data pipeline

When you use the Data Pipeline painter to create a pipeline, you define the:

- Source database

- Destination database

- Source of data

- Pipeline operation

- Destination table

After you create a pipeline, you can execute it immediately. If you want, you can also save it as a named object to use and reuse. Saving a pipeline enables you to pipe the data that might have changed since the last pipeline execution or to pipe the data to other databases later.

Datatype support

Each DBMS supports certain datatypes. When you pipe data from one DBMS to another, PowerBuilder makes a best guess at the appropriate destination datatypes. You can correct PowerBuilder's best guess in your pipeline definition as needed.

The Data Pipeline painter supports the piping of columns of any datatype, including columns with blob data. For information about piping a column that has a blob datatype, see "Piping blob data" on page 453.

Piping extended attributes

The first time PowerBuilder connects to a database, it creates five system tables called the extended attribute system tables. These system tables initially contain default extended attribute information for tables and columns. In PowerBuilder, you can create extended attribute definitions such as column headers and labels, edit styles, display formats, and validation rules.

For more information about the extended attribute system tables, see the Appendix, "The Extended Attribute System Tables."

Piping extended
attributes
automatically

When you pipe data, you can specify that you want to pipe the extended attributes associated with the columns you are piping. You do this by selecting the Extended Attributes check box in the Data Pipeline painter workspace:
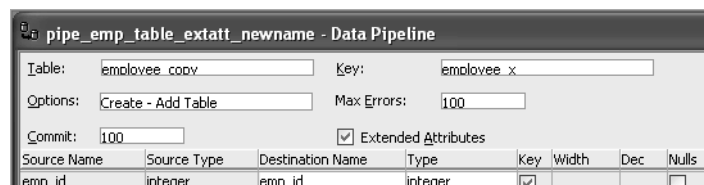
When the Extended Attributes check box is selected, the extended attributes associated with the source database's selected columns automatically go into the extended attribute system tables of the destination database, with one exception. When you pipe a column that has an edit style, display format, or validation rule associated with it, the style, rule, or format is not piped if one with the same name exists in the extended attribute system tables of the destination database. In this situation, the column uses the style, rule, or format already present in the destination database.

For example, for the Phone column in the Employee table, the display format with the name Phone_format would be piped unless a display format with the name Phone_format already exists in the destination database. If such a display format exists, the Phone column would use the Phone_format display format in the destination database.

Piping the extended attribute system tables

Selecting the Extended Attributes check box never results in the piping of named display formats, edit styles, and validation rules that are stored in the extended attribute system tables but are not associated with columns in tables you are piping. If you want such extended attribute definitions from one database to exist in another database, you can pipe the appropriate extended attribute system table or a selected row or rows from the table.

Piping an entire database

If you want to reproduce an entire database, you can pipe all database tables and extended attribute system tables, one table at a time.

Creating a data pipeline

You have a number of choices when creating a data pipeline. This section leads you through them.

❖ **To create a data pipeline:**

1 Click the New button in the PowerBar and then select the Database tab page.

2 Select Data Pipeline and click OK.

The New Data Pipeline dialog box displays.
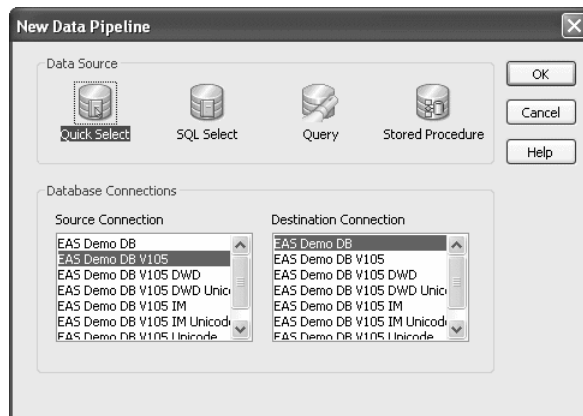
The Source Connection and Destination Connection boxes display database profiles that have been defined. The last database you connected to is selected as the source. The first database on the destination list is selected as the destination.

If you do not see the connections you need
To create a pipeline, the databases you want to use for your source and destination must each have a database profile defined. If you do not see profiles for the databases you want to use, select Cancel in the New Data Pipeline dialog box and then define those profiles. For information about defining profiles, see "Changing the destination and source databases" on page 455.

---

3   Select a data source.

The data source determines how PowerBuilder retrieves data when you execute a pipeline:

Data source	Use it if
Quick Select	The data is from tables that have a primary/foreign key relationship and you need only to sort and limit data
SQL Select	You want more control over the SQL SELECT statement generated for the data source or your data is from tables that are not connected through a key
Query	The data has been defined as a query
Stored Procedure	The data is defined in a stored procedure

4   Select the source and destination connections and click OK.

5   Define the data to pipe.

How you do this depends on what data source you chose in step 3, and is similar to the process used to define a data source for a DataWindow object. For complete information about using each data source and defining the data, see Chapter 18, "Defining DataWindow Objects."

When you finish defining the data to pipe, the Data Pipeline painter workspace displays the pipeline definition, which includes a pipeline operation, a check box for specifying whether to pipe extended attributes, and source and destination items.
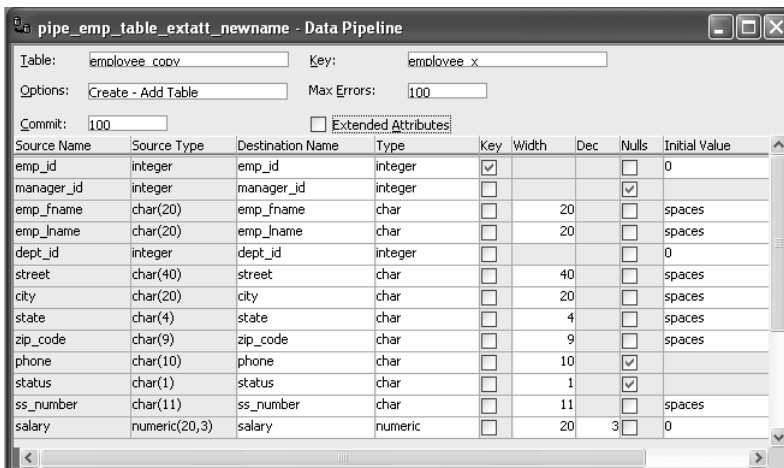
The pipeline definition is PowerBuilder's best guess based on the source data you specified.

6   Modify the pipeline definition as needed.

For information, see "Modifying the data pipeline definition" next.

7   (Optional) Modify the source data as needed. To do so, click the Data button in the PainterBar, or select Design>Edit Data Source from the menu bar.

For information about working in the Select painter, see Chapter 18, "Defining DataWindow Objects."

When you return to the Data Pipeline painter workspace, PowerBuilder reminds you that the pipeline definition will change. Click OK to accept the definition change.

8   If you want to try the pipeline now, click the Execute button or select Design>Execute from the menu bar.

PowerBuilder retrieves the source data and executes the pipeline. If you specified retrieval arguments in the Select painter, PowerBuilder first prompts you to supply them.

At runtime, the number of rows read and written, the elapsed execution time, and the number of errors display in MicroHelp. You can stop execution yourself or PowerBuilder might stop execution if errors occur.

For information about execution and how rows are committed to the destination table, see "When execution stops" on page 451.

9   Save the pipeline definition if appropriate.

For information, see "Saving a pipeline" on page 457.

Seeing the results of piping data
You can see the results of piping data by connecting to the destination database and opening the destination table.

Modifying the data pipeline definition

After you create a pipeline definition, you can modify it in a variety of ways. The changes you make depend on what pipeline operation you select, the destination DBMS, and what you are trying to accomplish by executing the pipeline.

Table 17-1 lists properties you can modify that apply to the destination table. These properties display at the top of the Data Pipeline painter workspace.

Table 17-1: Pipeline properties for the destination table

Item	Description	Default	How to edit
Table	Name of the destination table.	If source and destination are different, name of first table specified in the source data or name of the stored procedure. If the same, _copy is appended.	For Create or Replace, enter a name. For Refresh, Append, or Update, select a name from the drop-down list.
Options	Pipeline operation: Create, Replace, Refresh, Append, or Update.	Create - Add Table.	Select an option from the drop-down list. See Table 17-3 on page 449.
Commit	Number of rows piped to the destination database before PowerBuilder commits the rows to the database.	100 rows.	Select a number, All, or None from the drop-down list.
Key	Key name for the table in the destination database.	If the source is only one table, the table name is followed by _x.	(Create or Replace only) Enter a name.
Max Errors	Number of errors allowed before the pipeline stops.	100 errors.	Select a number or No Limit from the drop-down list.
Extended Attributes	(Create and Replace only) Specifies whether or not the extended attributes of the selected source columns are piped to the extended attribute system tables of the destination database.	Not checked.	Click the check box.

Table 17-2 lists properties that you can modify that apply to the destination table's columns and keys. These properties display under the properties that apply to the table itself and most can be modified only for the Create and Replace pipeline operations.

Column names and datatypes that cannot be modified
You cannot modify the source column names and datatypes that display at the left of the workspace.

Table 17-2: Pipeline properties for the destination table's columns and keys

Item	Description	Default	How to edit
Destination Name	Column name	Source column name.	Enter a name.
Type	Column datatype	If the DBMS is unchanged, source column datatype. If the DBMS is different, a best-guess datatype.	Select a type from the drop-down list.
Key	Whether the column is a key column (check means yes)	Source table's key columns (if the source is only one table and all key columns were selected).	Select or clear check boxes.
Width	Column width	Source column width.	Enter a number.
Dec	Decimal places for the column	Source column decimal places.	Enter a number.
Nulls	Whether NULL is allowed for the column (check means yes)	Source column value.	Select or clear check boxes.
Initial Value	Column initial value	Source column initial value. (If no initial value, character columns default to spaces and numeric columns default to 0.)	Select an initial value from the drop-down list.
Default Value	Column default value	None. Default values stored in the source database are not piped to the destination database.	Select a default value from the drop-down list or enter a default value. Keyword values depend on destination DBMS.

Choosing a pipeline operation

When PowerBuilder pipes data, what happens in the destination database depends on which pipeline operation you choose in the Options drop-down list at the top of the workspace.

Table 17-3: Effect of pipeline operations on the destination database

Pipeline operation	Effect on destination database
Create - Add Table	A new table is created and rows selected from the source tables are inserted. If a table with the specified name already exists in the destination database, a message displays and you must select another option or change the table name.
Replace - Drop/Add Table	An existing table with the specified table name is dropped, a new table is created, and rows selected from the source tables are inserted. If no table exists with the specified name, a table is created.
Refresh - Delete/Insert Rows	All rows of data in an existing table are deleted, and rows selected from the source tables are inserted.
Append - Insert Rows	All rows of data in an existing table are preserved, and new rows selected from the source tables are inserted.
Update - Update/Insert Rows	Rows in an existing table that match the key criteria values in the rows selected from the source tables are updated, and rows that do not match the key criteria values are inserted.

Dependency of modifications on pipeline operation

The modifications you can make in the workspace depend on the pipeline operation you have chosen.

When using
Create or Replace

When you select the Create - Add Table option (the default) or the Replace - Drop/Add Table option, you can:

• Change the destination table definition.

Follow the rules of the destination DBMS.

- Specify or clear a key name and/or key columns.

   Specify key columns by selecting one or more check boxes to define a unique identifier for rows. Neither a key name nor key columns are required.

- Allow or disallow NULLs for a column.

   If NULL is allowed, no initial value is allowed. If NULL is not allowed, an initial value is required. The words spaces (a string filled with spaces) and today (today's date) are initial value keywords.

- Modify the Commit and Max Errors values.

- Specify an initial value and a default value.

If you have specified key columns and a key name and if the destination DBMS supports primary keys, the Data Pipeline painter creates a primary key for the destination table. If the destination DBMS does not support primary keys, a unique index is created.

For Oracle databases
PowerBuilder generates a unique index for Oracle databases.

If you try to use the Create option, but a table with the specified name already exists in the destination database, PowerBuilder tells you, and you must select another option or change the table name.

When you use the Replace option, PowerBuilder warns you that you are deleting a table, and you can choose another option if needed.

When using Refresh and Append

For the Refresh - Delete/Insert Rows or Append - Insert Rows options, the destination table must already exist. You can:

- Select an existing table from the Table drop-down list.

- Modify the Commit and Max Errors values.

- Change the initial value for a column.

When using Update

For the Update - Update/Insert Rows option, the destination table must already exist. You can:

- Select an existing table from the Table drop-down list.

- Modify the Commit and Max Errors values.

- Change the Key columns in the destination table's primary key or unique index, depending on what the DBMS supports. Key columns must be selected; the key determines the UPDATE statement's WHERE clause.

- Change the initial value for a column.

Bind variables and the Update option
If the destination database supports bind variables, the Update option takes advantage of them to optimize pipeline execution.

When execution stops

Execution of a pipeline can stop for any of these reasons:

- You click the Cancel button

  During the execution of a pipeline, the Execute button in the PainterBar changes to a Cancel button.

- The error limit is reached

If there are rows that cannot be piped to the destination table for some reason, those error rows display once execution stops. You can correct error rows or return to the workspace to change the pipeline definition and then execute it again. For information, see "Correcting pipeline errors" on page 456.

Whether rows are committed

When rows are piped to the destination table, they are first inserted and then either committed or rolled back. Whether rows are committed depends on:

- What the Commit and Max Errors values are

- When errors occur during execution

- Whether you click the Cancel button or PowerBuilder stops execution

When you stop execution	When you click Cancel, if the Commit value is a number, every row that was piped is committed. If the Commit value is All or None, every row that was piped is rolled back.

For example, if you click the Cancel button when the 24th row is piped and the Commit value is 20, then:

1   20 rows are piped and committed.

2   3 rows are piped and committed.

3   Piping stops.

If the Commit value is All or None, 23 rows are rolled back.

When PowerBuilder stops execution	PowerBuilder stops execution if the error limit is reached. Table 17-4 shows how the Commit and Max Errors values affect the number of rows that are piped and committed.

Table 17-4: Rows committed when PowerBuilder stops execution

Commit value	Max Errors value	Result
A number *n*	No limit or a number *m*	Rows are piped and committed *n* rows at a time until the Max Errors value is reached.
All or None	No limit	Every row that pipes without error is committed.
All or None	A number *n*	If the number of errors is less than *n*, all rows are committed. If the number of errors is equal to *n*, every row that was piped is rolled back. No changes are made.

For example, if an error occurs when the 24th row is piped and the Commit value is 10 and the Max Errors value is 1, then:

1   10 rows are piped and committed.

2   10 rows are piped and committed.

3   3 rows are piped and committed.

4   Piping stops.

If the Commit value is All or None, 23 rows are rolled back.

About transactions	A transaction is a logical unit of work done by a DBMS, within which either all the work in the unit must be completed or none of the work in the unit must be completed. If the destination DBMS does not support transactions or is not in the scope of a transaction, each row that is inserted or updated is committed.

About the All and
None commit values

In the Data Pipeline painter, the Commit values All and None have the same meaning.

The None commit value is most useful at runtime. For example, some PowerBuilder applications require either all piped rows to be committed or no piped rows to be committed if an error occurs. Specifying None allows the application to control the committing and rolling back of piped rows by means of explicit transaction processing, such as the issuing of commits and rollbacks in pipeline scripts using COMMIT and ROLLBACK statements.

Piping blob data

Blob data is data that is a *b*inary *l*arge-*ob*ject such as a Microsoft Word document or an Excel spreadsheet. A data pipeline can pipe columns containing blob data.

The name of the datatype that supports blob data varies by DBMS. Table 17-4 shows some examples.

Table 17-5: Examples of datatypes that support blob data

DBMS	Data types that support blob data
Sybase Adaptive Server Anywhere	LONG BINARY, LONG VARCHAR (if more than 32 KB)
Sybase Adaptive Server Enterprise	IMAGE, TEXT
Microsoft SQL Server	IMAGETEXT
Oracle	RAW, LONG RAW
Informix	BYTE, TEXT

For information about the datatype that supports blob data in your DBMS, see your DBMS documentation.

Adding blob columns
to a pipeline definition

When you select data to pipe, you cannot select a blob column as part of the data source because blobs cannot be handled in a SELECT statement. After the pipeline definition is created, you add blob columns, one at a time, to the definition.

❖ **To add a blob column to a pipeline definition:**

1   Select Design>Database Blob from the menu bar.

If the Database Blob menu item is disabled
The Database Blob menu item is disabled if the pipeline definition does not contain a unique key for at least one source table, or if the pipeline operation is Refresh, Append, or Update and the destination table has no blob columns.

The Database Binary/Text Large Object dialog box displays. The Table box has a drop-down list of tables in the pipeline source that have a primary key and contain blob columns.

2   In the Table box, select the table that contains the blob column you want to add to the pipeline definition.

For example, in the EAS Demo DB, the ole table contains a blob column named Object with the large binary datatype.

3   In the Large Binary/Text Column box, select a column that has a blob datatype.

4   In the Destination Column box, change the name of the destination column for the blob if you want to.

If you want to add the column and see changes you make without closing the dialog box, click Apply after each change.

5   When you have specified the blob source and destination as needed, click OK.

❖ **To edit the source or destination name of the blob column in the pipeline definition:**

•   Display the blob column's pop-up menu and select Properties.

❖ **To delete a blob column from the pipeline definition:**

•   Display the blob column's pop-up menu and select Clear.

Executing a pipeline
with blob columns

After you have completed the pipeline definition by adding one or more blob columns, you can execute the pipeline. When you do, rows are piped a block at a time, depending on the Commit value. For a given block, Row 1 is inserted, then Row 1 is updated with Blob 1, then Row 1 is updated with Blob 2, and so on. Then Row 2 is inserted, and so on until the block is complete.

If a row is not successfully piped, the blob is not piped. Blob errors display, but the blob itself does not display. When you correct a row and execute the pipeline, the pipeline pipes the blob.

Changing the destination and source databases

Changing the
destination

When you create a pipeline, you can change the destination database. If you want to pipe the same data to more than one destination, you can change the destination database again and re-execute.

❖   **To change the destination database:**

•   Click the Destination button in the PainterBar, or select File>Destination Connect from the menu bar.

Changing the source

Normally you would not change the source database, because your pipeline definition is dependent on it, but if you need to (perhaps because you are no longer connected to that source), you can.

❖   **To change the source database:**

•   Select File>Source Connect from the menu bar.

Source changes when active profile changes
When you open a pipeline in the Data Pipeline painter, the source database becomes the active connection. If you change the active connection in the Database painter when the Data Pipeline painter is open, the source database in the Data Pipeline painter changes to the new active connection automatically.

Working with
database profiles

At any time in the Data Pipeline painter, you can edit an existing database profile or create a new one.

❖   **To edit or create a database profile:**

•   Click the Database Profile button in the PainterBar and then click the Edit button or the New button.

For information about how to edit or define a database profile, see *Connecting to Your Database*.

Correcting pipeline errors

If the pipeline cannot pipe certain rows to the destination table for some reason, PowerBuilder displays the following information for the error rows:

- Name of the table in the destination database

- Pipeline operation you chose in the Option box

- Error messages to identify the problem with each row

- Data values in the error rows

- Source and destination column information

The following screen shot displays this information:
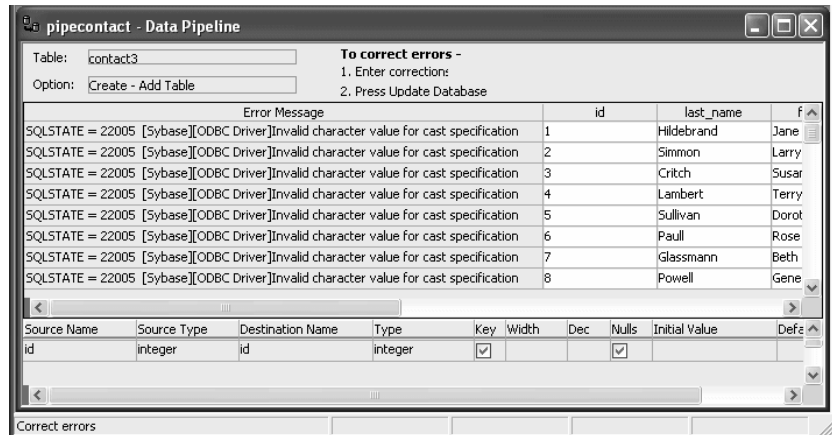
What you can do

You can correct the error rows by changing one or more of their column values so the destination table will accept them, or you can ignore the error rows and return to the Data Pipeline painter workspace. If you return to the workspace, you cannot redisplay the error rows without re-executing the pipeline.

Before you return to the workspace
You might want to print the list of errors or save them in a file. Select File>Print or File>Save As from the menu bar.

❖ **To return to the Data Pipeline painter workspace without correcting errors:**

- Click the Design button.

❖ **To correct pipeline errors:**

1   Change data values for the appropriate columns in the error rows.

2   Click the Update DB button, or select Design>Update Database from the menu bar.

PowerBuilder pipes rows in which errors were corrected to the destination table and displays any remaining errors.

3   Repeat steps 1 and 2 until all errors are corrected.

The Data Pipeline painter workspace displays.

Viewing an error message

Sometimes you cannot see an entire error message because the column is not wide enough. Move the pointer to the error message and press the Right Arrow key to scroll through it. You can also drag the Error Message column border to the width needed.

Making the error messages shorter
For ODBC data sources, you can set the MsgTerse database parameter in the destination database profile to make the error messages shorter. If you type `MsgTerse = 'Yes'`, then the SQLSTATE error number does not display. For more information on the MsgTerse parameter

, see the online Help.

Saving a pipeline

When you have generated a pipeline definition in the Data Pipeline painter workspace, you should save the pipeline. You can then reuse it later.

❖ **To save a pipeline:**

•   Click the Save button, or select File>Save from the menu bar.

For a new pipeline

When you save a pipeline for the first time, you must specify a name. The name can be any valid identifier with up to 40 characters. A common convention is to prefix the name with the string pipe_. You can also specify the library in which the pipeline is saved.

Using an existing pipeline

If you save a pipeline, you can modify and execute it any time. You can also pipe data that might have changed since the last pipeline execution or pipe data to other databases.

❖ **To use an existing pipeline:**

1 Click the Open button in the PowerBar.

2 In the Open dialog box, select the Pipelines object type in the Object Type drop-down list, select the library, select the pipeline you want to execute, and click OK.

In the Open dialog box, pipelines in the selected libraries are listed. If you do not see the pipeline you want, close the dialog box and add the library you need to the target's library search path.

3 If you want to change the pipeline operation, select a new option from the Options drop-down list in the workspace.

4 Modify the pipeline definition as needed.

5 Execute and/or save the pipeline.

Pipeline examples

Updating data in a
destination table

You might want to pipe data and then update the data often.

❖ **To update a destination table:**

1 Click the Pipeline button, select an existing pipeline that you executed before, and click OK.

The pipeline definition displays. Since this pipeline has been executed before, the table exists in the destination database.

2 Select the Update option in the pipeline definition.

3 Execute the pipeline.

The destination table is updated with current data from the source database.

Reproducing a table definition with no data

You can force a pipeline to create a table definition and not pipe data. To do this, you must use Quick Select, SQL Select, or Query as the data source. It is easiest to do it using SQL Select.

❖ **To reproduce a table definition with no data:**

1 Click the Pipeline button, click New, select SQL Select as the data source and specify the source and destination databases, and click OK.

2 In the Select painter, open the table you want to reproduce and select all columns.

3 On the Where tab page, type an expression that will never evaluate to true, such as $1 = 2$.

4 Click the SQL Select button to create the pipeline definition.

5 Select the Extended Attributes check box.

6 Click the Execute button to execute the pipeline.

The table definition is piped to the destination database, but no rows of data are piped. You can open the new table in the Database painter and then click the Grid, Table, or Freeform button to view the data. As specified, there is no data.

If you use a data source other than SQL Select, you can follow the previous procedure, but you need to edit the data source of the pipeline to open the Select painter in step 2.

Piping a table to many databases

In the Data Pipeline painter workspace, you can execute a pipeline many times with a different destination database each time.

❖ **To pipe a table to many databases:**

1 Select File>Destination Connect from the menu bar to change the destination to the database you want.

2 Execute the pipeline.

3 Repeat steps 1 and 2 for each database you want.

P A R T 6

Working with DataWindows

This part describes how to build DataWindow objects to retrieve, present, and manipulate data in your applications.

Defining DataWindow Objects

About this chapter

The applications you build are centered around your organization's data. This chapter describes how to define DataWindow objects to display and manipulate the data.

Contents

About DataWindow objects

A DataWindow object is an object you use to retrieve, present, and manipulate data from a relational database or other data source (such as an Excel worksheet or dBASE file).

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

DataWindow object examples

You can display the data in the format that best presents the data to your users.

Edit styles

If a column can take only a small number of values, you can have the data appear as radio buttons in a DataWindow object so that users know what their choices are.
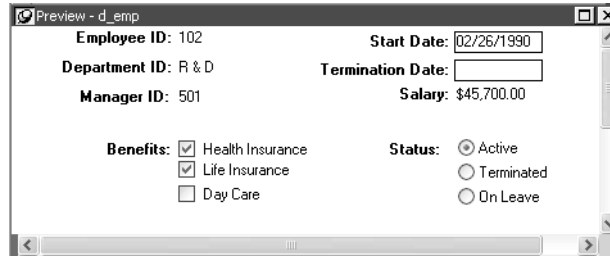
Display formats

If a column displays phone numbers, salaries, or dates, you can specify the format appropriate to the data.
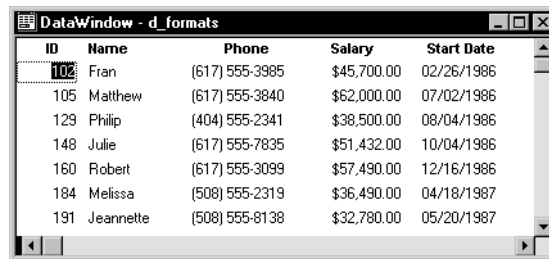
Validation rules

If a column can take numbers only in a specific range, you can specify a simple validation rule for the data, without writing any code, to make sure users enter valid data.

Enhancing DataWindow objects

If you want to enhance the presentation and manipulation of data in a DataWindow object, you can include computed fields, pictures, and graphs that are tied directly to the data retrieved by the object.

How to use DataWindow objects

Before you can use a DataWindow object, you need to build the object. To do that you can go to the DataWindow painter, which lets you create and edit DataWindow objects. It also lets you make PSR (Powersoft report) 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.

This section describes the overall process for creating and using DataWindow objects. You can use DataWindow objects in client/server, Web-based, and multitier applications. For more information about using DataWindow objects in different kinds of applications and writing code that interacts with DataWindow objects, see the *DataWindow Programmer's Guide*.

❖ **To use DataWindow objects in an application:**

1   Create the DataWindow *object* using one of the DataWindow wizards on the DataWindow tab page of the New dialog box.

The wizard helps you define the data source, presentation style, and other basic properties of the object, and the DataWindow object displays in the DataWindow painter. In this painter, you define additional properties for the DataWindow object, such as display formats, validation rules, and sorting and filtering criteria.

For more information about creating a DataWindow object, see "Building a DataWindow object" on page 474.

2   Place a DataWindow *control* in a window or user object.

It is through this control that your application communicates with the DataWindow object you created in the DataWindow painter.

3   Associate the DataWindow control with the DataWindow object.

4   Write scripts in the Window painter to manipulate the DataWindow control and its contents.

For example, you use the PowerScript Retrieve method to retrieve data into the DataWindow control.

You can write scripts for the DataWindow control to deal with error handling, sharing data between DataWindow controls, and so on.

Reports versus
DataWindow objects

Reports and DataWindow objects are the same objects. You can open and modify both in the DataWindow painter. However, a report is not updatable and can only be used to present data. For information about how you can specify whether users can update the data in a DataWindow object, see Chapter 21, "Controlling Updates in DataWindow Objects."

Choosing a presentation style

The presentation style you select for a DataWindow object determines the format PowerBuilder uses to display the DataWindow object in the Design view. You can use the format as displayed or modify it to meet your needs.

When you create a DataWindow object, you can choose from the presentation styles listed in the following table.

Table 18-1: Report presentation styles

Using this Report wizard	You create a new DataWindow object
Composite	That includes other DataWindow objects
Crosstab	With summary data in a spreadsheet-like grid
Freeform	With the data columns going down the page and labels next to each column
Graph	With data displayed in a graph
Grid	With data in row and column format with grid lines separating rows and columns
Group	With data in rows that are divided into groups
Label	That presents data as labels
N-Up	With two or more rows of data next to each other
OLE 2.0	That is a single OLE object
RichText	That combines input fields that represent database columns with formatted text
Tabular	With data columns going across the page and headers above each column
TreeView	With data grouped in rows in a TreeView; the TreeView displays the data hierarchically in a way that allows you to expand and collapse it

Using the Tabular style

The Tabular presentation style presents data with the data columns going across the page and headers above each column. As many rows from the database will display at one time as can fit in the DataWindow object. You can reorganize the default layout any way you want by moving columns and text:
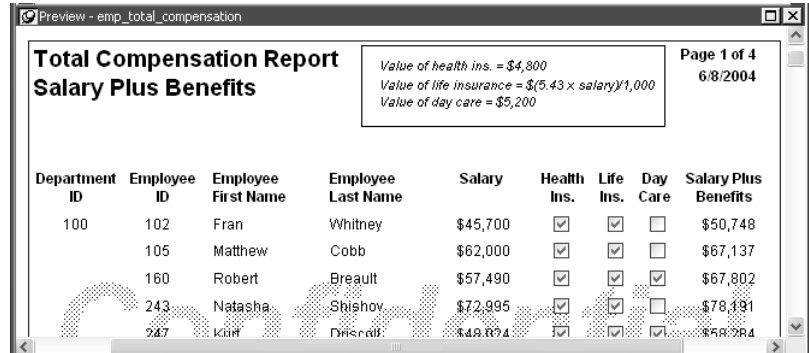
Using the Freeform style

The Freeform presentation style presents data with the data columns going down the page and labels next to each column. You can reorganize the default layout any way you want by moving columns and text. The Freeform style is often used for data entry forms.
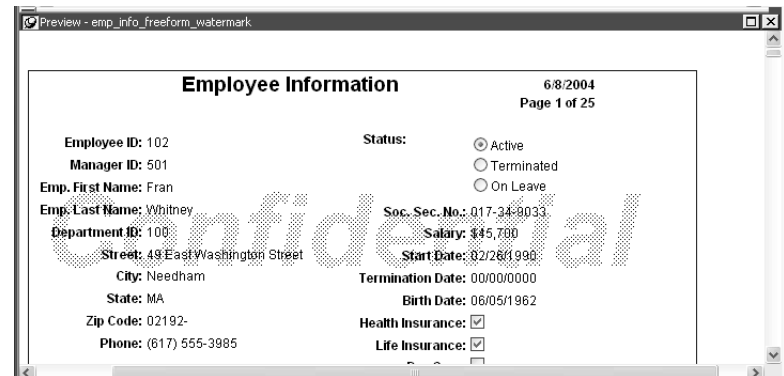
Using the Grid style

The Grid presentation style shows data in row-and-column format with grid lines separating rows and columns. With other styles, you can move text, values, and other objects around freely in designing the report. With the grid style, the grid lines create a rigid structure of cells.

An advantage of the Grid style is that users can reorder and resize columns at runtime.

Original Grid report

This grid report shows employee information. Several of the columns have a large amount of extra white space:

Employee ID	First Name	Last Name	Street	City	Sta
102	Fran	Whitney	49 East Washington Street	Needham	MA
105	Matthew	Cobb	77 Pleasant Street	Waltham	MA
129	Philip	Chin	59 Pond Street	Atlanta	GA
148	Julie	Jordan	144 Great Plain Avenue	Winchester	MA
160	Robert	Breault	58 Cherry Street	Milton	MA
184	Melissa	Espinoza	112 Apple Tree Way	Stow	MA
191	Jeannette	Bertrand	209 Concord Street	Acton	MA
195	Marc	Dill	89 Hancock Street	Milton	MA
207	Jane	Francis	12 Hawthorne Drive	Concord	MA

Grid report with modified column widths

This grid report was created from the original one by decreasing the width of some columns:

Employee ID	First Name	Last Name	Street	City	State	Phone
102	Fran	Whitney	49 East Washington Street	Needham	MA	(617) 555-3985
105	Matthew	Cobb	77 Pleasant Street	Waltham	MA	(617) 555-3840
129	Philip	Chin	59 Pond Street	Atlanta	GA	(404) 555-2341
148	Julie	Jordan	144 Great Plain Avenue	Winchester	MA	(617) 555-7835
160	Robert	Breault	58 Cherry Street	Milton	MA	(617) 555-3099
184	Melissa	Espinoza	112 Apple Tree Way	Stow	MA	(508) 555-2319
191	Jeannette	Bertrand	209 Concord Street	Acton	MA	(508) 555-8138
195	Marc	Dill	89 Hancock Street	Milton	MA	(617) 555-2144
207	Jane	Francis	12 Hawthorne Drive	Concord	MA	(508) 555-9022

Using the Label style

The Label presentation style shows data as labels. With this style you can create mailing labels, business cards, name tags, index cards, diskette labels, file folder labels, and many other types of labels.

Mailing labels

Business cards

Name tags

Specifying label properties

If you choose the Label style, you are asked to specify the properties for the label after specifying the data source. You can choose from a list of predefined label types or enter your own specifications manually.

Where label definitions come from

PowerBuilder gets the information about the predefined label formats from a preferences file called *PBLAB105.INI*.

Using the N-Up style

The N-Up style presents two or more rows of data next to each other. It is similar to the Label style in that you can have information from several rows in the database across the page. However, the information is not meant to be printed on labels. The N-Up presentation style is useful if you have periodic data; you can set it up so that each period repeats in a row.

After you select a data source, you are asked how many rows to display across the page.

For each column in the data source, PowerBuilder defines *n* columns in the DataWindow object (column_1 to column_*n*), where *n* is the number of rows you specified.

Table example
For a table of daily stock prices, you can define the DataWindow object as five across, so each row in the DataWindow object displays five days' prices (Monday through Friday). Suppose you have a table with two columns, day and price, that record the closing stock price each day for three weeks.

In the following n-up DataWindow object, 5 was selected as the number of rows to display across the page, so each line in the DataWindow object shows five days' stock prices. A computed field was added to get the average closing price in the week:

About computed fields in n-up DataWindow objects
You use subscripts, such as price[0], to refer to particular rows in the detail band in n-up DataWindow objects.

For more information, see Chapter 19, "Enhancing DataWindow Objects."

Here is the DataWindow object in the Preview view:

Monday		Tuesday		Wednesday		Thursday		Friday	
05/04/01	62.00	05/05/01	63.00	05/06/01	66.00	05/07/01	65.00	05/08/01	62.00
Average price for week: $63.6									
05/11/01	65.00	05/12/01	69.00	05/13/01	66.00	05/14/01	68.00	05/15/01	67.00
Average price for week: $67									
05/18/01	67.00	05/19/01	71.00	05/20/01	70.00	05/21/01	72.00	05/22/01	75.00
Average price for week: $71									

Another way to get multiple-column DataWindow objects
In an n-up DataWindow object, the data is displayed across and then down. If you want your data to go down the page and then across in multiple columns, as in a phone list, you should create a standard tabular DataWindow object, then specify newspaper columns.

For more information on newspaper columns, see Chapter 19, "Enhancing DataWindow Objects."

Using the Group style

The Group presentation style provides an easy way to create grouped DataWindow objects, where the rows are divided into groups, each of which can have statistics calculated for it. Using this style generates a tabular DataWindow object that has grouping properties defined.

This Group style report groups by department and lists employees and salaries. It also includes a subtotal and a grand total for the salary column:

Employee Report
08/14/02

Department ID	Employee ID	First Name	Last Name	Salary
500				
	191	Jeannette	Bertrand	$92,780
	703	Jose	Martinez	$91,051
	750	Jane	Braun	$77,730
	868	Felicia	Kuo	$88,200
	921	Charles	Crowley	$61,700
	1013	Joseph	Barker	$47,290
	1570	Anthony	Rebeiro	$54,576
	1615	Sheila	Romero	$77,500
	1658	Michael	Lynch	$64,903
			Total for department:	$655,730
			Grand Total:	$4,101,107

For more about the Group presentation style, see Chapter 23, "Filtering, Sorting, and Grouping Rows."

Using the Composite style

The Composite presentation style allows you to combine multiple DataWindow objects in the same object. It is particularly handy if you want to print more than one DataWindow object on a page.

This composite report consists of three nested tabular reports. One of the tabular reports includes a graph:

For more about the Composite presentation style, see Chapter 25, "Using Nested Reports."
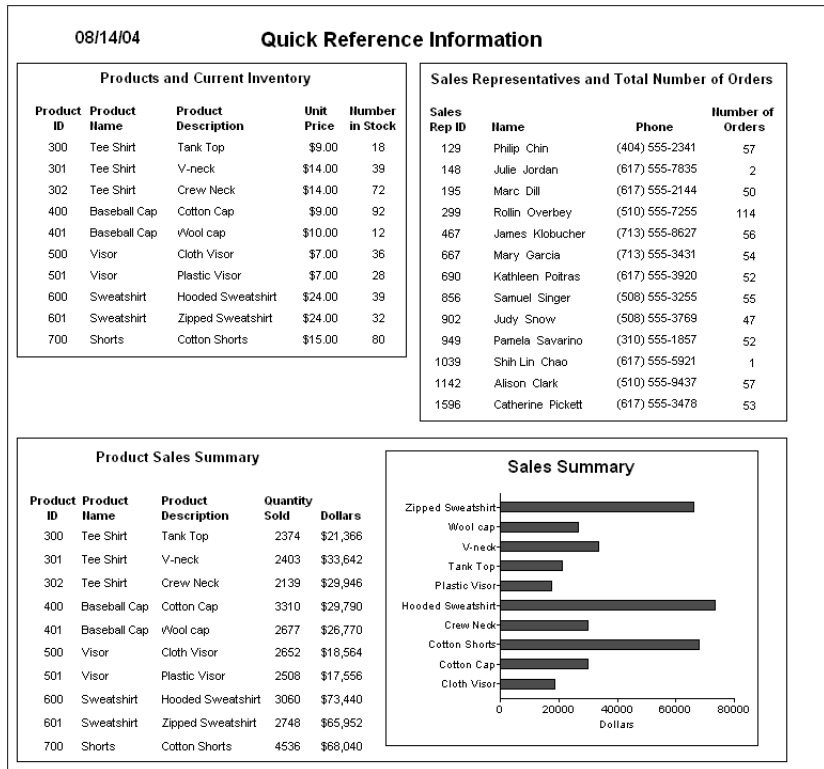
Using the Graph and Crosstab styles

In addition to the (preceding) text-based presentation styles, PowerBuilder provides two styles that allow you to display information graphically: Graph and Crosstab.

There is a graph report in the composite report in "Using the Composite style" on page 472. This crosstab report counts the number of employees that fit into each cell. For example, there are three employees in department 100 who make between $30,000 and $39,999:

Number of employees by department and salary 30,000 includes up to 39,999	Dept Id					Total number of employees making the salary
Salary	100	200	300	400	500	
20000				2	4	6
30000	3	8	2	5	3	21
40000	6	5	2	5	1	19
50000	4	3	3	2		12
60000	4	1		2	1	8
70000	2	1	1			4
80000	2	1				3
90000	1					1
130000			1			1
Total number of employees in the department	22	19	9	16	9	

For more information about these two presentation styles, see Chapter 26, "Working with Graphs," and Chapter 27, "Working with Crosstabs."

Using the OLE 2.0 style

The OLE presentation style lets you link or embed an OLE object in a DataWindow object.

For information about the OLE 2.0 presentation style, see Chapter 31, "Using OLE in a DataWindow Object."

Using the RichText style

The RichText presentation style lets you combine input fields that represent database columns with formatted text.

For more information about the RichText presentation style, see Chapter 30, "Working with Rich Text."

Using the TreeView style

The TreeView presentation style provides an easy way to create DataWindow objects that display hierarchical data in a TreeView, where the rows are divided into groups that can be expanded and collapsed. Icons (+ or –) show whether the state of a group in the TreeView is expanded or collapsed, and lines connect parents and their children.

This TreeView style report groups by manager ID and state and lists employee information and salaries:

Dept. Head ID	State	City	Employee ID	Name		Salary
501						
	MA					
	TX					
703						
902						
	CA					
		Emeryville				
			299	Rollin	Overbey	$39,300.00
			1142	Alison	Clark	$45,000.00
		Long Beach				
	GA					
	MA					
		Bedford				
			1596	Catherine	Pickett	$47,653.00
		Boston				
		Concord				
		Lexington				
		Marblehead				
		Milton				
		Needham				
		Newton				
		Stow				
		Wellesley				

For more about the TreeView presentation style, see Chapter 28, "Working with TreeViews."

Building a DataWindow object

You use a wizard to build a new DataWindow object. To create a DataWindow object or use the DataWindow painter, you must be connected to the database whose data you will be accessing. When you open the DataWindow painter or select a data source in the wizard, PowerBuilder connects you to the DBMS and database you used last. If you need to connect to a different database, do so before working with a DataWindow object.

Column limit
There is a limit of 1000 on the number of columns in a DataWindow object.

For information about changing your database connection, see *Connecting to Your Database*.

❖ **To create a new DataWindow object:**

1   Select File>New from the menu bar and select the DataWindow tab.

2   If there is more than one target in the workspace, select the target where you want the DataWindow to be created from the drop-down list at the bottom of the dialog box.

3   Choose a presentation style for the DataWindow object.

    The presentation style determines how the data is displayed. See "Choosing a presentation style" on page 466. When you choose the presentation style, the appropriate DataWindow object wizard starts up.

4   If you want data to be retrieved in the Preview view when the DataWindow object opens, select the Retrieve on Preview check box.

5   Define the data source.

    See "Selecting a data source" on page 476.

6   Choose options for the DataWindow object and click Next.

    See "Choosing DataWindow object-wide options" on page 506.

7   Review your specifications and click Finish.

    The DataWindow object displays in the Design view.

8   Save the DataWindow object in a library.

Selecting a data source

The data source you choose determines how you select the data that will be used in the DataWindow object.

About the term *data source*
The term *data source* used here refers to how you use the Report painter to specify the data to retrieve into the DataWindow object. Data source can also refer to where the data comes from, such as an Adaptive Server Anywhere data source (meaning a database file) or an XML data source (meaning an XML file). *Connecting to Your Database* uses the term data source in this second sense.

If the data is in the database

If the data for the DataWindow object will be retrieved from a database, choose one of the following data sources:

Data source	Use when
Quick Select	The data is from a single table (or from tables that are related through foreign keys) and you need only to choose columns, selection criteria, and sorting.
SQL Select	You want more control over the SQL SELECT statement generated for the data source *or* your data is from tables that are not connected through a key. For example, you need to specify grouping, computed columns, or retrieval arguments within the SQL SELECT statement.
Query	The data has been defined as a query.
Stored Procedure	The data is defined in a stored procedure.

If the data is not in a database

Select the External data source if:

• The DataWindow object will be populated programmatically.

• Data will be imported from a DDE application.

• Data will be imported from an external file, such as an XML, comma-separated values (*CSV*), tab-separated text (*TXT*), or dBASE (*DBF*) file.

You can also use an ODBC driver to access data from a file. PowerBuilder includes ODBC drivers for most file types.

For more information, see *Connecting to Your Database.*

After you choose a data source in the various DataWindow wizards, you specify the data. The data source you choose determines what displays in the wizards and how you define the data.

Why use a
DataWindow if the
data is not from a
DBMS

Even when the data is not coming from the database, there are many times when you want to take advantage of the intelligence of a DataWindow object:

- **Data Validation**   You have full access to validation rules for data

- **Display Formats**   You can use any existing display formats to present the data, or create your own

- **Edit Styles**   You can use any existing edit styles, such as radio buttons and edit masks, to present the data, or create your own

Using Quick Select

The easiest way to define a data source is using Quick Select.

❖ **To define the data using Quick Select:**

1   Click Quick Select in the Choose Data Source dialog box in the wizard and click Next.

2   Select the table that you will use in the DataWindow object.

    For more information, see "Selecting a table" next.

3   Select the columns to be retrieved from the database.

    For more information, see "Selecting columns" on page 480.

4   (Optional) Sort the rows before you retrieve data.

    For more information, see "Specifying sorting criteria" on page 480.

5   (Optional) Select what data to retrieve.

    For more information, see "Specifying selection criteria" on page 481.

6   Click the OK button Quick Select dialog box.

    You return to the wizard to complete the definition of the DataWindow object.

Quick Select
limitations

When you choose Quick Select as your data source, you cannot:

- Specify grouping before rows are retrieved

- Include computed columns

- Specify retrieval arguments for the SELECT statement that are supplied at runtime.

To use these options when you create a DataWindow object, choose SQL Select as your data source. If you decide later that you want to use retrieval arguments, you can define them by modifying the data source. For more information, see Chapter 19, "Enhancing DataWindow Objects."

Selecting a table

When you choose Quick Select, the Quick Select dialog box displays. The Tables box lists tables and views in the current database.

Displaying table comments
To display a comment about a table, position the pointer on the table and click the right mouse button or select the table.

Which tables and views display?

The DBMS determines what tables and views display. For some DBMSs, all tables and views display, whether or not you have authorization. If you select a table or view you are not authorized to access, the DBMS issues a message.

For ODBC databases, the tables and views that display depend on the driver for the data source. Adaptive Server Anywhere does not restrict the display, so all tables and views display, whether or not you have authorization.

Tables with key
relationships

When you select a table, the table's column names display in the Columns box, and any tables having a key relationship with the selected table display in the Tables box. These tables are indented and marked with an arrow to show their relationship to the selected table. You can select any of these related tables if you want to include columns from them in the DataWindow object.
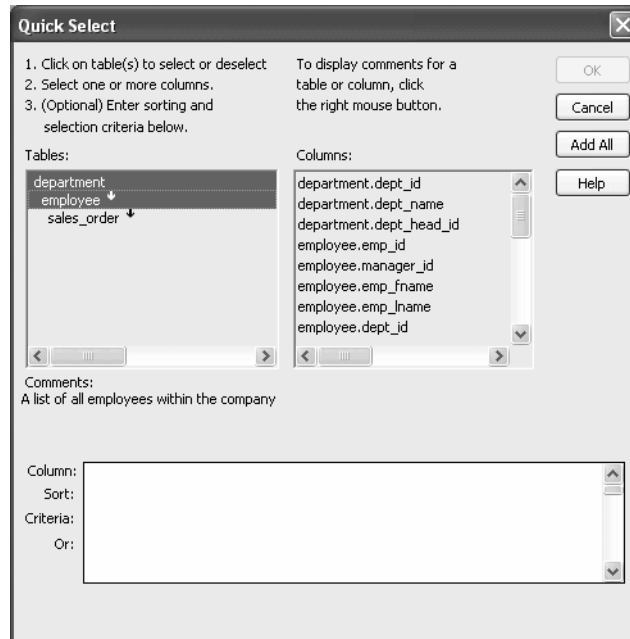
Meaning of the up and
down arrows

An arrow displays next to a table to indicate its relationship to the selected table. The arrow always points in the *many* direction of the relationship—toward the selected table (up) if the selected table contains a foreign key in the relationship and away from the selected table (down) if the selected table contains a primary key in the relationship:
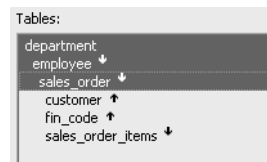
In this preceding illustration, the selected table is sales_order. The Up arrows indicate that a foreign key in the sales_order table is mapped to the primary key in the customer and fin_code tables. The Down arrow indicates that the sales_order_items table contains a foreign key mapped to the primary key in the sales_order table.

How columns from additional tables display	The column names of selected tables display in the Columns box. If you select more than one table, the column names are identified as:

tablename.columnname

For example, department.dept_name and employee.emp_id display when the Employee table and the Department table are selected.

To return to the original table list
Click the table you first selected at the top of the table list. |

Selecting columns

You can select columns from the primary table and from its related tables. Select the table whose columns you want to use in the Tables box, and add columns from the Columns box:

- To add a column, select it in the Columns box.

- To add all the columns that display in the Columns box, click Add All.

- To remove a column, deselect it in the Columns box.

- To view comments that describe a table or column, position the pointer on a table or column name, and press and hold the right mouse button.

As you select columns, they display in the grid at the bottom of the dialog box in the order in which you select them. If you want the columns to display in a different order in the DataWindow object, select a column name you want to move in the grid and drag it to the new location.

Specifying sorting criteria

In the grid at the bottom of the Quick Select dialog box, you can specify if you want the retrieved rows to be sorted. As you specify sorting criteria, PowerBuilder builds an ORDER BY clause for the SELECT statement.

❖ **To sort retrieved rows on a column:**

1   Click in the Sort row for the column you want to sort on.

PowerBuilder displays a drop-down list:

Column:	Emp Id	Dept Id	Salary	
Sort:		(not sorted) ▼		
Criteria:		Ascending		
		Descending		
Or:		(not sorted)		
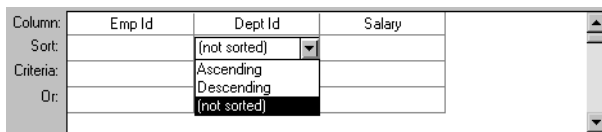
2   Select the sorting order for the rows: Ascending or Descending.

Multilevel sorts   You can specify as many columns for sorting as you want. PowerBuilder processes the sorting criteria left to right in the grid: the first column with Ascending or Descending specified becomes the highest level sorting column, the next column with Ascending or Descending specified becomes the next level sorting column, and so on.

If you want to do a multilevel sort that does not match the column order in the grid, drag the columns to the correct order and then specify the columns for sorting.

Specifying selection criteria

You can enter selection criteria in the grid to specify which rows to retrieve. For example, instead of retrieving data about all employees, you might want to limit the data to employees in Sales and Marketing, or to employees in Sales who make more than $80,000.

As you specify selection criteria, PowerBuilder builds a WHERE clause for the SELECT statement.

❖ **To specify selection criteria:**

1   Click the Criteria row below the first column for which you want to select the data to retrieve.

2   Enter an expression, or if the column has an edit style, select or enter a value.

If the column is too narrow for the criterion, drag the grid line to enlarge the column. This enlargement does not affect the column size in a DataWindow object.

3   Enter additional expressions until you have specified the data you want to retrieve.

About edit styles

If a column has an edit style associated with it in the extended attribute system tables (that is, the association was made in the Database painter), if possible, the edit style is used in the grid. Drop-down list boxes are used for columns with code tables and columns using the CheckBox and RadioButton edit styles.

SQL operators supported in Quick Select

You can use these SQL relational operators in the retrieval criteria:

Table 18-2: SQL relational operators used in retrieval criteria

Operator	Meaning
=	Is equal to (default operator)
>	Is greater than
<	Is less than
< >	Is not equal to
> =	Is greater than or equal to
< =	Is less than or equal to
LIKE	Matches this pattern
NOT LIKE	Does not match this pattern
IN	Is in this set of values
NOT IN	Is not in this set of values

Because = is the default operator, you can enter the value 100 instead of = 100, or the value New Hampshire instead of = New Hampshire.

Comparison operators

You can use the LIKE, NOT LIKE, IN, and NOT IN operators to compare expressions.

Use LIKE to search for strings that match a predetermined pattern. Use NOT LIKE to find strings that do not match a predetermined pattern. When you use LIKE or NOT LIKE, you can use wildcards:

- The percent sign (%), like the DOS wildcard asterisk (*), matches multiple characters. For example, Good% matches all names that begin with Good.

- The underscore character (_) matches a single character. For example, Good _ _ _ matches all seven-letter names that begin with Good.

Use IN to compare and include a value that is in a set of values. Use NOT IN to compare and include values that are not in a set of values. For example, the following clause selects all employees in department 100, 200, or 500:

```
SELECT * from employee
WHERE dept_id IN (100, 200, 500)
```

Using NOT IN in this clause would exclude employees in those departments.

Connection operators

You can use the OR and AND logical operators to connect expressions.

PowerBuilder makes some assumptions based on how you specify selection criteria. When you specify:

- Criteria for more than one column on one line

  PowerBuilder assumes a logical AND between the criteria. A row from the database is retrieved if *all* criteria in the line are met.

- Two or more lines of selection criteria

  PowerBuilder assumes a logical OR. A row from the database is retrieved if the criterion in *any* of the lines is met.

To override these defaults, begin an expression with the AND or OR operator:

Operator	Meaning
OR	The row is selected if one expression OR another expression is true
AND	The row is selected if one expression AND another expression are true
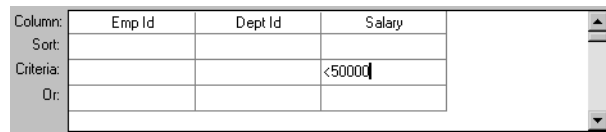
This technique is particularly handy when you want to retrieve a range of values in a column. See example 6 below.

SQL expression examples

The first six examples in this section all refer to a grid that contains three columns: EmpId, DeptId, and Salary.

Example 1

The expression `<50000` in the Criteria row in the Salary column in the grid retrieves information for employees whose salaries are less than $50,000.

The SELECT statement that PowerBuilder creates is:

```
SELECT emp_id, dept_id, salary
FROM employee
WHERE salary < 50000
```

Example 2

The expression 100 in the Criteria row in the DeptId column in the grid retrieves information for employees who belong to department 100.

Column:	Emp Id	Dept Id	Salary	
Sort:				
Criteria:		100		
Or:				

The SELECT statement that PowerBuilder creates is:

```
SELECT emp_id, dept_id, salary
FROM employee
WHERE dept_id = 100
```

Example 3

The expression >300 in the Criteria row in the EmpId column and the expression <50000 in the Criteria row in the Salary column in the grid retrieve information for any employee whose employee ID is greater than 300 *and* whose salary is less than $50,000.

Column:	Emp Id	Dept Id	Salary	
Sort:				
Criteria:	>300		<50000	
Or:				

The SELECT statement that PowerBuilder creates is:

```
SELECT emp_id, dept_id, salary
FROM employee
WHERE emp_id >300 AND salary <50000
```

Example 4

The expressions 100 in the Criteria row and >300 in the Or row for the DeptId column, together with the expression <50000 in the Criteria row in the Salary column, retrieve information for employees who belong to:

- Department 100 *and* have a salary less than $50,000

  or

- A department whose ID is greater than 300, no matter what their salaries

Column:	Emp Id	Dept Id	Salary	
Sort:				
Criteria:		100	<50000	
Or:		>300		

The SELECT statement that PowerBuilder creates is:

```
SELECT emp_id, dept_id, salary
FROM employee
```

```
WHERE (dept_id = 100 AND salary < 50000)
OR dept_id > 300
```

Example 5

The expression IN(100, 200, 500) in the Criteria row in the DeptId column in the grid retrieves information for employees who are in department 100 *or* 200 *or* 500.

Column:	Emp Id	Dept Id	Salary	
Sort:				
Criteria:		IN (100,200,500)		
Or:				

The SELECT statement that PowerBuilder creates is:

```
SELECT emp_id, dept_id, salary
FROM employee
WHERE dept_id IN (100, 200, 500)
```

Example 6

This example shows the use of the word AND in the Or criteria row. In the Criteria row, >=500 is in the EmpId column and >=30000 is in the Salary column. In the Or row, AND <=1000 is in the EmpId column and AND <=50000 is in the Salary column. These criteria retrieve information for employees who have an employee ID from 500 to 1000 and a salary from $30,000 to $50,000.

Column:	Emp Id	Dept Id	Salary	
Sort:				
Criteria:	>= 500		>= 30000	
Or:	AND <= 1000		AND <= 50000	

The SELECT statement that PowerBuilder creates is:

```
SELECT emp_id, dept_id, salary
FROM employee
WHERE (emp_id >= 500 AND emp_id <= 1000)
AND (salary >= 30000 AND salary <= 50000)
```

Example 7

In a grid with three columns: Emp Last Name, Emp First Name, and Salary, the expressions LIKE C% in the Criteria row and LIKE G% in the Or row in the Emp Last Name column retrieve information for employees who have last names that begin with C or G.

Column:	Emp Last Name	Emp First Name	Salary	
Sort:				
Criteria:	LIKE C%			
Or:	LIKE G%			

The SELECT statement that PowerBuilder creates is:

```
SELECT emp_last_name, emp_first_name, salary
FROM employee
WHERE emp_last_name LIKE 'C%'
OR emp_last_name LIKE 'G%'
```

Providing SQL
functionality to users

You can allow your users to specify selection criteria in a DataWindow object using these techniques at runtime:

- You can automatically pop up a window prompting users to specify criteria each time, just before data is retrieved.

  For more information, see Chapter 19, "Enhancing DataWindow Objects."

- You can place the DataWindow object in query mode using the Modify method.

For more information, see the *DataWindow Programmer's Guide*.

Using SQL Select

In specifying data for a DataWindow object, you have more options for specifying complex SQL statements when you use SQL Select as the data source. When you choose SQL Select, you go to the Select painter, where you can paint a SELECT statement that includes the following:

- More than one table

- Selection criteria (WHERE clause)

- Sorting criteria (ORDER BY clause)

- Grouping criteria (GROUP BY and HAVING clauses)

- Computed columns

- One or more arguments to be supplied at runtime

Saving your work as a query
While in the Select painter, you can save the current SELECT statement as a query by selecting File>Save Query from the menu bar. Doing so allows you to easily use this data specification again in other reports.

For more information about queries, see "Defining queries" on page 510.

❖ **To define the data using SQL Select:**

1   Click SQL Select in the Choose Data Source dialog box in the wizard and click Next.

    The Select Tables dialog box displays.

2   Select the tables and/or views that you will use in the DataWindow object.

    For more information, see "Selecting tables and views" next.

3   Select the columns to be retrieved from the database.

    For more information, see "Selecting columns" on page 489.

4   Join the tables if you have selected more than one.

    For more information, see "Joining tables" on page 492.

5   Select retrieval arguments if appropriate.

    For more information, see "Using retrieval arguments" on page 495.

6   Limit the retrieved rows with WHERE, ORDER BY, GROUP BY, and HAVING criteria, if appropriate.

    For more information, see "Specifying selection, sorting, and grouping criteria" on page 497.

7   If you want to eliminate duplicate rows, select Distinct from the Design menu. This adds the DISTINCT keyword to the SELECT statement.

8   Click the Return button on the PainterBar.

    You return to the wizard to complete the definition of the DataWindow object.

Selecting tables and views

After you have chosen SQL Select, the Select Tables dialog box displays in front of the Table Layout view of the Select painter. What tables and views display in the dialog box depends on the DBMS. For some DBMSs, all tables and views display, whether or not you have authorization. Then, if you select a table or view you are not authorized to access, the DBMS issues a message.

For ODBC databases, the tables and views that display depend on the driver for the data source. Adaptive Server Anywhere does not restrict the display, so all tables and views display, whether or not you have authorization.

❖ **To select the tables and views:**

- Do one of the following:

  - Click the name of each table or view you want to open.

    Each table you select is highlighted. (To deselect a table, click it again.) Click the Open button to close the Select Tables dialog box.

  - Double-click the name of each table or view you want to open.

    Each object opens immediately behind the Select Tables dialog box. Click the Cancel button to close the Select Tables dialog box.

Representations of the selected tables and views display. You can move or size each table to fit the space as needed.

Below the Table Layout view, several tabbed views also display by default. You use the views (for example, Compute, Having, Group) to specify the SQL Select statement in more detail. You can turn the views on and off from the View menu on the menu bar.


```
SELECT <not specified>
  FROM "department",
       "employee"
  WHERE ( "employee"."dept_id" = "department"."dept_id" )
```

Specifying what is displayed

You can display the label and datatype of each column in the tables (the label information comes from the extended attribute system tables). If you need more space, you can choose to hide this information.

❖ **To hide or display comments, datatypes, and labels:**

1 Position the pointer on any unused area of the Table Layout view and select Show from the pop-up menu.

A cascading menu displays.

2 Select or clear Datatypes, Labels, or Comments as needed.

Colors in the Select painter

The colors used by the Select painter to display the Table Layout view background and table information are specified in the Database painter. You can also set colors for the text and background components in the table header and detail areas.

For more information about specifying colors in the Database painter, see "Modifying database preferences" on page 394.

Adding and removing tables and views

You can add tables and views to your Table Layout view at any time.

Table 18-3: Adding tables and views in the Select painter

To do this	Do this
Add tables or views	Click the Tables button in the PainterBar and select tables or views to add
Remove a table or view	Display its pop-up menu and select Close
Remove all tables and views	Select Design>Undo All from the menu bar

You can also remove individual tables and views from the Table Layout view, or clear them all at once and begin selecting a new set of tables.

How PowerBuilder joins tables

If you select more than one table in the Select painter, PowerBuilder joins columns based on their key relationship.

For information about joins, see "Joining tables" on page 492.

Selecting columns

You can click each column you want to include from the table representations in the Table Layout view. PowerBuilder highlights selected columns and places them in the Selection List at the top of the Select painter.

❖ **To reorder the selected columns:**

• Drag a column in the Selection List with the mouse. Release the mouse button when the column is in the proper position in the list.

❖ **To select all columns from a table:**

- Move the pointer to the table name and select Select All from the pop-up menu.

❖ **To include computed columns:**

1 Click the Compute tab to make the Compute view available (or select View>Compute if the Compute view is not currently displayed).

Each row in the Compute view is a place for entering an expression that defines a computed column.

2 Enter one of the following:

- An expression for the computed column. For example: `salary/12`

- A function supported by your DBMS. For example, the following is an Adaptive Server Anywhere function:

    ```
    substr("employee"."emp_fname",1,2)
    ```

You can display the pop-up menu for any row in the Compute view. Using the pop-up menu, you can select and paste the following into the expression:

- Names of columns in the tables used in the report, form, or pipeline

- Any retrieval arguments you have specified

- Functions supported by the DBMS

    About these functions
    The functions listed here are provided *by your DBMS*. They are not PowerBuilder functions. This is so because you are now defining a SELECT statement that will be sent to your DBMS for processing.

3 Press the Tab key to get to the next row to define another computed column, or click another tab to make additional specifications.

PowerBuilder adds the computed columns to the list of columns you have selected.

About computed columns and computed fields

Computed columns you define in the Select painter are added to the SQL statement and used by the DBMS to retrieve the data. The expression you define here follows your DBMS's rules.

You can also choose to define computed fields, which are created and processed dynamically by PowerBuilder after the data has been retrieved from the DBMS. There are advantages to doing this. For example, work is offloaded from the database server, and the computed fields update dynamically as data changes in the DataWindow object. (If you have many rows, however, this updating can result in slower performance.) For more information, see Chapter 19, "Enhancing DataWindow Objects."

Displaying the underlying SQL statement

As you specify the data for the DataWindow object in the Select painter, PowerBuilder generates a SQL SELECT statement. It is this SQL statement that will be sent to the DBMS when you retrieve data into the DataWindow object. You can look at the SQL as it is being generated while you continue defining the data for the DataWindow object.

❖ **To display the SQL statement:**

• Click the Syntax tab to make the Syntax view available, or select View>Syntax if the Syntax view is not currently displayed.

You may need to use the scroll bar to see all parts of the SQL SELECT statement. This statement is updated each time you make a change.

Editing the SELECT statement syntactically

Instead of modifying the data source graphically, you can directly edit the SELECT statement in the Select painter.

Converting from syntax to graphics
If the SQL statement contains unions or the BETWEEN operator, it may not be possible to convert the syntax back to graphics mode. In general, once you convert the SQL statement to syntax, you should maintain it in syntax mode.

❖ **To edit the SELECT statement:**

1 Select Design>Convert to Syntax from the menu bar.

PowerBuilder displays the SELECT statement in a text window.

2 Edit the SELECT statement.

3   Do one of the following:

- Select Design>Convert to Graphics from the menu bar to return to the Select painter.

- Click the Return button to return to the wizard if you are building a new DataWindow object, or to the DataWindow painter if you are modifying an existing DataWindow object.

Joining tables

If the DataWindow object will contain data from more than one table, you should join the tables on their common columns. If you have selected more than one table, PowerBuilder joins columns according to whether they have a key relationship:

- Columns with a primary/foreign key relationship are joined automatically.

- Columns with no key relationship are joined, if possible, based on common column names and types.

PowerBuilder links joined tables in the Select painter Table Layout view. PowerBuilder joins can differ depending on the order in which you select the tables, and sometimes the PowerBuilder best-guess join is incorrect, so you may need to delete a join and manually define a join.

❖   **To delete a join:**

1   Click the join operator connecting the tables.

The Join dialog box displays.

2   Click Delete.

❖   **To join tables:**

1   Click the Join button in the PainterBar.

2   Click the columns on which you want to join the tables.

3   To create a join other than an equality join, click the join operator in the Table Layout view.

The Join dialog box displays:
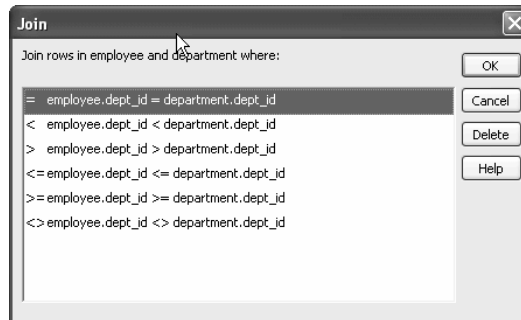
4   Select the join operator you want and click OK.

If your DBMS supports outer joins, outer join options also display in the Join dialog box.

Using ANSI outer joins

All PowerBuilder database interfaces provide support for ANSI SQL-92 outer join SQL syntax generation. PowerBuilder supports both left and right outer joins in graphics mode in the Select painter, and full outer and inner joins in syntax mode. Depending on your database interface, you might need to set the OJSyntax DBParm to enable ANSI outer joins. For more information, see OJSyntax in the online Help.

The syntax for ANSI outer joins is generated according to the following BNF (Backus Naur form):

OUTER-join ::=
table-reference {LEFT | RIGHT} OUTER JOIN *table-reference* ON *search-condition*

table-reference ::=
table_view_name [*correlation_name*] | OUTER-join

Order of evaluation and nesting

In ANSI SQL-92, when nesting joins, the result of the first outer join (determined by order of ON conditions) is the operand of the outer join that follows it. In PowerBuilder, an outer join is considered to be nested if the *table-reference* on the left of the JOIN has been used before within the same outer join nested sequence.

The order of evaluation for ANSI syntax nested outer joins is determined by the order of the ON search conditions. This means that you must create the outer joins in the intended evaluation order and add nested outer joins to the end of the existing sequence, so that the second *table-reference* in the outer join BNF above will always be a *table_view_name*.

Nesting example

For example, if you create a left outer join between a column in Table1 and a column in Table2, then join the column in Table2 to a column in Table3, the product of the outer join between Table1 and Table2 is the operand for the outer join with Table3.

For standard database connections, the default generated syntax encloses the outer joins in escape notation {oj ...} that is parsed by the driver and replaced with DBMS-specific grammar:

```
SELECT Table1.col1, Table2.col1, Table3.col1
FROM {oj {oj Table1 LEFT OUTER JOIN Table2 ON Table1.col1 =
Table2.col1}
LEFT OUTER JOIN Table3 ON Table2.col1 = Table3.col1}
```

Table references

Table references are considered equal when the table names are equal and there is either no alias (correlation name) or the same alias for both. Reusing the operand on the right is not allowed, because ANSI does not allow referencing the *table_view_name* twice in the same statement without an alias.

Determining left and right outer joins

When you create a join condition, the table you select first in the painter is the left operand of the outer join. The table that you select second is the right operand. The condition you select from the Joins dialog box determines whether the join is a left or right outer join.

For example, suppose you select the dept_id column in the employee table, then select the dept_id column in the department table, then choose the following condition:

```
employee.dept_id = department.dept_id and rows from
department that have no employee
```

The syntax generated is:

```
SELECT employee.dept_id, department.dept_id
FROM {oj "employee" LEFT OUTER JOIN "department" ON
"employee"."dept_id" = "department"."dept_id"}
```

If you select the condition with rows from department that have no employee, you create a right outer join instead.

Equivalent statements
The syntax generated when you select table A then table B and create a left outer join is equivalent to the syntax generated when you select table B then table A and create a right outer join.

For more about outer joins, see your DBMS documentation.

Using retrieval arguments

If you know which rows will be retrieved into the DataWindow object at runtime—that is, if you can fully specify the SELECT statement without having to provide a variable—you do not need to specify retrieval arguments.

Adding retrieval arguments

If you decide later that you need arguments, you can return to the Select painter to define the arguments.

Defining retrieval arguments in the DataWindow painter
You can select View>Column Specifications from the menu bar. In the Column Specification view, a column of check boxes next to the columns in the data source lets you identify the columns users should be prompted for. This, like the Retrieval Arguments prompt, calls the Retrieve method.

See Chapter 19, "Enhancing DataWindow Objects."

If you want the user to be prompted to identify which rows to retrieve, you can define retrieval arguments when defining the SQL SELECT statement. For example, consider these situations:

*   Retrieving the row in the Employee table for an employee ID entered into a text box. You must pass that information to the SELECT statement as an argument at runtime.

*   Retrieving all rows from a table for a department selected from a drop-down list. The department is passed as an argument at runtime.

Using retrieval arguments at runtime
If a DataWindow object has retrieval arguments, call the Retrieve method of the DataWindow control to retrieve data at runtime and pass the arguments in the method.

❖ **To define retrieval arguments:**

1   In the Select painter, select Design>Retrieval Arguments from the menu bar.

2   Enter a name and select a datatype for each argument.

    You can enter any valid SQL identifier for the argument name. The position number identifies the argument position in the Retrieve method you code in a script that retrieves data into the DataWindow object.

3   Click Add to define additional arguments as needed and click OK when done.

Specifying an array as a retrieval argument

You can specify an array of values as your retrieval argument. Choose the type of array from the Type drop-down list in the Specify Retrieval Arguments dialog box. You specify an array if you want to use the IN operator in your WHERE clause to retrieve rows that match one of a set of values. For example:

```
SELECT * from employee
WHERE dept_id IN (100, 200, 500)
```

retrieves all employees in department 100, 200, or 500. If you want your user to specify the list of departments to retrieve, you define the retrieval argument as a number array (such as *100, 200, 500*).

In the code that does the retrieval, you declare an array and reference it in the Retrieve method, as in:

```
int x[3]
// Now populate the array with values
// such as x[1] = sle_dept.Text, and so on,
// then retrieve the data, as follows.
dw_1.Retrieve(x)
```

PowerBuilder passes the appropriate comma-delimited list to the method (such as *100, 200, 500* if x[1] = 100, x[2] = 200, and x[3] = 500).

When building the SELECT statement, you reference the retrieval arguments in the WHERE or HAVING clause, as described in the next section.

Specifying selection, sorting, and grouping criteria

In the SELECT statement associated with a DataWindow object, you can add selection, sorting, and grouping criteria that are added to the SQL statement and processed by the DBMS as part of the retrieval.

Table 18-4: Adding selection, sorting, and grouping criteria to the SELECT statement

To do this	Use this clause
Limit the data that is retrieved from the database	WHERE
Sort the retrieved data before it is brought into the DataWindow object	ORDER BY
Group the retrieved data before it is brought into the DataWindow object	GROUP BY
Limit the groups specified in the GROUP BY clause	HAVING

Dynamically selecting, sorting, and grouping data
Selection, sorting, and grouping criteria that you define in the Select painter are added to the SQL statement and processed by the DBMS as part of the retrieval. You can also define selection, sorting, and grouping criteria that are created and processed dynamically by PowerBuilder *after* data has been retrieved from the DBMS.

For more information, see Chapter 23, "Filtering, Sorting, and Grouping Rows."

Referencing retrieval
arguments

If you have defined retrieval arguments, you reference them in the WHERE or
HAVING clause. In SQL statements, variables (called host variables) are always
prefaced with a colon to distinguish them from column names.

For example, if the DataWindow object is retrieving all rows from the
Department table where the dept_id matches a value provided by the user at
runtime, your WHERE clause will look something like this:

```
WHERE dept_id = :Entered_id
```

where Entered_id was defined previously as an argument in the Specify
Retrieval Arguments dialog box.

Referencing arrays

Use the IN operator and reference the retrieval argument in the WHERE or
HAVING clause.

For example, if you reference an array defined as deptarray, the expression in
the WHERE view might look like this:

```
"employee.dept_id" IN (:deptarray)
```

You need to supply the parentheses yourself.

Defining *WHERE*
criteria

You can limit the rows that are retrieved into the DataWindow object by
specifying selection criteria that correspond to the WHERE clause in the
SELECT statement.

For example, if you are retrieving information about employees, you can limit
the employees to those in Sales and Marketing, or to those in Sales and
Marketing who make more than $50,000.

❖ **To define *WHERE* criteria:**

1   Click the Where tab to make the Where view available (or select
    View>Where if the Where view is not currently displayed).

    Each row in the Where view is a place for entering an expression that
    limits the retrieval of rows.

2   Click in the first row under Column to display columns in a drop-down
    list, or select Columns from the pop-up menu.

3   Select the column you want to use in the left-hand side of the expression.

    The equality (=) operator displays in the Operator column.

Using a function or retrieval argument in the expression
To use a function, select Functions from the pop-up menu and click a listed function. These are the functions provided by the DBMS.

To use a retrieval argument, select Arguments from the pop-up menu. You must have defined a retrieval argument already.

---

4   (Optional) Change the default equality operator.

Enter the operator you want, or click to display a list of operators and select an operator.

5   Under Value, specify the right-hand side of the expression. You can:

- Type a value.

- Paste a column, function, or retrieval argument (if there is one) by selecting Columns, Functions, or Arguments from the pop-up menu.

- Paste a value from the database by selecting Value from the pop-up menu, then selecting a value from the list of values retrieved from the database. (It may take some time to display values if the column has many values in the database.)

- Define a nested SELECT statement by selecting Select from the pop-up menu. In the Nested Select dialog box, you can define a nested SELECT statement. Click Return when you have finished.

6   Continue to define additional WHERE expressions as needed.

For each additional expression, select a logical operator (AND or OR) to connect the multiple boolean expressions into one expression that PowerBuilder evaluates as true or false to limit the rows that are retrieved.

7   Define sorting (Sort view), grouping (Group view), and limiting (Having view) criteria as appropriate.

8   Click the Return button to return to the Report painter.

Defining *ORDER BY* criteria

You can sort the rows that are retrieved into the DataWindow object by specifying columns that correspond to the ORDER BY clause in the SELECT statement.

For example, if you are retrieving information about employees, you can sort on department, and then within each department, you can sort on employee ID.

❖ **To define *ORDER BY* criteria:**

1 Click the Sort tab to make the Sort view available (or select View>Sort if the Sort view is not currently displayed).

The columns you selected display in the order of selection. You might need to scroll to see your selections.

2 Drag the first column you want to sort on to the right side of the Sort view.

This specifies the column for the first level of sorting. By default, the column is sorted in ascending order. To specify descending order, clear the Ascending check box.

3 Continue to specify additional columns for sorting in ascending or descending order as needed.

You can change the sorting order by dragging the selected column names up or down. With the following sorting specification, rows will be sorted first by department name, then by employee ID:
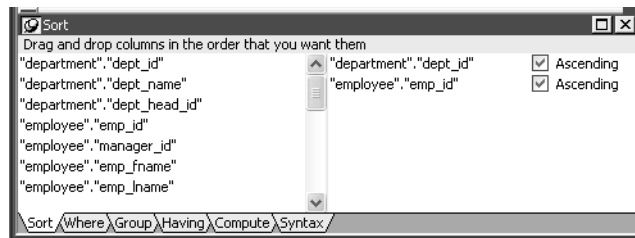
4 Define limiting (Where view), grouping (Group view), and limiting groups (Having view) criteria as appropriate.

5 Click the SQL Select button to return to the Report painter.

Defining *GROUP BY* criteria

You can group the retrieved rows by specifying groups that correspond to the GROUP BY clause in the SELECT statement. This grouping happens *before* the data is retrieved into the DataWindow object. Each group is retrieved as one row into the DataWindow object.

For example, if in the SELECT statement you group data from the Employee table by department ID, you will get one row back from the database for every department represented in the Employee table. You can also specify computed columns, such as total and average salary, for the grouped data. This is the corresponding SELECT statement:

```
SELECT dept_id, sum(salary), avg(salary)
FROM employee
GROUP BY dept_id
```

If you specify this with the Employee table in the EAS Demo DB, you get five rows back, one for each department.

Dept ID	Sum(salary)	Avg(salary)
100	$1,292,198	$58,736
200	$919,428	$48,391
300	$535,500	$59,500
400	$698,251	$43,641
500	$315,730	$35,081

For more about GROUP BY, see your DBMS documentation.

❖ **To define *GROUP BY* criteria:**

1   Click the Group tab to make the Group view available (or select View>Group if the Group view is not currently displayed).

The columns in the tables you selected display in the left side of the Group view. You might need to scroll to see your selections.

2   Drag the first column you want to group onto the right side of the Group view.

This specifies the column for grouping. Columns are grouped in the order in which they are displayed in the right side of the Group view.

3   Continue to specify additional columns for grouping within the first grouping column as needed.

To change the grouping order, drag the column names in the right side to the positions you want.

4   Define sorting (Sort view), limiting (Where view), and limiting groups (Having view) criteria as appropriate.

5   Click the Return button to return to the DataWindow painter.

Defining *HAVING* criteria

If you have defined groups, you can define HAVING criteria to restrict the retrieved groups. For example, if you group employees by department, you can restrict the retrieved groups to departments whose employees have an average salary of less than $50,000. This corresponds to:

```
SELECT dept_id, sum(salary), avg(salary)
FROM employee
GROUP BY dept_id
HAVING avg(salary) < 50000
```

If you specify this with the Employee table in the EAS Demo DB, you will get three rows back, because there are three departments that have average salaries less than $50,000.

Dept ID	Sum(salary)	Avg(salary)
200	$919,428	$48,391
400	$698,251	$43,641
500	$315,730	$35,081

❖ **To define *HAVING* criteria:**

• Click the Having tab to make the Having view available (or select View>Having if the Having view is not currently displayed).

Each row in the Having view is a place for entering an expression that limits which groups are retrieved. For information on how to define criteria in the Having view, see the procedure in "Defining WHERE criteria" on page 498.

Using Query

When you choose Query as the data source, you select a predefined SQL SELECT statement (a query) as specifying the data for your DataWindow object.

❖ **To define the data using Query:**

1 While using any of the DataWindow wizards, click Query in the Choose Data Source dialog box. Then click Next.

The Select Query dialog box displays.

2 Type the name of a query or use the Browse button to find the query. Then click Next.

3 Finish interacting with the DataWindow object wizard as needed for the presentation style you are using.

To learn how to create queries, see "Defining queries" on page 510.

Using External

If the data for the DataWindow object does not come from a database (either through a native Sybase database interface or through ODBC), specify External as the data source. You then specify the data columns and their types so PowerBuilder can build the appropriate DataWindow object to hold the data. These columns make up the result set. PowerBuilder places the columns you specified in the result set in the DataWindow object.

❖ **To define the data using External:**

1   Click External in the Choose Data Source dialog box in the wizard and click Next.

The Define Result Set dialog box displays for you to specify the first column in the result set.

2   Enter the name and type of the column.

Available datatypes are listed in the drop-down list.

3   Click Add to enter the name and type of any additional columns you want in the result set.

4   Click Next when you have added all the columns you want.

What you do next   In a script, you need to tell PowerBuilder how to get data into the DataWindow object in your application. Typically, you import data at runtime using a method (such as ImportFile and ImportString) or do some data manipulation and use the SetItem method to populate the DataWindow.

For more about these methods, see the online help.

You can also import data values from an external file into the DataWindow object or report.

❖ **To import the data values from an external file:**

1   Make sure the Preview view of the DataWindow object is selected.

2   Select Rows>Import from the menu bar.

The Select Import File dialog box displays.

3   Select the type of files to list from the List Files of Type drop-down list (an XML, CSV, TXT, or DBF file).

4   Enter the name of the import file and click OK.

Alternatively, you can select the name from the file list. Use the Drives drop-down list and the Directories box as needed to display the list of files that includes the one you want.

Using Stored Procedure

A stored procedure is a set of precompiled and preoptimized SQL statements that performs some database operation. Stored procedures reside where the database resides, and you can access them as needed.

Defining data using a stored procedure

You can specify a stored procedure as the data source for a DataWindow object if your DBMS supports stored procedures.

For information on support for stored procedures, see your database documentation.

If the Stored Procedure icon is not displayed
The icon for the Stored Procedure data source displays in the Choose Data Source dialog box in the DataWindow object wizards only if the database to which you are connected supports stored procedures.

❖ **To define the data using Stored Procedure:**

1   Select Stored Procedure in the Choose Data Source dialog box in the wizard and click Next.

The Select Stored Procedure dialog box displays a list of the stored procedures in the current database.

2   Select a stored procedure from the list.

To list system procedures, select the System Procedure check box.

The syntax of the selected stored procedure displays below the list of stored procedures.

3   Specify how you want the result set description built:

•   To build the result set description automatically, clear the Manual Result Set check box and click Next.

PowerBuilder executes the stored procedure and builds the result set description for you.

• To define the result set description manually, select the Manual Result Set check box and click Next.

In the Define Stored Procedure Result Set dialog box:

• Enter the name and type of the first column in the result set.

• To add additional columns, click Add.

Your preference is saved
PowerBuilder records your preference for building result set descriptions for stored procedure DataWindow objects in the variable *Stored_Procedure_Build* in the PowerBuilder initialization file. If this variable is set to 1, PowerBuilder will automatically build the result set; if the variable is set to 0, you are prompted to define the result set description.

---

4 Continue in the DataWindow object wizard as needed for the presentation style you are using.

When you have finished interacting with the wizard, you go to the Report painter with the columns specified in the result set placed in the DataWindow object.

For information about defining retrieval arguments for DataWindow objects, see Chapter 19, "Enhancing DataWindow Objects."

For information about using a stored procedure to update the database, see "Using stored procedures to update the database" on page 592.

Editing a result set description

After you create a result set that uses a stored procedure, you can edit the result set description from the DataWindow painter.

❖ **To edit the result set description:**

1 Select Design>Data Source from the menu bar.

This displays the Column Specification view if it is not already displayed.

2 Select Stored Procedure from the Column Specification view's pop-up menu.

The Modify Stored Procedure dialog box displays.

3 Edit the Execute statement, select another stored procedure, or add arguments.

The syntax is:

> execute *sp_procname*;*num arg1* = :*arg1*, *arg2* = :*arg2*..., *argn* =:*argn*

where sp_procname is the name of the stored procedure, *num* is the stored procedure group suffix, and *arg1*, *arg2*, and *argn* are the stored procedure's arguments.

The group suffix is an optional integer used in some DBMSs to group procedures of the same name so that they can be dropped together with a single DROP PROCEDURE statement. For other DBMSs the number is ignored.

4   When you have defined the entire result set, click OK.

You return to the Report painter with the columns specified in the result set placed in the DataWindow object.

For information about defining retrieval arguments for DataWindow objects, see Chapter 19, "Enhancing DataWindow Objects."

Choosing DataWindow object-wide options

You can set the default options, such as colors and borders, that PowerBuilder uses in creating the initial draft of a DataWindow object.

Report generation options are for styles that use a layout made up of bands, which include Freeform, Grid, Label, N-Up, Tabular, Group, TreeView, and Crosstab. PowerBuilder maintains a separate set of options for each of these styles.

When you first create any of these style DataWindow objects, you can choose options in the wizard and save your choices as the future defaults for the style.

❖ **To specify default colors and borders for a style:**

1   Select Design>Options from the menu bar.

The Report Options dialog box displays.

2   Select the Generation tab page if it is not on top.

3   Select the presentation style you want from the Presentation Style drop-down list.

The values for properties shown on the page are for the currently selected presentation style.

4   Change one or more of the following properties:

Property	Meaning for the DataWindow object
Background color	The default color for the background.
Text border and color	The default border and color used for labels and headings.
Column border and color	The default border and color used for data values.
Wrap Height (Freeform only)	The height of the detail band.
	When the value is None, the number of columns selected determines the height of the detail band. The columns display in a single vertical line.
	When the value is set to a number, the detail band height is set to the number specified and columns wrap within the detail band.

5   Click OK.

About color selections
If you select Window Background, Application Workspace, Button Face, or Window Text from the Color drop-down list, the DataWindow object uses the colors specified in the Windows Control Panel on the computer on which the DataWindow object is running.

Your choices are saved

PowerBuilder saves your generation option choices as the defaults to use when creating a DataWindow object with the same presentation style.

Generating and saving a DataWindow object

When you have finished interacting with the wizard, PowerBuilder generates the DataWindow object and opens the Report painter.

When generating the DataWindow object, PowerBuilder might use information from a set of tables called the extended attribute system tables. If this information is available, PowerBuilder uses it.

About the extended attribute system tables and DataWindow objects

The extended attribute system tables are a set of tables maintained by the Database painter. They contain information about database tables and columns. Extended attribute information extends database definitions by recording information that is relevant to using database data in screens and reports.

For example, labels and headings you defined for columns in the Database painter are used in the generated DataWindow object. Similarly, if you associated an edit style with a column in the Database painter, that edit style is automatically used for the column in the DataWindow object.

When generating a DataWindow object, PowerBuilder uses the following information from the extended attribute system tables:

For	PowerBuilder uses
Tables	Fonts specified for labels, headings, and data
Columns	Text specified for labels and headings Display formats Validation rules Edit styles

If there is no extended attribute information for the database tables and columns you are using, you can set the text for headings and labels, the fonts, and the display formats in the Report painter. The difference is that you have to do this individually for every DataWindow object that you create using the data.

If you want to change something that came from the extended attribute system tables, you can change it in the Report painter. The changes you make in the Report painter apply only to the DataWindow object you are working on.

The advantage of using the extended attribute system tables is that it saves time and ensures consistency. You only have to specify the information once, in the database. Since PowerBuilder uses the information whenever anyone creates a new DataWindow object with the data, it is more likely that the appearance and labels of data items will be consistent.

For more information about the extended attribute system tables, see Chapter 16, "Managing the Database," and Appendix A, "The Extended Attribute System Tables."

Saving the DataWindow object

When you have created a DataWindow object, you should save it. The first time you save it you give it a name. As you work, you should save your DataWindow object frequently so that you do not lose changes.

❖   **To save the DataWindow object:**

1   Select File>Save from the menu bar.

   If you have previously saved the DataWindow object, PowerBuilder saves the new version in the same library and returns you to the Report painter.

   If you have not previously saved the DataWindow object, PowerBuilder displays the Save Report dialog box.

2   (Optional) Enter comments in the Comments box to describe the DataWindow object.

3   Enter a name for the DataWindow object in the Reports box.

4   Specify the library in which the DataWindow object is to be saved and click OK.

Naming the DataWindow object

The DataWindow object name can be any valid PowerBuilder identifier up to 40 contiguous characters. A common convention is to prefix the name of the DataWindow object with d_.

For information about PowerBuilder identifiers, see the *PowerScript Reference*.

Modifying an existing DataWindow object

❖   **To modify an existing DataWindow object:**

1   Select File>Open from the menu bar.

   The Open dialog displays.

2   Select the object type and the library.

   PowerBuilder lists the DataWindow objects in the current library.

3   Select the object you want.

PowerBuilder opens the Report painter and displays the DataWindow object. You can also open a DataWindow object by double-clicking it in the System Tree, or, if it has been placed in a window or visual user object, selecting Modify DataWindow from the control's pop-up menu.

To learn how you can modify an existing DataWindow object, see Chapter 19, "Enhancing DataWindow Objects."

Defining queries

A query is a SQL SELECT statement created in the Query painter and saved with a name so that it can be used repeatedly as the data source for a DataWindow object.

Queries save time, because you specify all the data requirements just once. For example, you can specify the columns, which rows to retrieve, and the sorting order in a query. Whenever you want to create a DataWindow object using that data, simply specify the query as the data source.

❖   **To define a query:**

1   Select File>New from the menu bar.

2   In the New dialog box, select the Database tab.

3   Select the Query icon and click OK.

4   Select tables in the Select Tables dialog box and click Open.

You can select columns, define sorting and grouping criteria, define computed columns, and so on, exactly as you do when creating a DataWindow object using the SQL Select data source.

For more about defining the SELECT statement, see "Using SQL Select" on page 486.

Previewing the query

While creating a query, you can preview it to make sure it is retrieving the correct rows and columns.

❖ **To preview a query:**

1   Select Design>Preview from the menu bar.

    PowerBuilder retrieves the rows satisfying the currently defined query in a grid-style DataWindow object.

2   Manipulate the retrieved data as you do in the Database painter in the Output view.

    You can sort and filter the data, but you cannot insert or delete a row or apply changes to the database. For more about manipulating data, see Chapter 16, "Managing the Database."

3   When you have finished previewing the query, click the Close button in the PainterBar to return to the Query painter workspace.

Saving the query

❖ **To save a query:**

1   Select File>Save Query from the menu bar.

    If you have previously saved the query, PowerBuilder saves the new version in the same library and returns you to the Query painter. If you have not previously saved the query, PowerBuilder displays the Save Query dialog box.

2   Enter a name for the query in the Queries box (see "Naming the query" next).

3   (Optional) Enter comments to describe the query,.

    These comments display in the Library painter. It is a good idea to use comments to remind yourself and others of the purpose of the query.

4   Specify the library in which to save the query, and click OK.

Naming the query

The query name can be any valid PowerBuilder identifier up to 40 characters. When you name queries, use a unique name to identify each one. A common convention is to use a two-part name: a standard prefix that identifies the object as a query (such as q_) and a unique suffix. For example, you might name a query that displays employee data q_emp_data. For information about PowerBuilder identifiers, see the *PowerScript Reference*.

Modifying a query

❖ **To modify a query:**

1 Select File>Open from the menu bar.

2 Select the Queries object type and then the query you want to modify, and click OK.

3 Modify the query as needed.

What's next

After you have generated your DataWindow object, you will probably want to preview it to see how it looks. After that, you might want to enhance the DataWindow object in the Report painter before using it. PowerBuilder provides many ways for you to make a DataWindow object easier to use and more informative for users. See Chapter 19, "Enhancing DataWindow Objects," next.

CHAPTER 19 **Enhancing DataWindow Objects**

About this chapter

Before you put a DataWindow object into production, you can enhance it to make it easier to use and interpret data. You do that in the DataWindow painter. This chapter describes basic enhancements you can make to a DataWindow object.

Contents

Topic	Page
Working in the DataWindow painter	514
Using the Preview view of a DataWindow object	522
Saving data in an external file	533
Modifying general DataWindow object properties	540
Storing data in a DataWindow object using the Data view	553
Retrieving data	555

Related topics

Other ways to enhance DataWindow objects are covered in later chapters:

Chapter	Explains how to
Chapter 20, "Working with Controls in DataWindow Objects"	Add controls to a DataWindow object and reorganize, position, and rotate them
Chapter 21, "Controlling Updates in DataWindow Objects"	Control update capabilities
Chapter 22, "Displaying and Validating Data"	Specify display formats, edit styles, and validation rules for column data
Chapter 23, "Filtering, Sorting, and Grouping Rows"	Limit which rows are displayed, the order in which they are displayed, and whether they are divided into groups
Chapter 24, "Highlighting Information in DataWindow Objects"	Highlight data by using conditional expressions to modify the properties of controls in DataWindow objects
Chapter 25, "Using Nested Reports"	Place reports inside DataWindow objects
Chapter 26, "Working with Graphs"	Use graphs to visually present information retrieved in a DataWindow object

Chapter	Explains how to
Chapter 27, "Working with Crosstabs"	Use crosstabs to present analyses of data retrieved in a DataWindow object
Chapter 28, "Working with TreeViews"	Use TreeViews to group data and display it hierarchically in a way that allows you to expand and collapse it

Working in the DataWindow painter

The DataWindow painter provides views related to the DataWindow object you are working on. Interacting with these views is how you work in the DataWindow painter.

The following picture shows a DataWindow object in the DataWindow painter with the default layout.
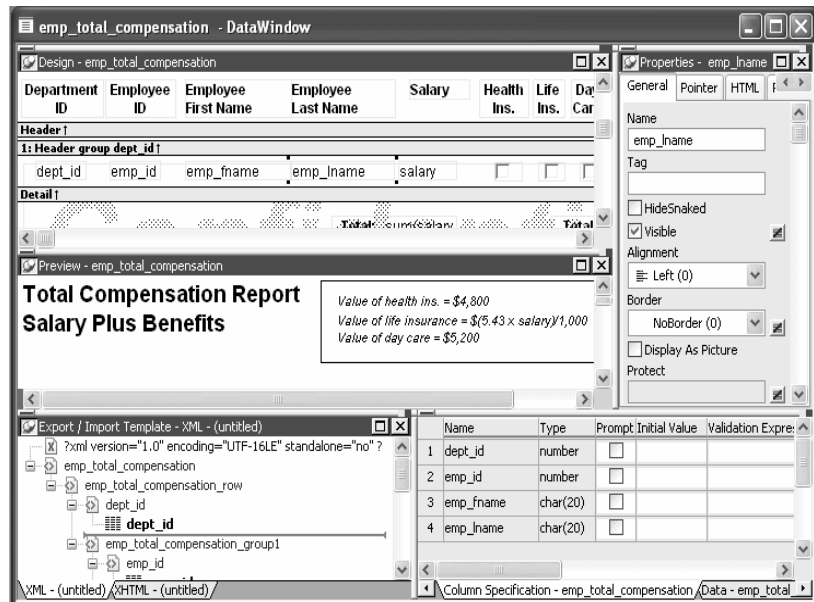
Design view

The Design view at the top left shows a representation of the DataWindow object and its controls. You use this view to design the layout and appearance of the DataWindow object. Changes you make are immediately shown in the Preview view and the Properties view.

Preview view	The Preview view in the middle on the left shows the DataWindow object with data as it will appear at runtime. If the Print Preview toggle is selected, you see the DataWindow object as it would appear when printed.
Export/Import Template view for XML	The Export/Import Template view for XML at the bottom left shows a default template for exporting and importing data in XML format. You can define custom templates for import and export. The templates are saved with the DataWindow object. For more information, see Chapter 29, "Exporting and Importing XML Data."
Export Template view for XHTML	The Export Template view for XHTML (not shown; see XHTML tab at the bottom left) shows a default template for exporting data in XHTML format. You can define custom XHTML export templates for customizing XML Web DataWindow generation. The templates are saved with the DataWindow object. For more information, see the *DataWindow Programmer's Guide*.
Properties view	The Properties view at the top right displays the properties for the currently selected control(s) in the DataWindow object, for the currently selected band in the DataWindow object, or for the DataWindow object itself. You can view and change the values of properties in this view.
Control List view	The Control List view in the stacked pane at the bottom right lists all controls in the DataWindow object. Selecting controls in this view selects them in the Design view and the Properties view. You can also sort controls by Control Name, Type, or Tag.
Data view	The Data view in the stacked pane at the bottom right displays the data that can be used to populate a DataWindow object and allows manipulation of that data.
Column Specifications view	The Column Specifications view in the stacked pane at the bottom right shows a list of the columns in the data source. For the columns, you can add, modify, and delete initial values, validation expressions, and validation messages. You can also specify that you want a column to be included in a prompt for retrieval criteria during data retrieval. To add a column to the DataWindow object, you can drag and drop the column from the Column Specifications view to the Design view. For external or stored procedure data sources, you can add, delete, and edit columns (column name, type, and length).

Understanding the DataWindow painter Design view

For most presentation styles, the DataWindow painter Design view is divided into areas called bands. Each band corresponds to a section of the displayed DataWindow object.

DataWindow objects with these presentation styles are divided into four bands: header, detail, summary, and footer. Each band is identified by a bar containing the name of the band above the bar and an Arrow pointing to the band.

These bands can contain any information you want, including text, drawing controls, graphs, and computed fields containing aggregate totals.

The following picture shows the Design view for a tabular DataWindow object.
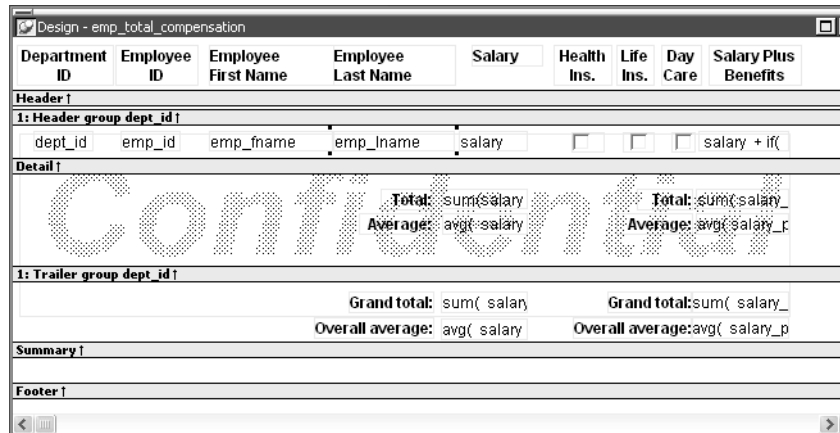
Table 19-1: Bands in the DataWindow painter Design view

Band	Used to display
Header	Information at the top of every screen or page, such as the name of the report or current date
Detail	Data from the database or other data source
Summary	Summary information that displays after all the data, such as totals and counts
Footer	Information displayed at the bottom of every page or screen, such as page number and page count

The header band

The header band contains heading information that is displayed at the top of every screen or page. The presentation style determines the contents of the header band:

- If the presentation style is Tabular, Grid, or N-Up, the headings defined for the columns in the Database painter display in the header band and the columns display on a single line across the detail band

- If the presentation style is Freeform, the header band is empty and labels display in the detail band next to each column

You can specify additional heading information (such as a date) in the header band and you can include pictures, graphic controls, and color to enhance the appearance of the band.

Displaying the current date
To include the current date in the header, you place a computed field that uses the Today DataWindow expression function in the header band. For information, see "Adding computed fields to a DataWindow object" on page 563.

The detail band

The detail band displays the retrieved data. It is also where the user enters new data and updates existing data. The number of rows of data that display in the DataWindow object at one time is determined by the following expression:

(Height of the DataWindow object *– Height of headers and footers) / Height of the detail band*

The presentation style determines the contents of the detail band:

- If the presentation style is Tabular, Grid, N-Up, or Label, the detail band displays column names, representing the columns

- If the presentation style is Freeform, the labels defined for the columns in the Database painter display in the detail band with boxes for the data to the right

---

How PowerBuilder names the columns in the Design view
If the DataWindow object uses one table, the names of the columns in the Design view are the same as the names in the table.

If the DataWindow object uses more than one table, the names of the columns in the Design view are *tablename_columnname*. PowerBuilder prefaces the name of the column with the table name to prevent ambiguity, since different tables can have columns with the same name.

When you design the detail band of a DataWindow object, you can specify display and validation information for each column of the DataWindow object and add other controls, such as text, pictures, graphs, and drawing controls.

The summary and footer bands

You use the summary and footer bands of the DataWindow object the same way you use summary pages and page footers in a printed report:

- The contents of the summary band display at the end, after all the detail rows; this band often summarizes information in the DataWindow object

- The contents of the footer band display at the bottom of each screen or page of the DataWindow object; this band often displays the page number and name of the report

Using the DataWindow painter toolbars

The DataWindow painter contains three customizable PainterBars and a StyleBar.

For more information about using toolbars, see "Using toolbars" on page 46.

PainterBars

The PainterBars include buttons for standard operations (such as Save, Print, and Undo on PainterBar1), for common formatting operations (such as Currency, Percent, and Tab Order on PainterBar2), and for database operations (such as Retrieve and Insert Row on PainterBar3).

They also include six drop-down toolbars, which are indicated by a small black triangle on the right part of a button. Table 19-2 lists the drop-down toolbars that are available. The Controls toolbar is on PainterBar1. The other drop-down toolbars are on PainterBar2.

Table 19-2: Drop-down toolbars in the DataWindow painter

Toolbar	Used to
Background Color	Specify the background color of one or more selected controls.
Borders	Specify borders for one or more selected controls.
Controls	Specify controls to add to a DataWindow object.
Foreground Color	Specify the foreground color of one or more selected controls. In a text control, the foreground color specifies the color of the text.
Layout	Specify the alignment, sizing, and spacing of selected controls.
Slide	Specify sliding for controls.

StyleBar

The StyleBar includes buttons for applying properties (such as bold) to selected text elements.

Using the Properties view in the DataWindow painter

Each part of the DataWindow object (such as text, columns, computed fields, bands, graphs, even the DataWindow object itself) has a set of properties appropriate to the part. The properties display in the Properties view.

You can use the Properties view to modify the parts of the DataWindow object.

❖ **To use the Properties view to modify the parts of the DataWindow object:**

1   Position the mouse over the part you want to modify.

2   Display the part's pop-up menu and select Properties.

If it is not already displayed, the Properties view displays. The view displays the properties of the currently selected control(s), the band, or the DataWindow object itself. The contents of the Properties view change as different controls are selected (made current).

For example, the Properties view for a column has seven tabbed property pages of information that you access by clicking the appropriate tab. If you want to choose an edit style for the column, you click the Edit tab. This brings the Edit page to the front of the Properties view.

Selecting controls in the DataWindow painter

The DataWindow painter provides several ways to select controls to act on. You can select multiple controls and act on all the selected controls as a unit. For example, you can move all of them or change the fonts used to display text for all of them.

Lasso selection
Use lasso selection when possible because it is fast and easy. Lasso selection is another name for the method described below for selecting neighboring multiple controls.

❖ **To select one control in a DataWindow object in the Design view:**

- Click it.

  The control displays with handles on it. Previously selected controls are no longer selected.

❖ **To select neighboring multiple controls in a DataWindow object in the Design view (lasso selection):**

1 Press and hold the left mouse button at one corner of the neighboring controls.

2 Drag the mouse over the controls you want to select.

  A bounding box (the lasso) displays.

3 Release the mouse button.

  All the controls in the bounding box are selected.

❖ **To select non-neighboring multiple controls in a DataWindow object in the Design view:**

1 Click the first control.

2 Press and hold the Ctrl key and click additional controls.

  All the controls you click are selected.

❖ **To select controls by type in the DataWindow object:**

- Do one of the following:

  - Select Edit>Select>Select All to select all controls

  - Select Edit>Select>Select Text to select all text

  - Select Edit>Select>Select Columns to select all columns

❖ **To select controls by position in the DataWindow object:**

- Do one of the following:

    - Select Edit>Select>Select Above to select all controls above the currently selected control

    - Select Edit>Select>Select Below to select all controls below it

    - Select Edit>Select>Select Left to select all controls to the left of it

    - Select Edit>Select>Select Right to select all controls to the right of it

❖ **To select controls in a DataWindow object in the Control List view:**

1   Select View>Control List from the menu bar.

2   Click a control in the list.

3   Press and hold the Ctrl key and click additional controls if desired.

Displaying information about the selected control

The name, x and y coordinates, width, and height of the selected control are displayed in the MicroHelp bar. If multiple controls are selected, *Group Selected* displays in the Name area and the coordinates and size do not display.

Resizing bands in the DataWindow painter Design view

You can change the size of any band in the DataWindow object.

❖ **To resize a band in the DataWindow painter Design view:**

- Position the pointer on the bar representing the band and drag the bar up or down to shrink or enlarge the band.

Using zoom in the DataWindow painter

You can zoom the display in and out in four views in the DataWindow painter: the Design view, Preview view, Data View, and Column Specifications view. For example, if you are working with a large DataWindow object, you can zoom out the Design view so you can see all of it on your screen, or you can zoom in on a group of controls to better see their details.

❖ **To zoom the display in the DataWindow painter:**

1   Select the view you want to zoom (click in the view).

    You can zoom the Design view, Preview view, Data View, and Column Specifications view.

2   Select Design>Zoom from the menu bar.

3   Select a built-in zoom percentage, or set a custom zoom percentage by typing an integer in the Custom box.

Using the IntelliMouse pointing device
Using the IntelliMouse pointing device, users can also zoom a view larger or smaller by holding down the Ctrl key while rotating the wheel.

Undoing changes in the DataWindow painter

You can undo your change by pressing Ctrl+Z or selecting Edit>Undo from the menu bar. Undo requests affect all views.

Using the Preview view of a DataWindow object

You use the Preview view of a DataWindow object to view it as it will appear with data and test the processing that takes place in it.

❖ **To display the Preview view of a DataWindow object open in the DataWindow painter:**

1   If the Preview view is not already displayed, select View>Preview from the menu bar.

    In the Preview view, the bars that indicate the bands do not display, and, if you selected Retrieve on Preview in the DataWindow wizard, PowerBuilder retrieves all the rows from the database. You are prompted to supply arguments if you defined retrieval arguments.

In external DataWindow objects
If the DataWindow object uses the External data source, no data is retrieved. You can import data, as described in "Importing data into a DataWindow object" on page 527.

> **In DataWindow objects that have stored data**
> If the DataWindow object has stored data in it, no data is retrieved from the database.

As the rows are being retrieved, the Retrieve button in the PainterBar changes to a Cancel button. You can click the Cancel button to stop the retrieval.

2   Test your DataWindow object.

For example, modify some data, update the database, re-retrieve rows, and so on, as described below.

Retrieving data

Where PowerBuilder gets data

PowerBuilder follows this order of precedence to supply the data in your DataWindow object:

1   If you have saved data in the DataWindow object, PowerBuilder uses the saved rows from the DataWindow object and does not retrieve data from the database.

2   PowerBuilder uses the data in the cache, if there is any.

3   If there is no data in the cache yet, PowerBuilder retrieves data from the database automatically, with one exception. If the Retrieve on Preview option is off, you have to request retrieval explicitly, as described next.

Previewing without retrieving data

If you do not want PowerBuilder to retrieve data from the database automatically when the Preview view opens, you can set the Retrieve on Preview option. The Preview view shows the DataWindow object without retrieving data.

❖   **To be able to preview without retrieving data automatically:**

1   Select Design>Options from the menu bar.

The Report Options dialog box displays.

2   Clear the Retrieve on Preview check box on the General page.

When this check box is cleared, your request to preview the DataWindow object does not result in automatic data retrieval from the database.

PowerBuilder uses data caching

When PowerBuilder first retrieves data, it stores the data internally. When it refreshes the Preview view, PowerBuilder displays the stored data instead of retrieving rows from the database again. This can save you a lot of time, since data retrieval can be time consuming.

How using data from the cache affects you

Because PowerBuilder accesses the cache and does not automatically retrieve data every time you preview, you might not have what you want when you preview. The data you see in preview and the data in the database can be out of sync.

For example, if you are working with live data that changes frequently or with statistics based on changing data and you spend time designing the DataWindow object, the data you are looking at may no longer match the database. In this case, retrieve again just before printing.

Explicitly retrieving data

You can explicitly request retrieval at any time.

❖ **To retrieve rows from the database:**

- Do one of the following:

  - Click the Retrieve button in the PainterBar.

  - Select Rows>Retrieve from the menu bar.

  - Select Retrieve from the Preview view's pop-up menu.

  Supplying argument values or criteria
  If the DataWindow object has retrieval arguments or is set up to prompt for criteria, you are prompted to supply values for the arguments or to specify criteria.

  PowerBuilder retrieves the rows. As PowerBuilder retrieves, the Retrieve button changes to a Cancel button. You can click the Cancel button to stop the retrieval at any time.

Sharing data with the Data view

The Data view displays data that can be used to populate a DataWindow object. When the ShareData pop-up menu item in the Data view is checked, changes you make in the Data view are reflected in the Preview view and vice versa.

Other options that affect retrieval

These other options can affect retrieval:

- **Retrieve Rows As Needed**   Lets you specify that only the rows needed to display the current portion of the DataWindow object should be retrieved. When you scroll downward, additional rows are retrieved. This can speed up reporting in certain situations.

  See "Retrieving rows as needed" on page 557.

- **Retrieve Rows to Disk**   Lets you specify that PowerBuilder should save retrieved data on your hard disk in a temporary file rather than keep the data in memory. When you preview the DataWindow object, PowerBuilder swaps rows of data from the temporary file into memory as needed.

  For information, see "Saving retrieved rows to disk" on page 557.

Modifying data

You can add, modify, or delete rows in the Preview view. When you have finished manipulating the data, you can apply the changes to the database.

If looking at data from a view or from more than one table
By default, you cannot update data in a DataWindow object that contains a view or more than one table. For more about updating DataWindow objects, see Chapter 21, "Controlling Updates in DataWindow Objects."

❖ **To modify existing data:**

- Tab to the field and enter a new value.

  The Preview view uses validation rules, display formats, and edit styles that you have defined for the columns, either in the Database painter or in this particular DataWindow object.

To save the changes to the database, you must apply them, as described below.

❖ **To add a row:**

1   Click the Insert Row button.

    PowerBuilder creates a blank row.

2   Enter data for a row.

To save the changes to the database, you must apply them, as described below.

Adding a row in an application
Clicking the Insert Row button in the Preview view is equivalent to calling the
InsertRow method and then the ScrollToRow method at runtime.

❖ **To delete a row:**

• Click the Delete Row button.

PowerBuilder removes the row from the display.

To save the changes to the database, you must apply them, as described below.

Deleting a row in an application
Clicking the Delete Row button in the Preview view is equivalent to calling the
DeleteRow method at runtime.

❖ **To apply changes to the database:**

• Click the Update Database button.

PowerBuilder updates the table with all the changes you have made.

Applying changes in an application
Clicking the Update Database button in the Preview view is equivalent to
calling the Update method at runtime.

Viewing row information

You can display information about the data you have retrieved.

❖ **To display the row information:**

• Select Rows>Described from the menu bar.

The Describe Rows dialog box displays, showing the number of:

• Rows that have been deleted in the painter *but not yet deleted from the
database*

• Rows displayed in the Preview view

- Rows that have been filtered

- Rows that have been modified in the painter *but not yet modified in the database*

All row counts are zero until you retrieve the data from the database or add a new row. The count changes when you modify the displayed data or test filter criteria.

Importing data into a DataWindow object

You can import and display data from an external source and save the imported data in the database.

❖ **To import data into a DataWindow object:**

1 Select Rows>Import from the menu bar.

2 Specify the file from which you want to import the data.

The types of files that you can import into the painter display in the List Files of Type drop-down list.

3 Click Open.

PowerBuilder reads the data from the file into the painter. You can then click the Update Database button in the PainterBar to add the new rows to the database.

Data from file must match DataWindow definition
When importing data from a file, the datatypes of the data must match, column for column, all the columns in the DataWindow definition (the columns specified in the SELECT statement), not just the columns that are displayed in the DataWindow object.

For information about importing XML data, see Chapter 29, "Exporting and Importing XML Data."

Using print preview

You can print the data displayed in the Preview view. Before printing, you can preview the output on the screen. Your computer must have a default printer specified, otherwise properties handled by the printer driver, such as page orientation, are ignored.

❖ **To preview printed output before printing:**

- Be sure the Preview view is selected (current) and then select File>Print Preview from the menu bar.

 Print Preview displays the DataWindow object as it will print.

Using the IntelliMouse pointing device
Using the IntelliMouse pointing device, users can scroll a DataWindow object by rotating the wheel. Users can also zoom a DataWindow object larger or smaller by holding down the Ctrl key while rotating the wheel.

Controlling the display of rulers

You can choose whether to display rulers around page borders.

❖ **To control the display of rulers in Print Preview:**

- Select/deselect File>Print Preview Rulers from the menu bar.

Changing margins

You can dynamically change margins while previewing a DataWindow object.

❖ **To change the margins in Print Preview:**

- Drag the margin boundaries on the rulers.

The following picture shows the left and top margin boundaries. There are also boundaries for the right and bottom margins. The picture shows the outline of the margin. If you do not want to see the outline, clear the Print Preview Shows Outline check box on the Print Specifications page in the Properties view.
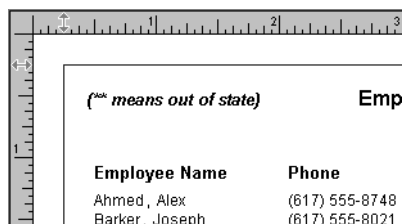
Changing margins at runtime
Using the Modify method, you can display a DataWindow object in print preview at runtime. While in print preview, users can also change margins by dragging boundaries. A user event in the DataWindow control (pbm_dwnprintmarginchange) is triggered when print margins are changed. Changing margins can affect the page count, so if you use the Describe method to display the page count in your application (for example, in MicroHelp), you must code a script for the user event to recalculate the page count.

Zooming the page

You can reduce or enlarge the amount of the page that displays in the Print Preview view. This does not affect the printed output.

❖ **To zoom the page on the display screen:**

1   Select File>Print Preview Zoom from the menu bar.

2   Select the magnification you want and click OK.

The display of the page zooms in or out as appropriate. The size of the contents of the page changes proportionately as you zoom. This type of zooming affects your display but does not affect printing.

Zooming the contents

In addition to zooming the display on the screen, you can also zoom the contents, affecting the amount of material that prints on a page.

❖ **To zoom the contents of a DataWindow object with respect to the printed page:**

1   Select Design>Zoom from the menu bar.

2   Select the magnification you want and click OK.

The contents of the page zooms in or out as appropriate. If you enlarge the contents so they no longer fit, PowerBuilder creates additional pages as needed.

Printing data

You can print a DataWindow object while the Preview view is displayed. You can print all pages, a range of pages, only the current page, or only odd or even pages. You can also specify whether you want multiple copies, collated copies, and printing to a file.

Avoiding large rows

To avoid multiple blank pages and other anomalies in printed reports, no row in the DataWindow object should be larger than the size of the target page. The page boundary is often reached in long text columns with AutoSizeHeight on. It can also be reached when detail rows are combined with page and group headers and trailers, or when they contain multiple nested DataWindow objects or a column that has been resized to be larger than the page.

When a row contains large multiline edit columns, it can be broken into a series of rows, each containing one text line. These text lines become the source for a nested DataWindow object. The nested DataWindow object determines how many of its rows fit in the remaining page space.

Page break before last row

The summary band in a report is always printed on the same page as the last row of data. This means that you sometimes find a page break before the last row of data even if there is sufficient space to print the row. If you want the last row to print on the same page as the preceding rows, the summary band must be made small enough to fit on the page as well.

To change printers or settings before printing

You can choose File>Printer Setup from the menu bar.

❖ **To print a DataWindow object:**

1 Select File>Print Report from the menu bar to display the Print dialog box.

2 Specify the number of copies to print.

3 Specify the pages: select All or Current Page, or type page numbers and/or page ranges in the Pages box.

4 Specify all pages, even pages, or odd pages in the Print drop-down list.

5 If you want to print to a file rather than to the printer, select the Print to File check box.

6 If you want to change the collating option, clear or select the Collate Copies check box and click OK.

If you specified print to file, the Print to File dialog box displays.

7 Enter a file name and click OK.

The extension *PRN* indicates that the file is prepared for the printer. Change the drive, the directory, or both, if you want.

Working in a grid DataWindow object

If you are viewing a grid-style DataWindow object in the Preview view, you can make the following changes. Whatever you do in the Preview view is reflected in the Design view:

- Resize columns

- Reorder columns

- Split the display into two horizontal scrolling regions

  You can use this feature to keep one or more columns stationary on the screen while scrolling through other columns.

- Copy data to the clipboard

These features are also available to your users
Users of your application can also manipulate columns in these ways in a grid DataWindow object at runtime.

❖ **To resize a column in a grid DataWindow object:**

1   Position the mouse pointer at a column boundary in the header.

    The pointer changes to a two-headed arrow.

2   Press and hold the left mouse button and drag the mouse to move the boundary.

3   Release the mouse button when the column is the correct width.

❖ **To reorder columns in a grid DataWindow object:**

1   Press and hold the left mouse button on a column heading.

    PowerBuilder selects the column and displays a line representing the column border.

2   Drag the mouse left or right to move the column.

3   Release the mouse button.

❖ **To use split horizontal scrolling in a grid DataWindow object:**

1 To divide the grid into two regions that can scroll independently of each other, position the mouse pointer at the left end of the horizontal scrollbar.
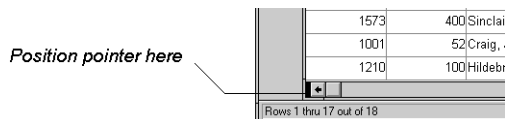
Position pointer here

1573	400 Sinclai
1001	52 Craig,
1210	100 Hildebr

Rows 1 thru 17 out of 18

The pointer changes to a two-headed arrow.

2 Press and hold the left mouse button and drag the mouse to the right to create a new horizontal scrolling border.

3 Release the mouse button.

You now have two independent scrolling regions in the grid DataWindow object.

❖ **To copy data to the clipboard from a grid DataWindow object:**

1 Select the cells whose data you want to copy to the clipboard:

- To select an entire column, click its header.

- To select neighboring columns, press and hold Shift, then click the headers.

- To select non-neighboring columns, press and hold Ctrl, then click the headers.

- To select cells, press the left mouse button on the bottom border of a cell and drag the mouse.

Selected cells are highlighted.

2 Select Edit>Copy from the menu bar.

The contents of the selected cells are copied to the clipboard. If you copied the contents of more than one column, the data is separated by tabs.

Saving data in an external file

While previewing, you can save the data retrieved in an external file. Note that the data and headers (if specified) are saved. Information in the footer or summary bands is not saved unless you are saving as PDF or as a PSR file.

❖ **To save the data in a DataWindow object in an external file:**

1   Select File>Save Rows As from the menu bar.

The Save As dialog box displays.

2   Choose a format for the file from the Save As Type drop-down list.

If you want the column headers saved in the file, select a file format that includes headers (such as Excel With Headers). When you select a *with headers* format, the names of the database columns (not the column labels) are also saved in the file.

When you choose a format, PowerBuilder supplies the appropriate file extension.

3   For TEXT, CSV, SQL, HTML, and DIF formats, select an encoding for the file.

You can select ANSI/DBCS, Unicode LE (Little-Endian), Unicode BE (Big-Endian), or UTF8.

4   Name the file and click Save.

PowerBuilder saves all displayed rows in the file; all columns in the displayed rows are saved. Filtered rows are not saved.

The rest of this section provides more information about saving data in PDF, HTML, and Powersoft Report (PSR) formats.

For more information about saving data as XML, see Chapter 29, "Exporting and Importing XML Data."

Saving the data as PDF

PowerBuilder provides two ways to save a DataWindow object or DataStore in Portable Document Format (PDF).

Using GNU Ghostscript

By default, when you select File>Save Rows As and select PDF as the file type, the data is printed to a PostScript file and automatically distilled to PDF using GNU Ghostscript. This option provides a robust solution that can save most types of DataWindow objects.

Installing Ghostscript
For licensing reasons, Ghostscript is not installed with PowerBuilder. You (and your users) must download and install it before you can use this technique. See "System requirements for the distill method" on page 535.

Using XSL-FO and Java printing

Building on the ability to save data as XML, PowerBuilder can also save the DataWindow object's data and presentation to PDF by generating XSL Formatting Objects (XSL-FO). This option provides a platform-independent solution by rendering the DataWindow using a Java process rather than the Microsoft GDI. It also offers the possibility of customizing the PDF file at the XSL-FO stage. Saving as PDF using XSL-FO is particularly useful if you want to print DataWindow objects in EAServer on a UNIX operating system by using Java printing. The Ghostscript method is not supported on UNIX.

The XSL (Extensible Stylesheet Language) W3C Recommendation has two parts, XSLT and XSL-FO. XSLT provides the transformation typically used to present XML documents as HTML in a browser. XSL-FO provides extensive formatting capabilities that are not dependent on the output format.

For more information about XSL, see the latest version of the Extensible Stylesheet Language (XSL) at http://www.w3.org/TR/xsl/.

Limitations

The Ghostscript method currently does not support OLE and RichText DataWindow objects. The XSL-FO method currently does not support OLE, RichText, graph, and composite DataWindow objects.

Saving as PDF using the distill method

If you want to save to PDF using the distill method, you do not need to change any properties. The distill method is used by default when you select Save Rows As from the File menu in the DataWindow painter and select PDF as the file type, or when you use the SaveAs method with PDF! as the file type.

PowerBuilder uses a PostScript printer driver specifically designed for distilling purposes to configure the PDF output. You can choose to use a different PostScript printer driver if you want to customize your PostScript settings for generating PDF.

| In the DataWindow painter | To use a custom PostScript printer driver, you must set some properties. |

❖ **To save customized distilled PDF output in the DataWindow painter:**

1   Select the Data Export tab in the Properties view for the DataWindow object.

2   Select PDF from the Format to Configure drop-down list, select Distill! from the Method drop-down list, and select the Distill Custom PostScript check box.

3   Select the Print Specifications tab and specify the name of the printer whose settings you want to use in the Printer Name box.

4   Save the DataWindow object, then select File>Save Rows As, select PDF as the Save As Type, specify a file name, and click Save.

| In a script | The properties you set in the DataWindow painter are saved with the DataWindow object and are used by default when your application runs, but for more control, specify the properties in a script before saving the DataWindow object. To specify a custom printer driver in a script, set the Export.PDF.Distill.CustomPostScript property to Yes and specify a printer with the DataWindow.Printer property: |

```
int li_ret

dw_1.Object.DataWindow.Export.PDF.Method = Distill!
dw_1.Object.DataWindow.Printer = "\\prntsrvr\pr-6"
dw_1.Object.DataWindow.Export.PDF.  &
   Distill.CustomPostScript="Yes"

li_ret = dw_1.SaveAs("custom.PDF", PDF!, true)
```

| System requirements for the distill method | Users must have administrative privileges to create a PDF file. |

To support saving as PDF using GNU Ghostscript, you must download and install Ghostscript files on your system as described in the *Installation Guide*. The default PostScript printer driver and related files are installed in *Sybase\Shared\PowerBuilder\drivers*.

When you deploy applications that use the ability to save as PDF with the distill method, you must make sure your users have installed Ghostscript and have access to the the *drivers* directory.

See the chapter on deployment in *Application Techniques* for more information about redistributing these files.

Saving as PDF using XSL-FO

If you want to save to PDF using XSL-FO, you *must* set one or more properties before saving.

In the DataWindow painter

In the DataWindow painter, you set PDF export properties on the Data Export page in the Properties view.

❖ **To save PDF output using XSL-FO in the DataWindow painter:**

1   Select the Data Export tab in the Properties view for the DataWindow object.

2   Select PDF from the Format to Configure drop-down list and select XSLFOP! from the Method drop-down list.

3   (Optional) If you want simultaneously to send the output directly to a printer using the Java printing option of the Apache FOP processor, select the Print Using XSLFOP check box.
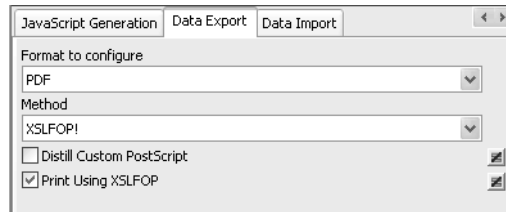
4   Save the DataWindow object, then select File>Save Rows As, select PDF as the Save As Type, specify a file name, and click Save.

PowerBuilder saves the data in the DataWindow object to the file you specified. If you selected the Print Using XSLFOP check box, it also sends the PDF file to the default printer for your system.

In a script

In a script, set the Export.PDF.Method property to XSLFOP! before saving the DataWindow object as PDF using the SaveAs method with the SaveAsType PDF!. To send the PDF file directly to the default printer, set the Export.PDF.XSLFOP.Print property to Yes before saving:

```
int li_ret
dw_1.Modify("Export.PDF.Method = XSLFOP! ")
dw_1.Modify("Export.PDF.xslfop.print='Yes'")
li_ret = dw_1.SaveAs("printed.pdf", PDF!, true)
```

Saving as XSL-FO

You can also save a DataWindow object as XSL-FO, then use the processor of your choice to convert the XSL-FO string to the format you want, applying your own customizations to the conversion. Processors such as the Apache XSL Formatting Objects processor (FOP) can convert XSL-FO documents into several output formats including PDF, PCL, and AWT.

In the DataWindow painter, select File>Save Rows As and select XSL-FO as the file type. In a script, you can use the SaveAs method with the SaveAsType XSLFO!.

For a DataWindow named *dwemp*, the following command lines show the FOP syntax for producing a PDF, a print preview rendered on screen (-awt), and printable output rendered and sent to a printer (-print):

```
Fop dwemp.fo dwemp.pdf
Fop dwemp.fo -awt
Fop dwemp.fo -print
```

For more information about using FOP, see the FOP page of the Apache XML Project Web site at http://xml.apache.org/fop/.

System requirements for XSL-FO

The Apache XSL Formatting Objects processor (FOP) and JDK 1.4 are installed with PowerBuilder to support saving as XSL-FO, saving as PDF using XSL-FO, and Java printing. FOP is installed in *Sybase\Shared\PowerBuilder\fop-0.20.4*. JDK 1.4 is installed in *Sybase\Shared\PowerBuilder\jdk14*.

When you deploy applications that use XSL-FO or Java printing, you must deploy the *fop-0.20.4* directory and the Java Runtime Environment (the JRE is available in a subdirectory of *jdk14*) with your application. These directories must be deployed in the same directory as the PowerBuilder runtime files. For more information, see the chapter on deploying your applications in *Application Techniques*.

On Windows DBCS platforms, you also need to install a file that supports DBCS characters to the Windows font directory, for example, *C:\WINNT\fonts*. To use these fonts, the *userconfig.xml* file in the directory *Shared\PowerBuilder\fop-0.20.4\conf* must be modified to give the full path name of the files you use, for example:

```
<font metrics-file="C:\Program%20Files\Sybase\Shared\
PowerBuilder\fop-0.20.4\conf\cyberbit.xml"
kerning="yes"embed-file="C:\WINNT\Fonts\Cyberbit.ttf">
```

For more information about configuring fonts, see the Apache Web site at http://xml.apache.org/fop/fonts.html.

Saving the data in HTML Table format

HTML Table format is one of the formats in which you can choose to save data. When you save in HTML Table format, PowerBuilder saves a style sheet along with the data. If you use this format, you can open the saved file in a browser such as Internet Explorer or Netscape. Once you have the file in HTML Table format, you can continue to enhance the file in HTML.

About the results
Some presentation styles translate better into HTML than others. The Tabular, Group, Freeform, Crosstab, and Grid presentation styles produce good results. The Composite, RichText, OLE 2.0, and Graph presentation styles and nested reports produce HTML tables based on the result set (data) only and not on the presentation style. DataWindow Objects with overlapping controls in them might not produce the results you want.

❖ **To save a report as an HTML table:**

1   Open a DataWindow object.

2   Open the Preview view if it is not already open.

3   Select File>Save Rows As from the menu bar.

4   Choose the HTML Table format for the file from the Save As Type drop-down list.

5   Name the file.

    PowerBuilder creates a file using the name you supplied and the extension *htm*.

6   Open a browser such as Netscape or Internet Explorer.

7   Use the browser's file open command to open the HTML file.

For more information about working with DataWindow objects and HTML, see the *DataWindow Programmer's Guide*.

Working with PSR files

A PSR file is a special file with the extension PSR created by PowerBuilder, InfoMaker, or DataWindow Designer. PSR stands for Powersoft report.

Windows and PSR files
When PowerBuilder is installed, the PSR file type is registered with Windows.

A PSR file contains a report definition (source and object) as well as the data contained in the report when the PSR file was created.

Figure 19-1: PSR file
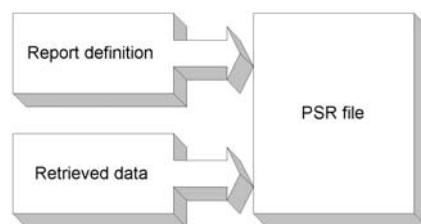
About reports
A report is the same as a nonupdatable DataWindow object. For more information, see "Reports versus DataWindow objects" on page 466.

You can use a PSR file to save a complete report (report design and data). This can be especially important if you need to keep a snapshot of data taken against a database that changes frequently.

PowerBuilder creates a PSR file when you save data in the Powersoft report file format. See "Saving data in an external file" on page 533. PSR files are used primarily by InfoMaker, a reporting tool. When an InfoMaker user opens a PSR file, InfoMaker displays the report in the Report painter. If InfoMaker is not already running, opening a PSR file automatically starts InfoMaker.

Modifying general DataWindow object properties

This section describes the general DataWindow object properties that you can modify.

Changing the DataWindow object style

The general style properties for a DataWindow object include:

- The unit of measure used in the DataWindow object

- A timer interval for events in the DataWindow object

- A background color for the DataWindow object

PowerBuilder assigns defaults when it generates the basic DataWindow object. You can change the defaults.

❖ **To change the default style properties:**

1 Position the pointer in the background of the DataWindow object, display the pop-up menu, and select Properties.

The Properties view displays with the General page on top.

2 Click the unit of measure you want to use to specify distances when working with the DataWindow object:

- PowerBuilder units (PBUs)

- Pixels (smallest element on the display monitor)

- Thousandths of an inch

- Thousandths of a centimeter

Choosing the unit of measure
If you plan to print the contents of the DataWindow object at runtime, change the unit of measure to inches or centimeters to make it easier to specify the margin measurements.

---

3 Specify the number of milliseconds you want between internal timer events in the DataWindow object.

This value determines how often PowerBuilder updates the time fields in the DataWindow object. (Enter 60,000 milliseconds to specify one minute.)

4   Select a background color from the Color drop-down list and click OK.
     The default color is the window background color.

Setting colors in a DataWindow object

You can set different colors for each element of a DataWindow object to
enhance the display of information.

❖   **To set the background color in a DataWindow object:**

1   Position the mouse on an empty spot in the DataWindow object, display
     the pop-up menu, and select Properties.

2   On the General page in the Properties view for the DataWindow object,
     select a color from the Color drop-down list.

❖   **To set the color of a band in a DataWindow object:**

1   Position the mouse pointer on the bar that represents the band, display the
     pop-up menu, then select Properties.

2   On the General page in the band's Properties view, select a color from the
     Color drop-down list.

     The choice you make here overrides the background color for the
     DataWindow object.

❖   **To set colors in controls in a DataWindow object:**

•   Position the mouse pointer on the control, display the pop-up menu, then
     select Properties.

     For controls that use text, you can set colors for background and text on
     the Font page in the Properties view. For drawing controls, you can set
     colors on the General page in the Properties view.

Specifying properties of a grid DataWindow object

In grid DataWindow objects, you can specify:

•   When grid lines are displayed

•   How users can interact with the DataWindow object at runtime

❖ **To specify basic grid DataWindow object properties:**

1 Position the mouse pointer on the background in a grid DataWindow object, display the pop-up menu, and select Properties.

2 Select the options you want in the Grid section on the General page in the Properties view as described in Table 19-3.

Table 19-3: Options for grid DataWindow objects

Option	Result
On	Grid lines always display
Off	Grid lines never display (users cannot resize columns at runtime)
Display Only	Grid lines display only when the DataWindow object displays online
Print Only	Grid lines display only when the contents of the DataWindow object are printed
Column Moving	Columns can be moved at runtime
Mouse Selection	Data can be selected at runtime (and, for example, copied to the clipboard)
Row Resize	Rows can be resized at runtime

Specifying pointers for a DataWindow object

Just as with colors, you can specify different pointers to use when the mouse is over a particular area of the DataWindow object. For example, you might want to change the pointer when the mouse is over a column whose data cannot be changed.

❖ **To change the mouse pointer used at runtime:**

1 Position the mouse over the element of the DataWindow object whose pointer you want to define, display the pop-up menu, and select Properties to display the appropriate Properties view.

You can set a pointer for the entire DataWindow object, specific bands, and specific controls.

2 Select the Pointer tab.

3 Either choose the pointer from the Stock Pointers list or, if you have a file containing pointer definitions (CUR files), enter a pointer file name.

You can use the Browse button to search for the file.

4 Click OK.

Defining print specifications for a DataWindow object

When you are satisfied with the look of the DataWindow object, you can define its print specifications.

❖ **To define print specifications for a DataWindow object:**

1   In the DataWindow painter, select Properties from the DataWindow object's pop-up menu to display the DataWindow object's Properties view.

2   In the Units box on the General page, select a unit of measure.

It is easier to specify the margins when the unit of measure is inches or centimeters.

3   Select the Print Specifications tab.

The Print Specifications page uses the units of measure you specified on the General page.

4   Specify print specifications for the current DataWindow object.

See Table 19-4 for more information.

Table 19-4: Setting print specifications for DataWindow objects

Setting	Description
Document Name	Specify a name to be used in the print queue to identify the report.
Printer Name	Specify the name of a printer to which this report should be sent. If this box is empty, the report is sent to the default system printer. If the specified printer cannot be found, the report is sent to the default system printer if the Can Use Default Printer check box is selected. If the specified printer cannot be found and the Can Use Default Printer check box is not selected, an error is returned.
Margins	Specify top, bottom, left, and right margins. You can also change margins in the Preview view while you are actually looking at data. If you change margins in the Preview view, the changes are reflected here on the Print Specifications page.
Paper Orientation	Choose one of the following: • Default: Uses the default printer setup. • Portrait: Prints the contents of the DataWindow object across the width of the paper. • Landscape: Prints the contents of the DataWindow object across the length of the paper.
Paper Size	Choose a paper size or leave blank to use the default.

Setting	Description
Paper Source	Choose a paper source or leave blank to use the default.
Prompt Before Printing	Select to display the standard Print Setup dialog box each time users make a print request.
Can Use Default Printer	Clear this check box if a printer has been specified in the Printer Name box and you do not want the report to be sent to the default system printer if the specified printer cannot be found. This box is checked by default if a printer name is specified.
Display Buttons - Print Preview	Select to display Button controls in Print Preview. The default is to hide them.
Display Buttons - Print	Select to display Button controls when you print the report. The default is to hide them.
Clip Text	Select to clip static text to the dimensions of a text field when the text field has no visible border setting. The text is always clipped if the text field has visible borders.
Override Print Job	When you print a series of reports using the PrintOpen, PrintDataWindow, and PrintClose methods, all the reports in the print job use the layout, fonts, margins, and other print specifications defined for the computer. Select this check box to override the default print job settings and use the print settings defined for this report.
Collate Copies	Select to collate copies when printing. Collating increases print time because the print operation is repeated to produce collated sets.
Print Preview Shows Outline	Select to display a blue line that represents the margins in Print Preview.
Newspaper Columns Across and Width	If you want a multiple-column report where the data fills one column on a page, then the second, and so on, as in a newspaper, select the number and width of the columns in the Newspaper Columns boxes. See "Printing with newspaper-style columns" next.

Printing with newspaper-style columns

When you define a DataWindow object, you can specify that it print in multiple columns across the page, like a newspaper. A typical use of newspaper-style columns is a phone list, where you want to have more than one column of names on a printed page.

Use Print Preview to see the printed output

Newspaper-style columns are used only when the DataWindow object is printed. They do not appear when a DataWindow object executes (or in Preview). Therefore, to see them in PowerBuilder, use Print Preview in the DataWindow painter.

❖ **To define newspaper-style columns for a DataWindow object:**

1 Build a tabular DataWindow object with the data you want.

2 Select Properties from the DataWindow object's pop-up menu.

3 Select the Print Specifications tab.

4 Specify the number of columns across the page and the width of columns in the Newspaper Columns Across and Newspaper Columns Width properties.

5 For each control in the DataWindow object that you do *not* want to have appear multiple times on the page (such as headers), select Properties from the control's pop-up menu and select the HideSnaked check box on the General page in the Properties view.

Example

This example describes how to create a newspaper-style DataWindow object using the Employee table in the EAS Demo DB.

1 Create a tabular DataWindow object, selecting the last name, first name, and phone number columns, and add a title, page number, and date.

The Emp_Fname column and the text control holding a comma are defined as Slide Left, so they display just to the right of the Emp_Lname column.
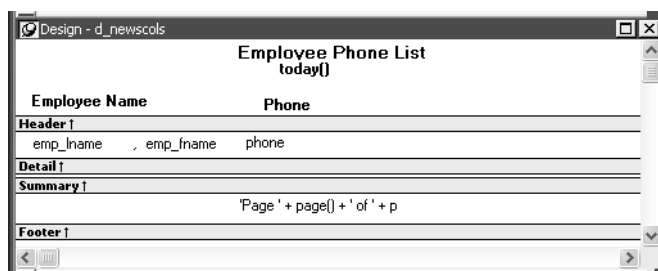
2 On the Print Specifications page of the DataWindow object's Properties view, specify two columns across and a column width of 3.5 inches in the Newspaper Columns boxes. (Make sure that Units is set to inches on the General property page.)

3   To view the DataWindow object as it will be printed, place the pointer in the Preview view and select File>PrintPreview.

The DataWindow object displays the result set in two columns. Everything above the column headers (which includes page number, title, and date) also shows twice because of the 2-column specification. This information should appear only once per page.

4   To specify that page number, title, and date appear only once on the page, you need to suppress printing after the first column. For each of these controls, select Properties from the control's pop-up menu. Then select the HideSnaked check box.

The finished DataWindow object has one set of page heading information and two columns of column header and detail information.

(** means out of state)	Employee Phone List		
	08/20/04		
Employee Name	**Phone**	**Employee Name**	**Phone**
Ahmed , Alex	(617) 555-8748	Pickett , Catherine	(617) 555-3478
Barker , Joseph	(617) 555-8021	Poitras , Kathleen	(617) 555-3920
Barletta , Irene	(617) 555-8345	Powell , Thomas	(617) 555-1956
Bertrand , Jeannette	(508) 555-8138	Preston , Mark	(617) 555-5862
Bigelow , Janet	(617) 555-1493	Rabkin , Andrew	(617) 555-4444
Blaikie , Barbara	(617) 555-9345	Rebeiro , Anthony	(617) 555-5737
Braun , Jane	(617) 555-7857	Romero , Sheila	(617) 555-8138
Breault , Robert	(617) 555-3099	Samuels , Peter	(617) 555-8342
Bucceri , Matthew	(617) 555-5336	** **Savarino, Pamela**	**(310) 555-1857**
Butterfield , Joyce	(617) 555-2232	Scott , David	(617) 555-3246
Chao , Shih Lin	(617) 555-5921	Shea , Mary Anne	(617) 555-4616
Charlton , Doug	(508) 555-9246	** **Sheffield, John**	**(713) 555-3877**
** **Chin, Philip**	**(404) 555-2341**	Shishov , Natasha	(617) 555-2755

Modifying text in a DataWindow object

When PowerBuilder initially generates the basic DataWindow object, it uses the following attributes and fonts:

• For the text and alignment of column headings and labels, PowerBuilder uses the extended column attributes made in the Database painter.

• For fonts, PowerBuilder uses the definitions made in the Database painter for the table. If you did not specify fonts for the table, PowerBuilder uses the defaults set in the Application painter.

You can override any of these defaults in a particular DataWindow object.

❖ **To change text in a DataWindow object:**

1   Select the text.

The first box in the StyleBar is now active.

2   Type the new text.

Use ~n~r to embed a newline character in the text.

❖ **To change the text properties for a text control in a DataWindow object:**

1   Select the text control.

2   Do one of the following:

•   Change the text properties in the StyleBar.

•   Select the Font page in the control's Properties view and change the properties there.

Defining the tab order in a DataWindow object

When PowerBuilder generates the basic DataWindow object, it assigns columns a default tab order, the default sequence in which focus moves from column to column when a user presses the Tab key at runtime. PowerBuilder assigns tab values in increments of 10 in left-to-right and top-to-bottom order.

Tab order is not used in the Design view
Tab order is used when a DataWindow object is executed, but it is not used in the DataWindow painter Design view. In the Design view, the Tab key moves to the controls in the DataWindow object in the order in which the controls were placed in the Design view.

If the DataWindow object contains columns from more than one table

If you are defining a DataWindow object with more than one table, PowerBuilder assigns each column a tab value of 0, meaning the user cannot tab to the column. This is because, by default, multitable DataWindow objects are not updatable—users cannot modify data in them. You can change the tab values to nonzero values to allow tabbing in these DataWindow objects.

For more about controlling updates in a DataWindow object, see Chapter 21, "Controlling Updates in DataWindow Objects."

Tab order changes have no effect in grid DataWindow objects
In a grid DataWindow object, the tab sequence is always left to right (except on right-to-left operating systems). Changing the tab value to any number other than 0 has no effect.

❖ **To change the tab order:**

1 Select Format>Tab Order from the menu bar or click the Tab Order button on PainterBar2.

The current tab order displays.

2 Use the mouse or the Tab key to move the pointer to the tab value you want to change.

3 Enter a new tab value in the range 0 to 9999.

0 removes the column from the tab order (the user cannot tab to the column). It does not matter exactly what value you use (other than 0); all that matters is relative value. For example, if you want the user to tab to column B after column A but before column C, set the tab value for column B so it is between the value for column A and the value for column C.

4 Repeat the procedure until you have the tab order you want.

5 Select Format>Tab Order from the menu bar or click the Tab Order button again.

PowerBuilder saves the tab order.

Each time you select Tab Order, PowerBuilder reassigns tab values to include any columns that have been added to the DataWindow object and to allow space to insert new columns in the tab order.

Changing tab order at runtime
To change tab order programmatically at runtime, use the SetTabOrder method.

Naming controls in a DataWindow object

You use names to identify columns and other controls in validation rules, filters, PowerScript functions, and DataWindow expression functions.

The DataWindow painter automatically generates names for all controls in a DataWindow object. To name columns, labels, and headings, the DataWindow painter uses database and extended attribute information. To name all other controls, it uses a system of prefixes. You can control the prefixes used for automatic name generation and you can specify the name of any control explicitly.

❖ **To specify prefixes for naming controls systematically in a DataWindow object:**

1 Select Design>Options from the menu bar and then select the Prefixes tab.

2 Change prefixes as desired and click OK.

❖ **To specify a name of a control in a DataWindow object:**

1 Select Properties from the control's pop-up menu and then select the General tab in the Properties view.

2 Type the name in the Name box.

Using borders in a DataWindow object

You can place borders around text, columns, graphs, and crosstabs to enhance their appearance. PowerBuilder provides six types of borders: Underline, Box, ResizeBorder, ShadowBox, Raised, and Lowered:

| Amo & Sons | Laura | McCarthy | 1210 Highway 36 | Carmel | IN |

Border appearance varies
Changing the border style may not have the same effect on all Windows platforms.

❖ **To add a border to a control in a DataWindow object:**

1   Select one or more controls.

2   Select the border you want from the Border drop-down toolbar in the PainterBar.

PowerBuilder places the border around the selected controls.

You can also specify a border for one or more controls in the Properties view on the General page.

Specifying variable-height bands in a DataWindow object

Sometimes DataWindow objects contain columns whose data is of variable length. For example, a Memo column in a table might be a character column that can take up to several thousand characters. Reserving space for that much information for the column in the detail band would make the detail band's height very large, meaning users could see few rows at a time.

The detail band can resize based on the data in the Memo column. If the Memo column has only one line of text, the detail band should be one line. If the Memo column has 20 lines of text, the detail band should be 20 lines high.

To provide a band that resizes as needed, specify that the variable-length columns and the band have Autosize Height. All bands in the DataWindow can be resized, but nested report overflow is supported only in the Detail band. If autosizing would preclude the display of at least one Detail band row per page, other bands cannot be autosized. Autosizing is not supported with the Graph, RichText, OLE, or Label presentation styles.

❖ **To create a resizable band in a DataWindow object:**

1   Select Properties from the pop-up menu of a column that should resize based on the amount of data.

2   Select the Autosize Height check box on the Position page.

3   Clear the Auto Horz Scroll check box on the Edit page.

PowerBuilder wraps text in the Preview view instead of displaying text on one scrollable line.

4   Repeat steps 1 to 3 for any other columns that should resize.

5   Select Properties from the band's pop-up menu.

6   Select the Autosize Height check box on the General page.

In the Preview view, the band resizes based on the contents of the columns you defined as having Autosize Height.

Using the RowHeight function with Autosize Height

When a detail band has Autosize Height set to true, you should avoid using the RowHeight DataWindow expression function to set the height of any element in the row. Doing so can result in a logical inconsistency between the height of the row and the height of the element. If you need to use RowHeight, you must set the Y coordinate of the element to 0 on the Position page in the Properties view, otherwise the bottom of the element might be clipped. You must do this for every element that uses such an expression. If you move any elements in the band, make sure that their Y coordinates are still set to 0.

You should not use an expression whose runtime value is greater than the value returned by RowHeight. For example, you should not set the height of a column to `rowheight() + 30`. Such an expression produces unpredictable results at runtime.

___

Clipping columns
You can have Autosize Height columns without an Autosize Height detail band. If such a column expands beyond the size of the detail band in the Preview view, it is clipped.

___

Modifying the data source of a DataWindow object

When modifying a DataWindow object, you might realize that you have not included all the columns you need, or you might need to define retrieval arguments. You can modify the data source from the DataWindow painter. How you do it depends on the data source.

Modifying SQL SELECT statements

If the data source is SQL (such as Quick Select, SQL Select, or Query), you can graphically modify the SQL SELECT statement.

❖   **To modify a SQL data source:**

1   Select Design>Data Source from the menu bar.

PowerBuilder returns you to the Select painter. (If you used Quick Select to define the data source, this might be the first time you have seen the Select painter.)

2   Modify the SELECT statement graphically using the same techniques as
     when creating it.

     For more information, see "Using SQL Select" on page 486.

Modifying the statement syntactically
Select Design>Convert to Syntax from the menu bar to modify the
SELECT statement syntactically.

---

3   Click the Return button to return to the painter.

Some changes you make (such as adding or removing columns) require
PowerBuilder to modify the update capabilities of the DataWindow object.

For more information about controlling updates in a DataWindow object, see
Chapter 21, "Controlling Updates in DataWindow Objects."

Changing the table
If you change the table referenced in the SELECT statement, PowerBuilder
maintains the columns in the Design view (now from a different table) only if
they match the data types and order of the columns in the original table.

Modifying the retrieval
arguments

You can add, modify, or delete retrieval arguments when modifying your data
source.

❖   **To modify the retrieval arguments:**

1   In the Select painter, select Design>Retrieval Arguments from the menu
     bar.

     The Specify Retrieval Arguments dialog box displays, listing the existing
     arguments.

2   Add, modify, or delete the arguments.

3   Click OK.

     You return to the Select painter, or to the text window displaying the
     SELECT statement if you are modifying the SQL syntactically.

4   Reference any new arguments in the WHERE or HAVING clause of the
     SELECT statement.

For more information about retrieval arguments, see Chapter 18, "Defining
DataWindow Objects."

Modifying the result set

If the data source is External or Stored Procedure, you can modify the result set description.

❖ **To modify a result set:**

1   If the Column Specification view is not open, select View>Column Specifications from the menu bar.

2   Review the specifications and make any necessary changes.

If the data source is a stored procedure

If you are modifying the result set for a DataWindow object whose data source is a stored procedure, the pop-up menu for the Column Specification view contains the menu item Stored Procedure.

Select Stored Procedure from the Column Specification view's pop-up menu to edit the Execute statement, select another stored procedure, or add retrieval arguments. For more information about editing the Execute statement, see "Using Stored Procedure" on page 504.

Storing data in a DataWindow object using the Data view

Usually you retrieve data into a DataWindow object from the database, because the data is changeable and you want the latest information. However, sometimes the data you display in a DataWindow object never changes (as in a list of states or provinces), and sometimes you need a snapshot of the data at a certain point in time. In these situations, you can store the data in the DataWindow object itself. You do not need to go out to the database or other data source to display the data.

The most common reason to store data in a DataWindow object is for use as a drop-down DataWindow where the data is not coming from a database. For example, you might want to display a list of postal codes for entering values in a State or Province column in a DataWindow object. You can store those codes in a DataWindow object and use the DropDownDataWindow edit style for the column.

For more information about using the DropDownDataWindow edit style, see Chapter 22, "Displaying and Validating Data."

❖ **To store data in a DataWindow object:**

1   If the Data view is not already displayed, select View>Data from the menu bar.

    In the default layout for the DataWindow painter, the Data view displays in a stacked pane under the Properties view. All columns defined for the DataWindow object are listed at the top.

2   Do any of the following:

    •   Click the Insert Row button in the PainterBar to create an empty row and type a row of data. You can enter as many rows as you want.

    •   Click the Retrieve button in the PainterBar to retrieve all the rows of data from the database. You can delete rows you do not want to save or manually add new rows.

    •   Click the Delete button in the PainterBar to delete unwanted rows.

    Data changes are local to the DataWindow object
    Adding or deleting data here does not change the data in the database. It only determines what data will be stored with the DataWindow object when you save it. The Update DB button is disabled.

3   When you have finished, save the DataWindow object.

When you save the DataWindow object, the data is stored in the DataWindow object.

Sharing data with the Preview view

To see changes you make in the Data view reflected in the Preview view, select ShareData from the pop-up menu in the Data view. The Preview view shows data from the storage buffer associated with the Data view.

Saving the DataWindow object without data

If you saved the DataWindow object with data to obtain a snapshot, you usually need to save it again without data. To do so, select Delete All Rows from the pop-up menu in the Data view before saving.

Sharing DataWindow objects with other developers
Storing data in a DataWindow object is a good way to share data *and its definition* with other developers. They can simply open the DataWindow object on their computers to get the data and all its properties.

What happens at runtime

Data stored in a DataWindow object is stored within the actual object itself, so when a window opens showing such a DataWindow, the data is already there. There is no need to issue Retrieve to get the data.

PowerBuilder *never* retrieves data into a drop-down DataWindow that already contains data. For all other DataWindow objects, if you retrieve data into a DataWindow object stored with data, PowerBuilder handles it the same as a DataWindow object that is not stored with data: PowerBuilder gets the latest data by retrieving rows from the database.

Retrieving data

In a DataWindow object, you can prompt for retrieval criteria, retrieve rows as needed, and save retrieved rows to disk.

Prompting for retrieval criteria in a DataWindow object

You can define your DataWindow object so that it always prompts for retrieval criteria just before it retrieves data. PowerBuilder allows you to prompt for criteria when retrieving data for a DataWindow control, but not for a DataStore object.

❖ **To prompt for retrieval criteria in a DataWindow object:**

1   If the Column Specifications view is not already displayed, select View>Column Specifications from the menu bar.

In the default layout for the DataWindow painter, the Column Specifications view displays in a stacked pane under the Properties view. All columns defined for the DataWindow object are listed in the view.

2   Select the Prompt check box next to each column for which you want to specify retrieval criteria at runtime.

When you specify prompting for criteria, PowerBuilder displays the Specify Retrieval dialog box just before a retrieval is to be done. (It is the last thing that happens before the SQLPreview event.)

Each column you selected in the Column Specification view displays in the grid. Users can specify criteria here exactly as in the grid in the Quick Select dialog box. Criteria specified here are added to the WHERE clause for the SQL SELECT statement defined for the DataWindow object.

Testing in PowerBuilder
You can test the prompting for criteria by retrieving data in the Preview view of the DataWindow object.

Using edit styles

If a column uses a code table or the RadioButton, CheckBox, or DropDownListBox edit style, an arrow displays in the column header and users can select a value from a drop-down list when specifying criteria:

If you do not want the drop-down list used for a column for specifying retrieval criteria, select the Override Edit check box on the General page of the column's Properties view.

Forcing the entry of criteria

If you have specified prompting for criteria for a column, you can force the entry of criteria for the column by selecting the Equality Required check box on the General page of the column's Properties view. PowerBuilder underlines the column header in the grid during prompting. Selection criteria for the specified column must be entered, and the = operator must be used.

For more information

The section "Using Quick Select" on page 477 describes in detail how you and your users can specify selection criteria in the grid.

The chapter on dynamic DataWindow objects in the *DataWindow Programmer's Guide* describes how to write code to allow users to specify retrieval criteria at runtime.

Retrieving rows as needed

If a DataWindow object retrieves hundreds of rows, there can be a noticeable delay at runtime while all the rows are retrieved and before control returns to the user. For these DataWindow objects, PowerBuilder can retrieve only as many rows as it has to before displaying data and returning control to the user.

For example, if a DataWindow object displays only 10 rows at a time, PowerBuilder only needs to retrieve 10 or more rows before presenting the data. Then, as the user pages through the data, PowerBuilder continues to retrieve what is necessary to display the new information. There may be slight pauses while PowerBuilder retrieves the additional rows, but these pauses are usually preferable to waiting a long time to start working with data.

❖ **To specify that a DataWindow object retrieve only as many rows as it needs to:**

•   Select Rows>Retrieve Options>Rows As Needed from the menu bar.

With this setting, PowerBuilder presents data and returns control to the user when it has retrieved enough rows to display in the DataWindow object.

Retrieve Rows As Needed is overridden if you have specified sorting or have used aggregate functions, such as Avg and Sum, in the DataWindow object. This is because PowerBuilder must retrieve every row before it can sort or perform aggregates.

In a multiuser situation, Retrieve Rows As Needed might lock other people out of the tables.

Saving retrieved rows to disk

If you want to maximize the amount of memory available to PowerBuilder and other running applications, PowerBuilder can save retrieved data on your hard disk in a temporary file rather than keep the data in memory. PowerBuilder swaps rows of data from the temporary file into memory as needed to display data.

❖ **To maximize available memory by saving retrieved rows to disk:**

•   Select Rows>Retrieve Options>Rows to Disk from the menu bar.

With this setting, when displaying data, PowerBuilder swaps rows of data from the temporary file into memory instead of keeping all the retrieved rows of data in memory.

Working with Controls in DataWindow Objects

About this chapter

One of the ways you can enhance a DataWindow object is to add controls, such as columns, drawing objects, buttons, and computed fields. You can also change the layout of the DataWindow object by reorganizing, positioning, and rotating controls. This chapter shows you how.

Contents

Adding controls to a DataWindow object

This section describes adding controls to enhance your DataWindow object.

Adding columns to a DataWindow object

You can add columns that are included in the data source to a DataWindow object. When you first create a DataWindow object, each of the columns in the data source is automatically placed in the DataWindow object. Typically, you would add a column to restore one that you had deleted from the DataWindow object, or to display the column more than once in the DataWindow object.

Adding columns not previously retrieved to the data source
To specify that you want to retrieve a column not previously retrieved (that is, add a column to the data source), you must modify the data source.

See "Modifying the data source of a DataWindow object" on page 551.

❖ **To add a column from the data source to a DataWindow object:**

1 Select Insert>Control>Column from the menu bar.

2 Click where you want to place the column.

   The Select Column dialog box displays, listing all columns included in the data source of the DataWindow object.

3 Select the column and click OK.

Adding text to a DataWindow object

When PowerBuilder generates a basic DataWindow object from a presentation style and data source, it places columns and their headings in the workspace. You can add text anywhere you want to make the DataWindow object easier to understand.

❖ **To add text to a DataWindow object:**

1 Select Insert>Control>Text from the menu bar.

2 Click where you want the text.

   PowerBuilder places the text control in the Design view and displays the word `text`.

3 Type the text you want.

4 (Optional) Change the font, size, style, and alignment for the text using the StyleBar.

Displaying an ampersand character
If you want to display an ampersand character, type a double ampersand in the Text field. A single ampersand causes the next character to display with an underscore because it is used to indicate accelerator keys.

About the default font and style

When you place text in a DataWindow object, PowerBuilder uses the font and style (such as boldface) defined for the application's text in the Application painter. You can override the text properties for any text in a DataWindow object.

For more about changing the application's default text font and style, see Chapter 5, "Working with PowerScript Targets."

Adding drawing controls to a DataWindow object

You can add the following drawing controls to a DataWindow object to enhance its appearance:

Rectangle
RoundRectangle
Line
Oval

❖ **To place a drawing control in a DataWindow object:**

1   Select the drawing control from the Insert>Control menu.

2   Click where you want the control to display.

3   Resize or move the drawing control as needed.

4   Use the drawing control's Properties view to change its properties as needed.

For example, you might want to specify a fill color for a rectangle or thickness for a line.

Adding a group box to a DataWindow object

To visually enhance the layout of a DataWindow object, you can add a group box. A group box is a static frame used to group and label a set of controls in a DataWindow object. The following example shows two group boxes in a report (nonupdatable DataWindow object). The Address group box groups address information and the Phone/Fax group box groups telephone numbers.

❖ **To add a group box to a DataWindow object:**

1   Select Insert>Control>Group Box from the menu bar and click in the Design view.

2   With the group box selected, type the text to display in the frame.

3   Move and resize the group box as appropriate.

Adding pictures to a DataWindow object

You can place pictures, such as your company logo, in a DataWindow object to enhance its appearance. If you place a picture in the header, summary, or footer band of the DataWindow object, the picture displays each time the content of that band displays. If you place the picture in the detail band of the DataWindow object, it displays in each row.

❖ **To place a picture in a DataWindow object:**

1   Select Insert>Control>Picture from the menu bar.

2   Click where you want the picture to display.

The Select Picture dialog box displays.

3   Use the Browse button to find the file or enter a file name in the File Name box. Then click Open.

The picture must be a bitmap (BMP), runlength-encoded (RLE), Windows metafile (WMF), Graphics Interchange Format (GIF), or Joint Photographic Experts Group (JPEG) file.

4   Display the pop-up menu and select Original Size to display the bitmap in its original size.

You can use the mouse to change the size of the bitmap in the DataWindow painter.

5   Click the Invert Image check box on the General page of the Properties view to display the picture with its colors inverted.

Tips for using pictures   To display a different picture for each row of data, retrieve a column containing picture file names from the database.

For more information, see "Specifying additional properties for character columns" on page 403.

To compute a picture name at runtime, use the Bitmap function in the expression defining a computed field. If you change the bitmap in the Picture control in a DataWindow object, you need to reset the original size property. The property automatically reverts to the default setting when you change the bitmap.

To use a picture to indicate that a row has focus at runtime, use the SetRowFocusIndicator function.

Adding computed fields to a DataWindow object

You can use computed fields in any band of the DataWindow object. Typical uses with examples include:

•   Calculations based on column data that change for each retrieved row

If you retrieve yearly salary, you can define a computed field in the detail band that displays monthly salary: `Salary / 12`.

•   Summary statistics of the data

In a grouped DataWindow object, you can use a computed field to calculate the totals of a column, such as salary, for each group: `sum (salary for group 1)`.

- Concatenated fields

  If you retrieve first name and last name, you can define a computed field that concatenates the values so they appear with only one space between them: `Fname + " " + Lname`.

- System information

  You can place the current date and time in a DataWindow object's header using the built-in functions `Today()` and `Now()` in computed fields.

Computed columns versus computed fields

When creating a DataWindow object, you can define computed columns and computed fields as follows:

- In the Select painter, you can define computed columns when you are defining the SELECT statement that will be used to retrieve data into the DataWindow object.

- In the DataWindow painter, you can define computed fields after you have defined the SELECT statement (or other data source).

The difference between the two ways

When you define the computed column in the Select painter, the value is calculated by the DBMS when the data is retrieved. The computed column's value does not change until data has been updated and retrieved again.

When you define the computed field in the DataWindow painter, the value of the column is calculated in the DataWindow object after the data has been retrieved. The value changes dynamically as the data in the DataWindow object changes.

Example

Consider a DataWindow object with four columns: Part number, Quantity, Price, and Cost. Cost is computed as Quantity * Price.

Part #	Quantity	Price	Cost
101	100	1.25	125.00

If Cost is defined as a computed column in the Select painter, the SELECT statement is as follows:

```
SELECT part.part_num,
part.part_qty,
part.part_price,
part.part_qty * part.part_price
FROM part;
```

If the user changes the price of a part in the DataWindow object in this scenario, the cost does not change in the DataWindow object until the database is updated and the data is retrieved again. The user sees a display with the changed price but the unchanged, incorrect cost.

Part #	Quantity	Price	Cost
101	100	2.50	*125.00*

If Cost is defined as a computed field in the DataWindow object, the SELECT statement is as follows, with no computed column:

```
SELECT part.part_num,
part.part_qty,
part.part_price
FROM part;
```

The computed field is defined in the DataWindow object as `Quantity * Price`.

In this scenario, if the user changes the price of a part in the DataWindow object, the cost changes immediately.

Part #	Quantity	Price	Cost
101	100	2.50	*250.00*

Recommendation   If you want your DBMS to do the calculations on the server before bringing data down and you do not need the computed values to be updated dynamically, define the computed column as part of the SELECT statement.

If you need computed values to change dynamically, define computed fields in the DataWindow painter Design view, as described next.

Defining a computed field in the DataWindow painter Design view

❖ **To define a computed field in the DataWindow painter Design view:**

1   Select Insert>Control>Computed Field from the menu bar.

2   Click where you want to place the computed field.

     If the calculation is to be based on column data that changes for each row, make sure you place the computed field in the detail band.

     The Modify Expression dialog box displays, listing:

     •   DataWindow expression functions you can use in the computed field

- The columns in the DataWindow object

- Operators and parentheses

3   Enter the expression that defines the computed field as described in "Entering the expression" next.

4   (Optional) Click Verify to test the expression.

PowerBuilder analyzes the expression.

5   Click OK.

Entering the expression

You can enter any valid DataWindow expression when defining a computed field. You can paste operators, columns, and DataWindow expression functions into the expression from information in the Modify Expression dialog box. Use the + operator to concatenate strings.

You can use any built-in or user-defined global function in an expression. You cannot use object-level functions.

DataWindow expressions
You are entering a DataWindow expression, not a SQL expression processed by the DBMS, so the expression follows the rules for DataWindow expressions. For complete information about DataWindow expressions, see the *DataWindow Reference*.

Referring to next and previous rows

You can refer to other rows in a computed field. This is particularly useful in n-up DataWindow objects when you want to refer to another row in the detail band. Use this syntax:

   ColumnName[*x*]

where *x* is an integer. 0 refers to the current row (or first row in the detail band), 1 refers to the next row, –1 refers to the previous row, and so on.

Examples

Table 20-1 shows some examples of computed fields.

Table 20-1: Computed field examples

To display	Enter this expression	In this band
Current date at top of each page	`Today()`	Header
Current time at top of each page	`Now()`	Header
Current page at bottom of each page	`Page()`	Footer
Total page count at bottom of each page	`PageCount()`	Footer
Concatenation of Fname and Lname columns for each row	`Fname + " " + Lname`	Detail

To display	Enter this expression	In this band
Monthly salary if Salary column contains annual salary	`Salary / 12`	Detail
Four asterisks if the value of the Salary column is greater than $50,000	`IF(Salary> 50000, "****", "")`	Detail
Average salary of all retrieved rows	`Avg(Salary)`	Summary
Count of retrieved rows, assuming each row contains a value for EmpID	`Count(EmpID)`	Summary

For complete information about the functions you can use in computed fields in the DataWindow painter, see the *DataWindow Reference*.

Menu options and PainterBar buttons for common functions

PowerBuilder provides a quick way to create computed fields that summarize values in the detail band, display the current date, or show the current page number.

❖ **To summarize values:**

1   Select one or more columns in the DataWindow object's detail band.

2   Select Insert>Control from the menu bar.

3   Select one of the options at the bottom of the cascading menu: Average, Count, or Sum.

    The same options are available at the bottom of the Controls drop-down toolbar on the PainterBar.

    PowerBuilder places a computed field in the summary band or in the group trailer band if the DataWindow object is grouped. The band is resized automatically to hold the computed field. If there is already a computed field that matches the one being generated, it is skipped.

❖ **To insert a computed field for the current date or page number:**

1   Select Insert>Control from the menu bar.

2   Select Today() or Page n of n from the options at the bottom of the cascading menu.

    The same options are available at the bottom of the Controls drop-down toolbar on the PainterBar.

3   Click anywhere in the DataWindow object.

    If you selected Today, PowerBuilder inserts a computed field containing this expression: `Today()`. For Page n of n, the computed field contains this expression: `'Page ' + page() + ' of ' + pageCount()`.

Adding custom buttons that place computed fields	You can add buttons to the PainterBar in the DataWindow painter that place computed fields using any of the aggregate functions, such as Max, Min, and Median.

❖ **To customize the PainterBar with custom buttons for placing computed fields:**

1   Place the mouse pointer over the PainterBar and select Customize from the pop-up menu.

   The Customize dialog box displays.

2   Click Custom in the Select palette group to display the set of custom buttons.

3   Drag a custom button into the Current toolbar group and release it.

   The Toolbar Item Command dialog box displays.

4   Click the Function button.

   The Function For Toolbar dialog box displays.

5   Select a function and click OK.

   You return to the Toolbar Item Command dialog box.

6   Specify text and microhelp that displays for the button, and click OK.

   PowerBuilder places the new button in the PainterBar. You can click it to add a computed field to your DataWindow object the same way you use the built-in Sum button.

Adding buttons to a DataWindow object

Buttons make it easy to provide command button actions in a DataWindow object. No coding is required. The use of Button controls in the DataWindow object, instead of CommandButton controls in a window, ensures that actions appropriate to the DataWindow object are included in the object itself.

The Button control is a command or picture button that can be placed in a DataWindow object. When clicked at runtime, the button activates either a built-in or user-supplied action.

For example, you can place a button in a report and specify that clicking it opens the Filter dialog box, where users can specify a filter to be applied to the currently retrieved data.

❖ **To add a button to a DataWindow object:**

1 Select Insert>Control>Button from the menu bar.

2 Click where you want the button to display.

   You may find it useful to put a Delete button or an Insert button in the detail band. Clicking a Delete button in the detail band will delete the row next to the button clicked. Clicking an Insert button in the detail band will insert a row following the current row.

   Be careful when putting buttons in the detail band
   Buttons in the detail band repeat for every row of data, which is not always desirable. Buttons in the detail band are not visible during retrieval, so a Cancel button in the detail band would be unavailable when needed.

   ---

3 With the button still selected, type the text to display on the button.

4 Display the General page of the Properties view for the button.

5 Select the action you want to assign to the button from the Action drop-down list.

   For information about actions, see "Actions assignable to buttons in DataWindow objects" on page 571.

6 If you want to add a picture to the button, select the Action Default Picture check box or enter the name of the Picture file to display on the button.

   If you plan to use the DataWindow object as a Web DataWindow, you can use images for default buttons provided in the *dwaction.jar* file. For more information, see the *DataWindow Programmer's Guide*.

7 If you want to suppress event processing when the button is clicked at runtime, select the Suppress Event check box.

   When this option has been selected for the button and the button is clicked at runtime, only the action assigned to the button and the Clicked event are executed. The ButtonClicking and the ButtonClicked events are not triggered.

8 Click OK.

What happens if Suppress Event is off

If Suppress Event is off and the button is clicked, the Clicked and ButtonClicking events are fired. Code in the ButtonClicking event (if any) is executed. Note that the Clicked event is executed before the ButtonClicking event.

- If the return code from the ButtonClicking event is 0, the action assigned to the button is executed and then the ButtonClicked event is executed.

- If the return code from the ButtonClicking event is 1, neither the action assigned to the button nor the ButtonClicked event are executed.

Do not use a message box in the Clicked event
If you call the MessageBox function in the Clicked event, the action assigned to the button is executed, but the ButtonClicking and ButtonClicked events are not executed.

Example

For an example of a DataWindow object that uses buttons, see the d_button_report object in the Code Examples application.

This DataWindow object has several buttons that have default actions, and two that have user-defined actions. In the Properties view in the DataWindow painter, these buttons are named cb_help and cb_exit. Suppress Event is off for all buttons.

In the Window painter, the Clicked and ButtonClicking events for the DataWindow control that contains d_button_report are not scripted. This is the ButtonClicked event script:

```
stringls_Object
stringls_win

ls_Object = String(dwo.name)

If ls_Object = "cb_exit" Then
    Close(Parent)
ElseIf ls_Object = "cb_help" Then
    ls_win = parent.ClassName()
    f_open_help(ls_win)
End If
```

This script is triggered when any button in the DataWindow object is clicked.

Controlling the display of buttons in print preview and on printed output

You can choose whether to display buttons in print preview or in printed output. You control this in the Properties view for the DataWindow object (not the Properties view for the button).

❖ **To control the display of buttons in a DataWindow object in print preview and on printed output:**

1 Display the DataWindow object's Properties view with the Print Specification page on top.

2 Select the Display Buttons – Print check box.

The buttons are included in the printed output when the DataWindow object is printed.

3 Select the Display Buttons – Print Preview check box.

The buttons display on the screen when viewing the DataWindow object in print preview.

Actions assignable to buttons in DataWindow objects

Table 20-2 shows the actions you can assign to a button in a DataWindow object. Each action is associated with a numeric value (the Action DataWindow object property) and a return code (the actionreturncode event argument).

The following code in the ButtonClicked event displays the value returned by the action:

```
MessageBox("Action return code", actionreturncode)
```

Table 20-2: Button actions for DataWindow objects

Action	Effect	Value	Action return code
User Defined (default)	Allows the developer to program the ButtonClicked event with no intervening action occurring.	0	The return code from the user's coded event script.
Retrieve (Yield)	Retrieves rows from the database. Before retrieval occurs, the option to yield is turned on; this will allow the Cancel action to take effect during a long retrieve.	1	Number of rows retrieved. -1 if retrieve fails.
Retrieve	Retrieves rows from the database. The option to yield is not automatically turned on.	2	Number of rows retrieved. -1 if retrieve fails.

Action	Effect	Value	Action return code
Cancel	Cancels a retrieval that has been started with the option to yield.	3	0
Page Next	Scrolls to the next page.	4	The row displayed at the top of the DataWindow control when the scrolling is complete or attempts to go past the first row. -1 if an error occurs.
Page Prior	Scrolls to the prior page.	5	The row displayed at the top of the DataWindow control when the scrolling is complete or attempts to go past the first row. -1 if an error occurs.
Page First	Scrolls to the first page.	6	1 if successful. -1 if an error occurs.
Page Last	Scrolls to the last page.	7	The row displayed at the top of the DataWindow control when the scrolling is complete or attempts to go past the first row. -1 if an error occurs.
Sort	Displays Sort dialog box and sorts as specified.	8	1 if successful. -1 if an error occurs.
Filter	Displays Filter dialog box and filters as specified.	9	Number of rows filtered. Number < 0 if an error occurs.
Delete Row	If button is in detail band, deletes row associated with button; otherwise, deletes the current row.	10	1 if successful. -1 if an error occurs.
Append Row	Inserts row at the end.	11	Row number of newly inserted row.
Insert Row	If button is in detail band, inserts row using row number associated with the button; otherwise, inserts row using the current row.	12	Row number of newly inserted row.
Update	Saves changes to the database. If the update is successful, a Commit will be issued; if the update fails, a Rollback will be issued.	13	1 if successful. -1 if an error occurs.
Save Rows As	Displays Save As dialog box and saves rows in the format specified.	14	Number of rows filtered. Number < 0 if an error occurs.
Print	Prints one copy of the DataWindow object.	15	0
Preview	Toggles between preview and print preview.	16	0

Action	Effect	Value	Action return code
Preview With Rulers	Toggles between rulers on and off.	17	0
Query Mode	Toggles between query mode on and off.	18	0
Query Sort	Allows user to specify sorting criteria (forces query mode on).	19	0
Query Clear	Removes the WHERE clause from a query (if one was defined).	20	0

Adding graphs to a DataWindow object

Graphs are one of the best ways to present information. For example, if your application displays sales information over the course of a year, you can easily build a graph in a DataWindow object to display the information visually.

PowerBuilder offers many types of graphs and provides you with the ability to control the appearance of a graph to best meet your application's needs.

For information on using graphs, see Chapter 26, "Working with Graphs."

Adding InkPicture controls to a DataWindow object

The InkPicture control is designed for use on a Tablet PC and provides the ability to capture ink input from users of Tablet PCs. The control captures signatures, drawings, and other annotations that do not need to be recognized as text.

The InkPicture control is fully functional on Tablet PCs. If the Microsoft Tablet PC Software Development Kit (SDK) 1.7 is installed on other computers, InkPicture controls in DataWindow objects can accept ink input from the mouse.

For more information about ink controls and the Tablet PC, and to download the Tablet PC SDK, go to the Microsoft Tablet PC Web site at http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnanchor/html/tabletpc.asp

You use an InkPicture control with a table that has a blob column to store the ink data, and optionally a second blob column to provide a background image.

The InkPicture control behaves like a Picture control that accepts annotation. You can associate a picture with the control so that the user can draw annotations on the picture, then save the ink, the picture, or both. If you want to use the control to capture and save signatures, you usually do not associate a picture with it.

To add an InkPicture control to a DataWindow object, select Insert>Control>InkPicture from the menu. A dialog box displays to let you specify a blob column to store the ink data and another to use as a background image. After you specify the columns in the dialog box, the InkPicture control displays in the DataWindow and its Properties view includes a Definition tab page where you can view or change the column definitions.

Adding OLE controls to a DataWindow object

You can add the following to a DataWindow object:

- A column that contains a database binary large object (a blob object) using OLE 2.0

- OLE 2.0 objects

For information on using OLE in a DataWindow object, see Chapter 31, "Using OLE in a DataWindow Object."

Adding reports to a DataWindow object

You can nest reports (nonupdatable DataWindow objects) in a DataWindow object.

For information on nesting reports, see Chapter 25, "Using Nested Reports."

Reorganizing controls in a DataWindow object

This section describes the activities that help you change the layout and appearance of the controls in a DataWindow object.

Displaying boundaries for controls in a DataWindow object

When reorganizing controls in the Design view, it is sometimes helpful to see how large all the controls are. That way you can easily check for overlapping controls and make sure that the spacing around controls is what you want.

❖ **To display control boundaries in a DataWindow object:**

1   Select Design>Options from the menu bar.

The Report Options dialog box displays.

2   Select the Show Edges check box.

PowerBuilder displays the boundaries of each control in the DataWindow object.

Boundaries display only in the Design view
The boundaries displayed for controls are for use only in the Design view. They do not display in a running DataWindow object or in a printed report.

Using the grid and the ruler in a DataWindow object

The DataWindow painter provides a grid and a ruler to help you align controls.

❖ **To use the grid and the ruler:**

1   Select Design>Options from the menu bar.

The Report Options dialog box displays. The Alignment Grid box contains the alignment grid options.

2   Use the options as needed:

Option	Meaning
Snap to Grid	Make controls snap to a grid position when you place them or move them.
Show Grid	Show or hide the grid when the workspace displays.
X	Specify the size (width) of the grid cells.
Y	Specify the size (height) of the grid cells.
Show Ruler	Show a ruler. The ruler uses the units of measurement specified in the Style dialog box. See "Changing the DataWindow object style" on page 540.

Your choices for the grid and the ruler are saved and used the next time you start PowerBuilder.

Deleting controls in a DataWindow object

❖ **To delete controls in a DataWindow object:**

1 Select the controls you want to delete.

2 Select Edit>Delete from the menu bar or press the Delete key.

Moving controls in a DataWindow object

In all presentation
styles except Grid

In all presentation styles except Grid, you can move all the controls (such as headings, labels, columns, graphs, and drawing controls) anywhere you want.

❖ **To move controls in a DataWindow object:**

1 Select the controls you want to move.

2 Do one of the following:

• Drag the controls with the mouse.

• Press an arrow key to move the controls in one direction.

In grid DataWindow
objects

You can reorder columns in a grid DataWindow object at runtime.

See "Working in a grid DataWindow object" on page 531.

Copying controls in a DataWindow object

You can copy controls within a DataWindow object and to other DataWindow objects. All properties of the controls are copied.

❖ **To copy a control in a DataWindow object:**

1 Select the control.

2 Select Edit>Copy from the menu bar.

The control is copied to a private PowerBuilder clipboard.

3 Copy (paste) the control to the same DataWindow object or to another one:

• To copy the control within the same DataWindow object, select Edit>Paste from the menu bar.

• To copy the control to another DataWindow object, open the desired DataWindow object and paste the control.

PowerBuilder pastes the control at the same location as in the source DataWindow object. If you are pasting into the same DataWindow object, you should move the pasted control so it does not cover the original control. PowerBuilder displays a message box if the control you are pasting is not valid in the destination DataWindow object.

Resizing controls in a DataWindow object

You can resize a control using the mouse or the keyboard. You can also resize multiple controls to the same size using the Layout drop-down toolbar on PainterBar2.

Using the mouse   To resize a control using the mouse, select it, then grab an edge and drag it with the mouse.

Using the keyboard   To resize a control using the keyboard, select it and then do the following:

To make the control	Press
Wider	Shift+Right Arrow
Narrower	Shift+Left Arrow
Taller	Shift+Down Arrow
Shorter	Shift+Up Arrow

In grid DataWindow objects   You can resize columns in grid DataWindow objects.

❖ **To resize a column in a grid DataWindow object:**

1   Position the mouse pointer at a column boundary.

The pointer changes to a two-headed arrow.

2   Press and hold the left mouse button and drag the mouse to move the boundary.

3   Release the mouse button when the column is the correct width.

Aligning controls in a DataWindow object

Often you want to align several controls or make them all the same size. You can use the grid to align the controls or you can have PowerBuilder align them for you.

❖ **To align controls in a DataWindow object:**

1 Select the control whose position you want to use to align the others.

PowerBuilder displays handles around the selected control.

2 Extend the selection by pressing and holding the Ctrl key and clicking the controls you want to align with the first one.

All the controls have handles on them.

3 Select Format>Align from the menu bar.

4 From the cascading menu, select the dimension along which you want to align the controls.

For example, to align the controls along the left side, select the first choice on the cascading menu. You can also use the Layout drop-down toolbar on PainterBar2.

PowerBuilder moves all the selected controls to align with the first one.

Equalizing the space between controls in a DataWindow object

If you have a series of controls and the spacing is fine between two of them but wrong for the rest, you can easily equalize the spacing around all the controls.

❖ **To equalize the space between controls in a DataWindow object:**

1 Select the two controls whose spacing is correct.

To do so, click one control, then press Ctrl and click the second control.

2 Select the other controls whose spacing match that of the first two controls. To do so, press Ctrl and click each control.

3 Select Format>Space from the menu bar.

4 From the cascading menu, select the dimension whose spacing you want to equalize.

You can also use the Layout drop-down toolbar on PainterBar2.

Equalizing the size of controls in a DataWindow object

Suppose you have several controls in a DataWindow object and want their sizes to be the same. You can accomplish this manually or by using the Format menu.

❖ **To equalize the size of controls in a DataWindow object:**

1   Select the control whose size is correct.

2   Press Ctrl and click to select the other controls whose size should match that of the first control.

3   Select Format>Size from the menu bar.

4   From the cascading menu, select the dimension whose size you want to equalize.

You can also use the Layout drop-down toolbar on PainterBar2.

Sliding controls to remove blank space in a DataWindow object

You can specify that you want to eliminate blank lines or spaces in a DataWindow object by sliding columns and other controls to the left or up if there is blank space. You can use this feature to remove blank lines in mailing labels or to remove extra spaces between fields (such as first and last name).

Slide is used by default in nested reports
PowerBuilder uses slide options automatically when you nest a report to ensure that the reports are positioned properly.

❖ **To use sliding columns or controls in a DataWindow object:**

1   Select Properties from the control's pop-up menu and then select the Position tab in the Properties view.

2   Select the Slide options you want:

Option	Description
Slide Left	Slide the column or control to the left if there is nothing to the left. Be sure the control does not overlap the control to the left. Sliding left will not work if the controls overlap.
Slide Up - All Above	Slide the column or control up if there is nothing in the row above. The row above must be completely empty for the column or control to slide up.
Slide Up - Directly Above	Slide the column or control up if there is nothing *directly above it* in the row above.

These options are also available on PainterBar2.

If you are sliding columns up
Even blank columns have height; if you want columns to slide up, you need to specify as Autosize Height all columns above them that might be blank and that you want to slide other columns up through.

Example

In a mailing label that includes first and last names, as well as address information, you can use sliding to combine the columns appropriately.

In the following label, emp_lname, the comma, state, and zip_code are specified as slide left. Edges are shown to indicate the spacing between the columns. Notice that there is a small amount of space between controls. This space is necessary for Slide Left to work properly:
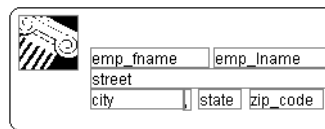
When you preview (run) the DataWindow object, the last name, comma, state, and zip code slide left to remove the blank space:
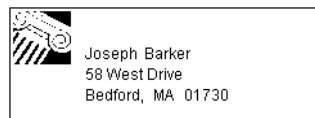
Positioning controls in a DataWindow object

Table 20-3 shows the properties for each control in a DataWindow object that determine how it is positioned within the DataWindow object.

Table 20-3: Position properties for controls in a DataWindow object

Property	Meaning
Background	Control is behind other controls. It is not restricted to one band. This is useful for adding a watermark (such as the word CONFIDENTIAL) to the background of a report.
Band	Control is placed within one band. It cannot extend beyond the band's border.
Foreground	Control is in front of other controls. It is not restricted to one band.
Moveable	Control can be moved at runtime and in preview. This is useful for designing layout.
Resizable	Control can be resized at runtime and in preview. This is useful for designing layout.
HideSnaked	Control appears only in the first column on the page; in subsequent columns the control does not appear. This is only for newspaper columns, where the entire DataWindow object snakes from column to column (set on the General page of the Properties view).

Default positioning

PowerBuilder uses the defaults shown in Table 20-4 when you place a new control in a DataWindow object.

Table 20-4: Default position properties for controls in a DataWindow object

Control	Default positioning
Graph	Foreground, movable, resizable
All other controls	Band, not movable, not resizable

❖ **To change the position of a control in a DataWindow object:**

1 Select Properties from the control's pop-up menu and then select the Position tab.

2 From the Layer option drop-down list, select Background, Band, or Foreground.

3 Select Resizable or Moveable as appropriate.

Rotating controls in a DataWindow object

Controls that display text such as text controls, columns, and computed fields can be rotated from the original baseline of the text. The Escapement option on the Font property page for the control lets you specify the amount of rotation, also known as escapement.

Several other properties of a rotated control affect its final placement when the DataWindow object runs. The location of the control in Design view, the amount of rotation specified for it, and the location of the text within the control (for example, centered text as opposed to left-aligned text) all contribute to what you see in the DataWindow object Preview view.

The following procedure includes design practices that help ensure that you get the final results you want. As you become more experienced, you can drop or alter some of the steps. The procedure recommends setting a visible border on the control so that you can see where the control is located in the Preview view and making the control movable in the Preview view, which is often helpful.

❖ **To rotate a control in a DataWindow object:**

1   Select the control in the Design view.

2   Change its border to Box (General property page>Border>Box) and make it movable (Position property page>Moveable check box).

3   In Design view, enlarge the area in which the control is placed.

    For example, in a grid DataWindow object, make the band deeper and move the control down into the center of the band.

4   Display the Modify expression dialog box for the Escapement property. (Click the button next to the Escapement property on the Font property page.)

5   Specify the amount of rotation you want as an integer in tenths of a degree. (For example, 450 means 45 degrees of rotation; 0 means horizontal or no rotation.)

The origin of rotation is the center of the top border of the box containing the text. It is often helpful to use left-aligned text (General property page>Alignment>Left) because it makes it easier to position the control correctly. This example shows text centered within the control.
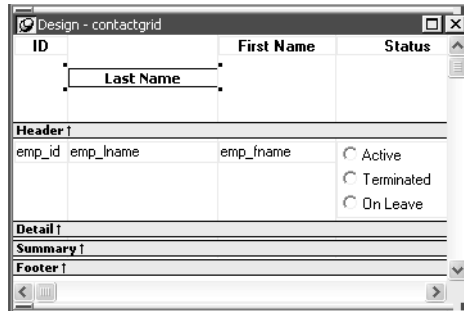
If the box that contains the text overlaps the border of the page or the border of a label in a DataWindow object with the Label presentation style, the origin of rotation is the center of the portion of the top border that is within the page or label, and the portion that is outside the page or label is cut off. This can cause the text in the box to run to a second line when it is rotated. If you want the text to display close to the border, you can add one or more line breaks ("~r~n") before the text and adjust the size of the box.

6   To display the current rotation in Preview, close the Preview view and reopen it (View>Preview on the menu bar).
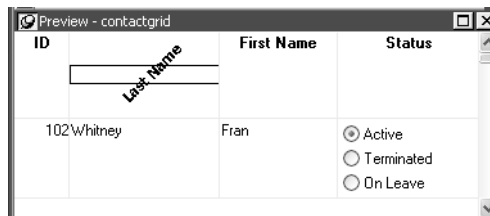
7   Drag and drop the control in the Preview view or Design view until it is where you want it.

8   In Design view, select the control that is being rotated, remove the temporary border, and deselect the Moveable check box.

If you are using a conditional expression for rotation
If you are specifying different rotations depending on particular conditions, you might need to add conditions to the x and y properties for the control to move the control conditionally to match the various amounts of rotation. An alternative to moving the control around is to have multiple controls positioned exactly as you want them, taking into account the different amounts of rotation. Then you can add a condition to the visible property of each control to ensure that the correctly rotated control shows.

Controlling Updates in DataWindow Objects

About this chapter

When PowerBuilder generates the basic DataWindow object, it defines whether the data is updatable. This chapter describes the default settings and how you can modify them.

Contents

About controlling updates

When PowerBuilder generates the basic DataWindow object, it defines whether the data is updatable by default as follows:

- If the DataWindow object contains columns from a single table and includes that table's key columns, PowerBuilder defines all columns as updatable and specifies a nonzero tab order for each column, allowing users to tab to the columns.

- If the DataWindow object contains columns from two or more tables or from a view, PowerBuilder defines all columns as not being updatable and sets all tab orders to zero, preventing users from tabbing to them.

You can accept the default settings or modify the update characteristics for a DataWindow object.

If using a Stored Procedure or External data source
If the data source is Stored Procedure or External, you can use the
GetNextModified method to write your own update script. For more
information, see the *DataWindow Reference*.

What you can do

You can:

- Allow updates in a DataWindow object associated with multiple tables or
  a view; you can define one of the tables as being updatable

- Prevent updates in a DataWindow object associated with one table

- Prevent updates to specific columns in a DataWindow object that is
  associated with an updatable table

- Specify which columns uniquely identify a row to be updated

- Specify which columns will be included in the WHERE clause of the
  UPDATE or DELETE statement PowerBuilder generates to update the
  database

- Specify whether PowerBuilder generates an UPDATE statement, or a
  DELETE then an INSERT statement, to update the database when users
  modify the values in a key column

Updatability of views
Some views are logically updatable; some are not. For the rules your DBMS
follows for updating views, see your DBMS documentation.

❖ **To specify update characteristics for a DataWindow object:**

1   Select Rows>Update Properties from the menu bar.

     The Specify Update Properties dialog box displays.

2   To prevent updates to the data, make sure the Allow Updates box is not
     selected.

     To allow updates, select the Allow Updates box and specify the other
     settings as described below.

3   Click OK.

Changing tab values
PowerBuilder does not change the tab values associated with columns after you change the update characteristics of the DataWindow object. If you have allowed updates to a table in a multitable DataWindow object, you should change the tab values for the updatable columns so that users can tab to them.

For more information, see "Defining the tab order in a DataWindow object" on page 547.

Specifying the table to update

Each DataWindow object can update one table, which you select from the Table to Update box in the Specify Update Properties dialog box.

Specifying the unique key columns

The Unique Key Columns box in the Specify Update Properties dialog box specifies which columns PowerBuilder uses to identify a row being updated. PowerBuilder uses the column or columns you specify here as the key columns when generating the WHERE clause to update the database (as described below):
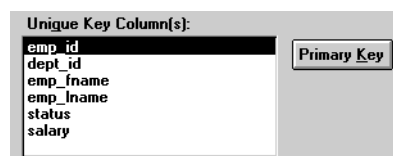
The key columns you select here must uniquely identify a row in the table. They can be the table's primary key, though they don't have to be.

Using the primary key
Clicking the Primary Key button cancels any changes in the Unique Key Columns box and highlights the primary key for the updatable table.

Specifying an identity column

Many DBMSs allow you to specify that the value for a column in a new row is to be automatically assigned by the DBMS. This kind of column is called an identity column. Different DBMSs provide different types of identity columns.

For example, some DBMSs allow you to define autoincrement columns so that the column for a new row is automatically assigned a value one greater than that of the previous highest value. You could use this feature to specify that an order number be automatically incremented when someone adds a new order:
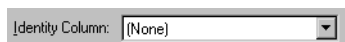
By specifying an identity column in the Specify Update Properties dialog box, you tell PowerBuilder to bring back the value of a new row's identity column after an insert in the DataWindow object so that users can see it.

For information about identity columns in your DBMS, see your DBMS documentation.

Specifying updatable columns

You can make all or some of the columns in a table updatable.

Updatable columns are displayed highlighted. Click a nonupdatable column to make it updatable. Click an updatable column to make it nonupdatable.

Changing tab values
If you have changed the updatability of a column, you should change its tab value. If you have allowed a column to be updated, you should change its tab value to a nonzero number so users can tab to it.

Specifying the WHERE clause for update/delete

Sometimes multiple users are accessing the same tables at the same time. In these situations, you need to decide when to allow your application to update the database. If you allow your application to always update the database, it could overwrite changes made by other users:
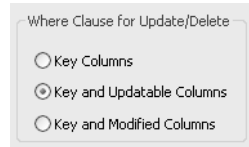
You can control when updates succeed by specifying which columns PowerBuilder includes in the WHERE clause in the UPDATE or DELETE statement used to update the database:

```
UPDATE table...
SET column = newvalue
WHERE col1 = value1
AND col2 = value2 ...

DELETE
FROM table
WHERE col1 = value1
AND col2 = value2 ...
```

Using timestamps
Some DBMSs maintain timestamps so you can ensure that users are working with the most current data. If the SELECT statement for the DataWindow object contains a timestamp column, PowerBuilder includes the key column and the timestamp column in the WHERE clause for an UPDATE or DELETE statement regardless of which columns you specify in the Where Clause for Update/Delete box.

If the value in the timestamp column changes (possibly due to another user modifying the row), the update fails.

To see whether you can use timestamps with your DBMS, see *Connecting to Your Database*.

Choose one of the options in Table 21-1 in the Where Clause for Update/Delete box. The results are illustrated by an example following the table.

Table 21-1: Specifying the WHERE clause for UPDATE and DELETE

Option	Result
Key Columns	The WHERE clause includes the key columns only. These are the columns you specified in the Unique Key Columns box.
	The values in the originally retrieved key columns for the row are compared against the key columns in the database. No other comparisons are done. If the key values match, the update succeeds.
	Caution Be very careful when using this option. If you tell PowerBuilder only to include the key columns in the WHERE clause and someone else modified the same row after you retrieved it, their changes will be overwritten when you update the database (see the example following this table).
	Use this option only with a single-user database or if you are using database locking. In other situations, choose one of the other two options described in this table.
Key and Updatable Columns	The WHERE clause includes all key and updatable columns.
	The values in the originally retrieved key columns and the originally retrieved updatable columns are compared against the values in the database. If any of the columns have changed in the database since the row was retrieved, the update fails.
Key and Modified Columns	The WHERE clause includes all key and modified columns.
	The values in the originally retrieved key columns and the modified columns are compared against the values in the database. If any of the columns have changed in the database since the row was retrieved, the update fails.

Example

Consider this situation: a DataWindow object is updating the Employee table, whose key is Emp_ID; all columns in the table are updatable. Suppose the user has changed the salary of employee 1001 from $50,000 to $65,000. This is what happens with the different settings for the WHERE clause columns:

• If you choose Key Columns for the WHERE clause, the UPDATE statement looks like this:

```
UPDATE Employee
SET Salary = 65000
WHERE Emp_ID = 1001
```

This statement will succeed *regardless of whether other users have modified the row since your* application *retrieved the row*. For example, if another user had modified the salary to $70,000, that change will be overwritten when your application updates the database.

• If you choose Key and Modified Columns for the WHERE clause, the UPDATE statement looks like this:

```
UPDATE Employee
SET Salary = 65000
WHERE Emp_ID = 1001
   AND Salary = 50000
```

Here the UPDATE statement is also checking the original value of the modified column in the WHERE clause. The statement will fail if another user changed the salary of employee 1001 since your application retrieved the row.

• If you choose Key and Updatable Columns for the WHERE clause, the UPDATE statement looks like this:

```
UPDATE Employee
SET Salary = 65000
WHERE Emp_ID = 1001
   AND Salary = 50000
   AND Emp_Fname = original_value
   AND Emp_Lname = original_value
   AND Status = original_value
   ...
```

Here the UPDATE statement is checking all updatable columns in the WHERE clause. This statement will fail if any of the updatable columns for employee 1001 have been changed since your application retrieved the row.

Specifying update when key is modified

The Key Modification property determines the SQL statements PowerBuilder generates whenever a key column—a column you specified in the Unique Key Columns box—is changed. The options are:

• Use DELETE then INSERT (default)

• Use UPDATE

How to choose a setting

Consider the following when choosing the Key Modification setting:

- If multiple rows are changed, DELETE and INSERT always work. In some DBMSs, UPDATE fails if the user modifies two keys and sets the value in one row to the original value of the other row.

- You might choose the setting here based on your DBMS triggers. For example, if there is an Insert trigger, select Use Delete then Insert.

- If only one row can be modified by the user before the database is updated, use UPDATE because it is faster.

Using stored procedures to update the database

Updates to the database can be performed using stored procedures.

Why use stored procedures?

The DataWindow control submits updates to the database by dynamically generating INSERT, DELETE, and UPDATE SQL statements after determining the status of each row in the DataWindow object. You can also define procedural SQL statements in a stored procedure for use by all applications accessing a database. Using stored procedures to perform database updates allows you to enhance database security, integrity, and performance. Since stored procedures provide for conditional execution, you can also use them to enforce additional business rules.

Updating using stored procedures

The Stored Procedure Update dialog box only allows you to associate an existing stored procedure with your DataWindow object. The stored procedure must have been previously defined in the database.
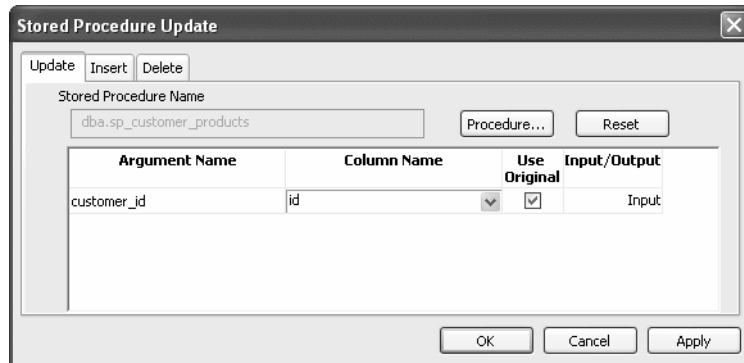
❖ **To use stored procedures to update the database**

1   In the DataWindow painter, select Rows>Stored Procedure Update to display the Stored Procedure Update dialog box.

2   Select the tab for the SQL update method (Delete, Insert, or Update) with which you want to associate a stored procedure.

3   Click the Procedure button, select the stored procedure you want to have execute when the SQL update method is generated, and click OK.

The parameters used in the stored procedure are displayed in the Argument Name list in the order in which they are defined in the procedure. Column Name lists the columns used in your DataWindow object.

4   Associate a column in the DataWindow object or an expression with a procedure parameter.

If a stored procedure uses parameters that are not matched to column names, you can substitute the value from a DataWindow object computed field or expression.

Matching a column to a procedure parameter
You must be careful to correctly match a column in the DataWindow object to a procedure parameter, since PowerBuilder is able to verify only that datatypes match.

5   If the parameter is to receive a column value, indicate whether the parameter will receive the updated column value entered through the DataWindow object or retain the original column value from the database.

Typically, you select Use Original when the parameter is used in a WHERE clause in an UPDATE or DELETE SQL statement. If you do not select Use Original, the parameter will use the new value entered for that column. Typically, you would use the new value when the parameter is used in an INSERT or UPDATE SQL statement.

What happens when the stored procedure is executed
The stored procedure you associate with a SQL update method in the Stored Procedure Update dialog box is executed when the DataWindow control calls the Update method. The DataWindow control examines the table in the DataWindow object, determines the appropriate SQL statement for each row, and submits the appropriate stored procedure (as defined in the Stored Procedure Update dialog box) with the appropriate column values substituted for the procedure arguments.

If a stored procedure for a particular SQL update method is not defined, the DataWindow control submits the appropriate SQL syntax in the same manner it always has.

Return values from procedures cannot be handled by the DataWindow control. The Update method returns 1 if it succeeds and -1 if an error occurs. Additional information is returned to SQLCA.

Using Describe and Modify

You can use the DataWindow Describe and Modify methods to access DataWindow property values including the stored procedures associated with a DataWindow object. For information, see the DataWindow object property Table.property in the *DataWindow Reference*.

Restrictions on the use of Modify

Since a database driver can only report stored procedure names and parameter names and position, it cannot verify that changes made to stored procedures are valid. Consequently, if you use Modify to change a stored procedure, be careful that you do not inadvertently introduce changes into the database.

In addition, using Modify to enable a DataWindow object to use stored procedures to update the database when it is not already using stored procedures requires that the type qualifier be specified first. Calling the type qualifier ensures that internal structures are built before subsequent calls to Modify. If a new method or method arguments are specified without a preceding definition of type, Modify fails.

Displaying and Validating Data

About displaying and validating data

When PowerBuilder generates a basic DataWindow object, it uses the extended attributes defined for the data and stored in the extended attribute system tables.

For more information about the extended attribute system tables, see Appendix A, "The Extended Attribute System Tables."

In the Database painter, you can create the extended attribute definitions that specify a column's display format, edit style, and validation rules.

In the DataWindow painter, you can override these extended attribute definitions for a column in a DataWindow object. These overrides do not change the information stored with the column definition in the extended attribute system tables.

Presenting the data

When you generate a new DataWindow object, PowerBuilder presents the data according to the properties already defined for a column, such as a column's display format and edit style.

Display formats

Display formats embellish data values while still displaying them as letters, numbers, and special characters. Using display formats, for example, you can:

- Change the color of numbers to display a negative value

- Add parentheses and dashes to format a telephone number

- Add a dollar sign and period to indicate a currency format

For information, see "About display formats" on page 597.

Edit styles

Edit styles usually take precedence over display formats and specify how column data is presented. For example, using edit styles, you can:

- Display valid values in a drop-down list

- Indicate that a single value is selected by a check box

- Indicate which of a group of values is selected with radio buttons

Edit styles affect not only the way data displays, they also affect how the user interacts with the data at runtime.

For more information, see "About edit styles" on page 608.

About display format masks and EditMask masks

The differences between display format masks and EditMask masks can be confusing. A display format mask determines the appearance of the column when the focus is *off* the column, or when the DataWindow object is in print preview mode. When you apply an EditMask edit style, the mask you use determines the appearance of the column when focus is *on* the column.

If you want data to display differently depending on whether the focus is on or off the column, specify an edit mask (on the Edit property page for the column) as well as a display format (on the Format property page for the column), then check the Use Format check box on the Format property page. The Use Format check box displays only when an edit mask has been specified.

If you want the data to display in the same way whether focus is on or off the column and you have defined an edit mask, you do not need to define a display format. The edit mask is used for display if the Use Format box is not checked (the default).

Validating data
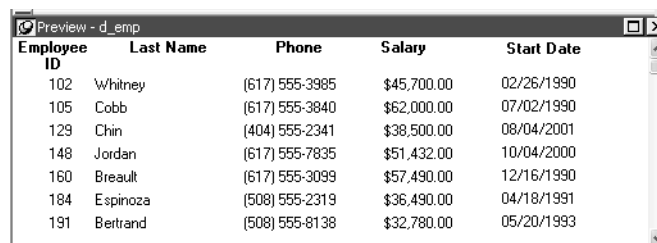
When data is entered in the Database painter or in a DataWindow object, PowerBuilder evaluates the data against validation rules defined for that column. If the data is valid, PowerBuilder accepts the entry; otherwise, PowerBuilder displays an error message and does not accept the entry.

For more information, see "About validation rules" on page 625.

About display formats

You can use display formats to customize the display of column data in a DataWindow object. Display formats are masks in which certain characters have special significance. For example, you can display currency values preceded by a dollar sign, show dates with month names spelled out, and use a special color for negative numbers. PowerBuilder comes with many predefined display formats. You can use them as is or define your own.

Here the Phone, Salary, and Start Date columns use display formats so the data is easier to interpret:

Display formats not used for data entry
When users tab to a column containing a display format, PowerBuilder removes the display format and displays the raw value for users to edit.

If you want to provide formatting used for data entry, you need to specify edit masks, as described in "The EditMask edit style" on page 616.

Working with display formats

You work with display formats in the Database painter and the DataWindow painter.

What you do in the Database painter

In the Database painter, you can:

•   Create, modify, and delete named display formats

    The named display formats are stored in the extended attribute system tables. When you have defined a display format, it can be used by any column of the appropriate datatype in the database.

•   Assign display formats to columns and remove them from columns

    These formats are used by default when you place the column in a DataWindow object in the DataWindow painter.

What you do in the DataWindow painter

In the DataWindow painter, you can:

•   Accept the default display format assigned to a column in the Database painter

•   Override the default display format with another named format stored in the extended attribute system tables

•   Create an ad hoc, unnamed format to use with one specific column

Display formats and the extended attribute system tables

When you have placed a column in a DataWindow object and have given it a display format (either the default format from the assignment made in the Database painter for the column or a format assigned in the DataWindow painter), there is no longer any link to the named format in the extended attribute system tables.

If the definition of the display format later changes in the extended attribute system tables, the format for the column in a DataWindow object does not change. If you want to use the modified format, you can reapply it to the column in the DataWindow painter.

Working with display formats in the Database painter

Typically, you define display formats and associate them with columns in the Database painter, because display formats are properties of the data itself. Once you have associated a display format with a column in the Database painter, it is used by default each time the column is placed in a DataWindow object.

Edit style takes precedence

If a column has an associated edit style, the edit style takes precedence over a display format unless you use an EditMask edit style and check the Use Format box on the Format property page.

For more information, see "About edit styles" on page 608.

❖ **To create a new display format:**

1   In the Database painter, select Object>Insert>Display Format from the menu bar.

The Display Format view displays.

2   Name the display format and specify a datatype.

3   Define the display format using masks.

For information, see "Defining display formats" on page 601.

You can use this display format with any column of the appropriate datatype in the database.

❖ **To modify an existing display format:**

1   In the Database painter, open the Extended Attributes view.

2   In the Extended Attributes view, open the list of display formats.

3   Position the pointer on the display format you want to modify, display the pop-up menu, and select Properties.

4   In the Display Format view, modify the display format as desired.

For information, see "Defining display formats" on page 601.

❖ **To associate a display format with a column in the Database painter:**

1   In the Database painter Objects view, position the pointer on the column, select Properties from the pop-up menu, and select the Display tab in the Properties view.

2   Select a format from the list in the Display Format box.

The column now has the selected format associated with it in the extended attribute system tables.

❖ **To remove a display format from a column in the Database painter:**

1 In the Database painter Objects view, position the pointer on the column, select Properties from the pop-up menu, and select the Display tab in the Properties view.

2 Select (None) from the list in the Display Format box.

The display format is no longer associated with the column.

Working with display formats in the DataWindow painter

Display formats you assign to a column in the Database painter are used by default when you place the column in a DataWindow object. You can override the default format in the DataWindow painter by choosing another format from the extended attribute system tables or defining an ad hoc format for one specific column.

About computed fields
You can assign display formats to computed fields using the same techniques as for columns in a table.

❖ **To specify a display format for a column in the DataWindow painter:**

1 In the DataWindow painter, move the pointer to the column, select Properties from the column's pop-up menu, and then select the Format tab.

Information appropriate to the datatype of the selected column displays. The currently used format displays in the Format box. All formats for the datatype defined in the extended attribute system tables are listed in the pop-up list (displayed by clicking the button).

2 Do one of the following:

   • Delete the display format.

   • Select a format in the extended attribute system tables from the pop-up list.

   • Create a format for the column by typing it in the Format box. For more information, see "Defining display formats" next.

Format not saved in the extended attribute system tables
If you create a format here, it is used only for the current column and is not saved in the extended attribute system tables.

Shortcuts
To assign the Currency or Percent display format to a numeric column in a report, select the column, then click the Currency or Percent button in the PainterBar or select Format>Currency or Format>Percent from the menu bar.

Customizing the
toolbar

You can add buttons to the PainterBar that assign a specified display format to selected columns in reports.

For more information, see "Customizing toolbars" on page 49.

Defining display formats

Display formats are represented through masks, where certain characters have special significance. PowerBuilder supports four kinds of display formats, each using different mask characters:

    Numbers
    Strings
    Dates
    Times

For example, in a string format mask, each @ represents a character in the string and all other characters represent themselves. You can use the following mask to display phone numbers:

    (@@@) @@@-@@@@

Combining formats

You can include different types of display format masks in a single format. Use a space to separate the masks. For example, the following format section includes a date and time format:

    mmmm/dd/yyyy h:mm

Using sections

Each type of display format can have multiple sections, with each section corresponding to a form of the number, string, date, or time. Only one section is required; additional sections are optional and should be separated with semicolons (;). You cannot use sections in edit masks. Semicolons can be used only in display formats.

The following format specifies different displays for positive and negative numbers—negative numbers are displayed in parentheses:

    $#,##0;($#,##0)

Using keywords

Enclose display format keywords in square brackets. For example, you can use the keyword [General] when you want PowerBuilder to determine the appropriate format for a number.

Using colors

You can define a color for each display format section by specifying a color keyword before the format. The color keyword is the name of the color, or a number that represents the color, enclosed in square brackets: [RED] or [255]. The number is usually used only when a color is required that is not provided by name. The named color keywords are:

[BLACK]
[BLUE]
[CYAN]
[GREEN]
[MAGENTA]
[RED]
[WHITE]
[YELLOW]

The formula for combining primary color values into a number is:

256*256*$blue$ + 256*$green$ + red=$number$

where the amount of each primary color is specified as a value from 0 to 255. For example, to specify cyan, substitute 255 for blue, 255 for green, and 0 for red. The result is 16776960.

If you want to add text to a numeric display format and use a color attribute, you must include the escape character (\) before each literal in the mask. For example:

```
[red]\D\e\p\t\: ###
```

Table 22-1 lists the blue, green, and red values you can use in the formula to create other colors.

Table 22-1: Numeric values used to create colors

Blue	Green	Red	Number	Color
0	0	255	255	Red
0	255	0	65280	Green
0	128	0	32768	Dark green
255	0	0	16711680	Blue
0	255	255	65535	Yellow
0	128	128	32896	Brown
255	255	0	16776960	Cyan
192	192	192	12632256	Light gray

Using special characters	To include a character in a mask that has special meaning in a display format, such as [, precede the character with a backslash (\\). For example, to display a single quotation mark, enter \\'.
Setting display formats at runtime	In scripts, you can use GetFormat to get the current format for a column and SetFormat to change the format for a column at runtime.

Number display formats

A number display format can have up to four sections. Only the first is required. The three other sections determine how the data displays if its value is negative, zero, or NULL. The sections are separated by semi-colons:

> Positive-format;negative-format;zero-format;null-format

Special characters

Table 22-2 lists characters that have special meaning in number display formats.

Table 22-2: Characters with special meaning in display formats

Character	Meaning
#	A number
0	A required number; a number will display for every 0 in the mask

Percent signs, decimal points, parentheses, and spaces display as entered in the mask.

Use at least one 0

In general, a number display format should include at least one 0. If users enter 0 in a field with the mask ###, the field will appear to be blank if you do not provide a zero-format section. If the mask is ###.##, only the period displays. If you want two decimal places to display even if both are 0, use the mask ##0.00.

Number keywords

You can use the following keywords as number display formats when you want PowerBuilder to determine an appropriate format to use:

- [General]

- [Currency]

Note that [Currency(7)] and [Currency(n)] are legal edit masks, but they are *not* legal display formats.

Number and currency settings	To ensure that an application behaves the same in every country where it is deployed, DataWindow expressions and the masks used in display formats and edit masks require U.S. notation for numbers. That is, when you specify a number in a DataWindow expression or in a number mask, a comma always represents the thousands delimiter and a period always represents the decimal place. You should also always use the $ sign to represent the symbol for currency.

At runtime, the locally correct symbols are displayed for numbers and currency. The comma and period are replaced by the delimiters defined in the user's Number settings in the Regional or International Settings property sheet in the Windows Control Panel. The $ sign in the mask is replaced by the local currency symbol as defined in the user's Currency setting in the Windows Control Panel. For example, in countries where a comma represents the decimal place and a period represents thousands, users see numbers in those formats.

Percentages	Use caution when defining an edit mask for a percentage. When you enter a number in a column with a percent edit mask and tab off the column, PowerBuilder divides the number by 100 and stores the result in the buffer. For example, if you enter 23, PowerBuilder passes .23 to the buffer. When you retrieve from the database, PowerBuilder multiplies the number by 100 and, if the mask is ##0%, displays 23%.

The datatype for the column must be numeric or decimal to handle the result of a division by 100. If the column has an integer datatype, a percentage entered as 333 is retrieved from the database as 300, and 33 is retrieved as 0.

If you use an edit mask with decimals, such as ##0.00%, the datatype must have enough decimal places to handle the division. For example, if you enter 33.33, the datatype for the column must have at least four decimal places because the result of the division is .3333. If the datatype has only three decimal places, the percentage is retrieved as 33.30.

Examples	Table 22-3 shows how the values 5, –5, and .5 display when different format masks are applied.

Table 22-3: Number display format examples

Format	5	-5	.5
[General]	5	-5	0.5
0	5	-5	1
0.00	5.00	-5.00	0.50
#,##0	5	-5	1
#,##0.00	5.00	-5.00	0.50

Format	5	-5	.5
$#,##0;($#,##0)	$5	($5)	$1
$#,##0;-$#,##0	$5	-$5	$1
$#,##0;[RED]($#,##0)	$5	($5)	$1
[Currency]	$5.00	($5.00)	$0.50
$#,##0.00;($#,##0.00)	$5.00	($5.00)	$0.50
$#,##0.00;[RED]($#,##0.00)	$5.00	($5.00)	$0.50
##0%	500%	-500%	50%
##0.00%	500.00%	-500.00%	50.00%
0.00E+00	5.00E+00	-5.00E+00	5.00E-01

String display formats

String display formats can have two sections. The first is required and contains the format for strings; the second is optional and specifies how to represent NULLs:

   string-format;null-format

In a string format mask, each at-sign (@) represents a character in the string and all other characters represent themselves.

Example   This format mask:

   [red](@@@) @@@-@@@@

displays the string 800YESCELT in red as:

   (800) YES-CELT

Date display formats

Date display formats can have two sections. The first is required and contains the format for dates; the second is optional and specifies how to represent NULLs:

date-format;null-format

Special characters

Table 22-4 shows characters that have special meaning in date display formats.

Table 22-4: Characters with special meaning in data display formats

Character	Meaning	Example
d	Day number with no leading zero	9
dd	Day number with leading zero if appropriate	09
ddd	Day name abbreviation	Mon
dddd	Day name	Monday
m	Month number with no leading zero	6
mm	Month number with leading zero if appropriate	06
mmm	Month name abbreviation	Jun
mmmm	Month name	June
yy	Two-digit year	97
yyyy	Four-digit year	1997

Colons, slashes, and spaces display as entered in the mask.

About 2-digit years

If users specify a 2-digit year in a DataWindow object, PowerBuilder assumes the date is the 20th century if the year is greater than or equal to 50. If the year is less than 50, PowerBuilder assumes the 21st century. For example:

- 1/1/85 is interpreted as January 1, 1985.

- 1/1/40 is interpreted as January 1, 2040.

Date keywords

You can use the following keywords as date display formats when you want PowerBuilder to determine an appropriate format to use:

- [ShortDate]

- [LongDate]

The format used is determined by the regional settings for date in the registry. Note that [Date] is not a valid display format.

Examples

Table 22-5 shows how the date Friday, January 30, 1998, displays when different format masks are applied.

Table 22-5: Date display format examples

Format	Displays
[red]m/d/yy	1/30/98 in red
d-mmm-yy	30-Jan-98
dd-mmmm	30-January
mmm-yy	Jan-98
dddd, mmm d, yyyy	Friday, Jan 30, 1998

Time display formats

Time display formats can have two sections. The first is required and contains the format for times; the second is optional and specifies how to represent NULLs:

> time-format;null-format

Special characters

Table 22-6 shows characters that have special meaning in time display formats.

Table 22-6: Characters with special meaning in time display formats

Character	Meaning
h	Hour with no leading zero (for example, 1)
hh	Hour with leading zero if appropriate (for example, 01)
m	Minute with no leading zero (must follow h or hh)
mm	Minute with leading zero if appropriate (must follow h or hh)
s	Second with no leading zero (must follow m or mm)
ss	Second with leading zero (must follow m or mm)
ffffff	Microseconds with no leading zeros. You can enter one to six f's; each f represents a fraction of a second (must follow s or ss)
AM/PM	Two-character, uppercase abbreviation (AM or PM as appropriate)
am/pm	Two-character, lowercase abbreviation (am or pm as appropriate)
A/P	One-character, uppercase abbreviation (A or P as appropriate)
a/p	One-character, lowercase abbreviation (a or p as appropriate)

Colons, slashes, and spaces display as entered in the mask.

24-hour format is the default
Times display in 24-hour format unless you specify AM/PM, am/pm, A/P, or a/p.

Time keyword

You can use the following keyword as a time display format to specify the format specified in the Windows control panel:

- [Time]

Examples

Table 22-7 shows how the time 9:45:33:234567 PM displays when different format masks are applied.

Table 22-7: Time display format examples

Format	Displays
h:mm AM/PM	9:45 PM
hh:mm A/P	09:45 P
h:mm:ss am/pm	9:45:33 pm
h:mm	21:45
h:mm:ss	21:45:33
h:mm:ss:f	21:45:33:2
h:mm:ss:fff	21:45:33:234
h:mm:ss:ffffff	21:45:33:234567
m/d/yy h:mm	1/30/98 21:45

About edit styles

You can define edit styles for columns. Edit styles specify how column data is presented in DataWindow objects. Unlike display formats, edit styles do not only affect the display of data; they also affect how users interact with the data at runtime. Once you define an edit style, it can be used by any column of the appropriate datatype in the database.
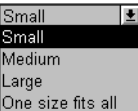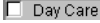
When edit styles are used

If both a display format and an edit style have been assigned to a column, the edit style is always used, with one exception. When you assign an EditMask edit style to a column, you can check the Use Format check box on the Format property page for the column to use the edit mask format when focus is on the column, and the display format mask when focus is off the column.

Edit styles

Table 22-8 shows the available edit styles.

Table 22-8: Edit styles

Edit style	What the edit style does	Example
Edit box (default)	Displays a value in the box For data entry, type a value	Boston
DropDownListBox	Displays a value from the drop-down list For data entry, select or enter a value	Small ▾ Small Medium Large One size fits all
CheckBox	Displays a check box selected or cleared For data entry, select or clear the check box	☐ Day Care
RadioButtons	Displays radio buttons, one of which is selected For data entry, select one of the radio buttons	⦿ Male ○ Female
EditMask	Displays formatted data For data entry, type a value	$62,000.00
DropDownDataWindow	Displays a value from a drop-down DataWindow For data entry, select a value	100 ▾ R & D Sales Finance
InkEdit	On Tablet PCs, displays an InkEdit control so the user can enter data with the stylus.	

For example, suppose you have a column Status that takes one of three values: the letters A, T, and L, each representing a status (Active, Terminated, or On Leave). If you assign it the RadioButton edit style, userscan simply click a button instead of having to type A, T, or L. You do not have to create a validation rule to validate typed input.

Working with edit styles

You work with edit styles in the Database painter and DataWindow painter.

What you do in the
Database painter

In the Database painter, you can:

• Create, modify, and delete named edit styles

The edit styles are stored in the extended attribute system tables. Once you define an edit style, it can be used by any column of the appropriate datatype in the database.

• Assign edit styles to columns

These styles are used by default when you place the column in a DataWindow object in the DataWindow painter.

What you do in the
DataWindow painter

In the DataWindow painter, you can:

• Accept the default edit style assigned to a column in the Database painter

• Override the default edit style with another named style stored in the extended attribute system tables

• Create an ad hoc, unnamed edit style to use with one specific column

Edit styles and the
extended attribute
system tables

When you have placed a column in a DataWindow object and have given it an edit style (either the default style from the assignment made in the Database painter for the column or a style assigned in the DataWindow painter), PowerBuilder records the name and definition of the edit style in the DataWindow object.

However, if the definition of the edit style later changes in the extended attribute system tables, the edit style for the column in a DataWindow object will not change automatically. You can update the column by reassigning the edit style to it in the DataWindow object.

Working with edit styles in the Database painter

Typically, you define edit styles in the Database painter, because edit styles are properties of the data itself. Once defined in the Database painter, the styles are used by default each time the column is placed in a DataWindow object.

❖ **To create a new edit style:**

1   In the Database painter, select Object>Insert>Edit Style from the menu bar.

2   In the Edit Style dialog box, select the edit style type from the Style drop-down list.

3   Specify the properties of the edit style and click OK.

For information, see "Defining edit styles" on page 612.

You can use the new edit style with any column of the appropriate datatype in the database.

❖ **To modify an existing edit style:**

1   In the Database painter, open the Extended Attributes view.

2   In the Extended Attributes view, open the list of edit styles.

3   Position the pointer on the Edit style you want to modify, display the pop-up menu, then select Properties.

4   In the Edit Style dialog box, modify the edit style as desired and click OK.

For information, see "Defining edit styles" on page 612.

You can use the modified edit style with any column of the appropriate datatype in the database.

❖ **To associate an edit style with a column in the Database painter:**

1   In the Database painter (Objects view), position the pointer on the column, select Properties from the pop-up menu, then select the Edit Style tab in the Properties view.

2   Select a style for the appropriate datatype from the list in the Style Name box.

PowerBuilder associates the selected edit style with the column in the extended attribute system tables.

❖ **To remove an edit style from a column in the Database painter:**

1   In the Database painter (Objects view), position the pointer on the column, select Properties from the pop-up menu, then select the Edit Style tab in the Properties view.

2   Select (None) from the list in the Style Name box.

The edit style is no longer associated with the column.

Working with edit styles in the DataWindow painter

An edit style you assign to a column in the Database painter is used by default when you place the column in a DataWindow object. You can override the edit style in the DataWindow painter by choosing another edit style from the extended attribute system tables or defining an ad hoc style for one specific column.

❖ **To specify an edit style for a column:**

1   In the DataWindow painter, move the pointer to the column, select Properties from the column's pop-up menu, and then select the Edit tab.

2   Select the type of edit style you want from the Style Type drop-down list.

The information on the Edit property page changes to be appropriate to the type of edit style you selected.

3   Do one of the following:

- Select an edit style from the Style Name box.

- Create an ad hoc edit style for the column, as described in "Defining edit styles" next.

Defining edit styles

This section describes how to specify each type of edit style.

The Edit edit style

By default, columns use the Edit edit style, which displays data in an edit control. You can customize the appearance and behavior of the edit control by modifying a column's Edit edit style. To do so, select Edit in the Style Type drop-down list and specify the properties for that style:

- To restrict the number of characters users can enter, enter a value in the Limit box.

- To convert the case of characters upon display, enter an appropriate value in the Case box.

- To have entered values display as asterisks for sensitive data, check the Password box.

- To allow users to tab to the column but not change the value, check the Display Only box.

- To define a code table to determine which values are displayed to users and which values are stored in the database, check the Use Code Table box and enter display and data values for the code table.

  See "Defining a code table" on page 622.

❖ **To use the Edit edit style:**

1   Select Edit from the Style Type box, if it is not already selected.

2   Select the properties you want.

Date columns and regional settings

Using the Edit edit style, or no edit style, with a date column can cause serious data entry and validation problems if a user's computer is set up to use a nonstandard date style, such as yyyy/dd/mm. For example, if you enter 2001/03/05 in the Retrieval Arguments dialog box for a date column when the mask is yyyy/dd/mm, the date is interpreted as March 5 instead of May 3. To ensure that the order of the day and month is interpreted correctly, use an EditMask edit style.

The DropDownListBox edit style

You can use the DropDownListBox edit style to have columns display as drop-down lists at runtime:

Typically, this edit style is used with code tables, where you can specify display values (which users see) and shorter data values (which are stored in the database).

In the DropDownListBox edit style, the display values of the code table display in the ListBox portion of the DropDownListBox. The data values are the values that are put in the DataWindow buffer (and sent to the database when an Update is issued) when the user selects an item in the ListBox portion of the drop-down list.

In the preceding example, when users see the value Business Services, the corresponding data value could be 200.

❖ **To use the DropDownListBox edit style:**

1 Select DropDownListBox from the Style Type box.

2 Select the appropriate properties.

3 Enter the value you want to have appear in the Display Value box and the corresponding data value in the Data Value box.

At runtime

You can define and modify a code table for a column in a script by using the SetValue method at runtime. To obtain the value of a column at runtime, use the GetValue method. To clear the code table of values, use the ClearValues method.

For more about code tables, see "Defining a code table" on page 622.

The CheckBox edit style

If a column can take only one of two (or perhaps three) values, you might want to display the column as a check box; users can select or clear the check box to specify a value. In the following entry from a DataWindow object, users can simply check or clear a box to indicate whether an employee has health insurance:

Health Insurance: ☒

❖ **To use the CheckBox edit style:**

1 In the Text box, enter the text you want displayed next to the check box.

Using accelerator keys
If the CheckBox has an accelerator key, enter an ampersand (&) before the letter in the text that represents the accelerator key.

---

2 In the Data Value For boxes, enter the values you want put in the DataWindow buffer when the CheckBox is checked (on) or unchecked (off).

If you selected the 3 States box, an optional third state box (other) appears, for the case when the condition is neither on nor off.

What happens

The value you enter in the Text box becomes the display value, and values entered for On, Off, and Other become the data values.

When users check or clear the check box at runtime, PowerBuilder enters the appropriate data value in its buffer. When the Update method is called, PowerBuilder sends the corresponding data values to the database.

Centering check boxes without text

You may find it useful to center check boxes used for columns of information. First make the text control used for the column header and the column control the same size and left aligned. Then you can center the check boxes and the column header.

❖ **To center check boxes without text:**

1   In the Edit property page for the column, make sure the Left Text check box is not selected and that the Text box where you specify associated text is empty.

2   In the General property page, specify centering (Alignment>Center) or specify centering using the StyleBar.

The RadioButtons edit style

If a column can take one of a small number of values, you might want to display the column as radio buttons:
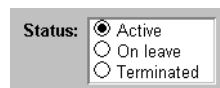
❖ **To use the RadioButtons edit style:**

1   Specify how many radio buttons will display in the Columns Across box.

2   Enter a set of display and data values for each button you want to display.

The display values you enter become the text of the buttons; the data values are put in the DataWindow buffer when the button is clicked.

Using accelerator keys
To use an accelerator key on a radio button, enter an ampersand (&) in the Display Value before the letter that will be the accelerator key.

What happens

Users select values by clicking a radio button. When the Update method is issued, the data values are sent to the database.

The EditMask edit style

Sometimes users need to enter data that has a fixed format. For example, in North America phone numbers have a 3-digit area code, followed by three digits, followed by four digits. You can define an edit mask that specifies the format to make it easier for users to enter values:

Phone: (508) 324-1596
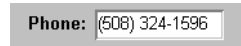
Edit masks consist of special characters that determine what can be entered in the column. They can also contain punctuation characters to aid users.

For example, to make it easier for users to enter phone numbers in the proper format, specify this mask:

```
(###) ###-####
```

At runtime, the punctuation characters display in the box and the cursor jumps over them as the user types:

Phone: ( ) -
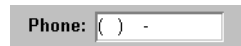
Special characters and keywords

Edit masks use the same special characters as display formats, and there are special considerations for using numeric, string, date, and time masks.

For information, see "Defining display formats" on page 601.

Semicolons invalid in EditMask edit styles
In a display format, you can use semicolons to separate sections in number, date, time, and string formats. You cannot use semicolons in an EditMask edit style.

Keyboard behavior

Note the following about how certain keystrokes behave in edit masks:

- Both Backspace and Shift + Backspace delete the preceding character.

- Delete deletes everything that is selected.

- Non-numeric edit masks treat any characters that do not match the mask pattern as delimiters.

Also, note certain behavior in Date edit masks:

- Entering zero for the day or month causes the next valid date to be entered. For example, if the edit mask is DD/MM/YY, typing `00/11/01` results in `01/11/01`. You can override this behavior in the development environment by adding the following lines to your *PB.INI* file:

```
[Edit Mask Behaviors]
AutocompleteDates=no
```

For deployed applications, the date is completed automatically unless you provide a file called *PB.INI* in the same directory as the executable file that contains these lines. Note that this section must be in a file called *PB.INI*. Adding the section to a different INI file shipped with the application will have no effect.

• The strings 00/00/00 or 00/00/0000 are interpreted as the NULL value for the column.

Using the Mask pop-up menu

Click the button to the right of the Mask box on the Mask property page to display a list that contains complete masks that you can click to add to the mask box, as well as special characters that you can use to construct your own mask. For example, the menu for a Date edit mask contains complete masks such as mm/dd/yy and dd/mmm/yyyy. It also has components such as dd and jjj (for a Julian day). You might use these to construct a mask like dd-mm-yy, typing in the hyphens as separators.

You cannot use a partial mask, such as dd or mmm, in a date edit mask. Any mask that does not include any characters representing the year will be replaced by a mask that does.

Using masks with "as is" characters

You can define a mask that contains "as is" characters that always appear in the control or column. For example, you might define a numeric mask such as Rs0000.00 to represent Indian rupees in a currency column. However, you cannot enter a minus sign to represent negative numbers in a mask that contains "as is" characters.

The preferred method of creating a currency editmask is to use the predefined [currency(7)] - International mask. You can change the number in parentheses, which is the number of characters in the mask including two decimal places. When you use this mask, PowerBuilder uses the currency symbol and format defined in the regional settings section of the Windows control panel. You can enter negative values in a column that uses a currency mask.

Using spin controls

You can define an edit mask as a spin control, a box that contains up and down arrows that users can click to cycle through fixed values. For example, you can set up a code table that provides the valid entries in a column; users simply click an arrow to select an entry. Used this way, a spin control works like a drop-down list that displays one value at a time:
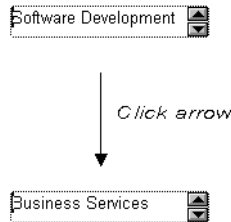
For more about code tables, see "Defining a code table" on page 622.

❖ **To use an EditMask edit style:**

1 Select EditMask in the Style Type box if it is not already selected.

2 Define the mask in the Mask box. Click the special characters in the pop-up menu to use them in the mask. To display the pop-up menu, click the button to the right of the Mask box.

3 Specify other properties for the edit mask.

When you use your EditMask, check its appearance and behavior. If characters do not appear as you expect, you might want to change the font size or the size of the EditMask.

Using a drop-down calendar

You can use a drop-down calendar option on any DataWindow column with an EditMask edit style and a Date, DateTime, or TimeStamp datatype. The DDCalendar EditMask property option allows for separate selections of the calendar month, year, and date. This option can be set in a check box on the Edit page of the DataWindow painter Properties view when a column with the EditMask edit style is selected. It can also be set in code, as in this example for the birth_date column:

```
dw_1.Modify("birth_date.EditMask.DDCalendar='Yes'")
```

If you do not include script for client formatting in a Web DataWindow, the drop-down calendar uses a default edit mask to display the column data based on the client computer's default localization settings. To make sure that dates selected with the drop-down calendar option are displayed with the desired edit mask, specify that the Client Formatting option be included with the static JavaScript generated and deployed for the DataWindow. To conserve bandwidth, JavaScript for client formatting is not included by default. To include this script, you can select the Client Formatting check box on the Web Generation tab of the DataWindow Properties view.

The drop-down calendar option is supported in all Web DataWindow rendering formats (HTML, XHTML, and XML).

The DropDownDataWindow edit style

Sometimes another data source determines which data is valid for a column.

Consider this situation: the Department table includes two columns, Dept_id and Dept_name, to record your company's departments. The Employee table records your employees. The Department column in the Employee table can have any of the values in the Dept_id column in the Department table.

As new departments are added to your company, you want the DataWindow object containing the Employee table to automatically provide the new departments as choices when users enter values in the Department column.

In situations such as these, you can specify the DropDownDataWindow edit style for a column: it is populated from another DataWindow object. When users go to the column, the contents of the DropDownDataWindow display, showing the latest data:

❖ **To use the DropDownDataWindow edit style:**

1 Create a DataWindow object that contains the columns in the detail band whose values you want to use in the column.

You will often choose at least two columns: one column that contains values that the user sees and another column that contains values to be stored in the database. In the example above, you would create a DataWindow object containing the dept_id and dept_name columns in the Department table. Assume this DataWindow object is named d__dddw_dept.

2 For the column getting its data from the DataWindow object, select the DropDownDW edit style.

In the example, you would specify the DropDownDataWindow edit style for the dept_name column:
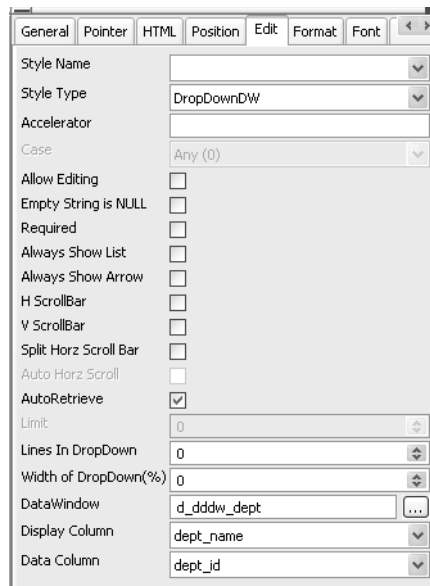
3 Click the browse button next to the DataWindow box and select the DataWindow object that contains the data for the column from the drop-down list (in the example, d__dddw_dept).

4 In the Display Column box, select the column containing the values that will display in the DataWindow object (in the example, dept_name).

5   In the Data Column box, select the column containing the values that will be stored in the database (in the example, dept_id).

6   Specify other properties for the edit style and click OK when done.

What happens

At runtime, when data is retrieved into the DataWindow object, the column whose edit style is DropDownDataWindow will itself be populated as data is retrieved into the DataWindow object serving as the drop-down DataWindow object.

When the user goes to the column and drops it down, the contents of the drop-down DataWindow object display. When the user selects a display value, the corresponding data value is stored in the DataWindow buffer and is stored in the database when an Update is issued.

Limit on size of data value
The data value for a column that uses the DropDownDataWindow edit style is limited to 511 characters.

The InkEdit edit style

The InkEdit edit style is designed for use on a Tablet PC and provides the ability to capture ink input from users of Tablet PCs.

You can specify InkEdit as a style type on the Edit page in the Properties view for columns. When the column gets focus, an InkEdit control displays so that the user can enter text with the stylus or mouse. The text is recognized and displayed, then sent back to the database when the column loses focus.

The InkEdit edit style is fully functional on Tablet PCs. On other computers, it behaves like the Edit edit style.

For more information about ink controls and the Tablet PC, and to download the Tablet PC SDK, go to the Microsoft Tablet PC Web site at http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnanchor/html/tabletpc.asp

Defining a code table

To reduce storage needs, frequently you might want to store short, encoded values in the database, but these encoded values might not be meaningful to users. To make DataWindow objects easy to use, you can define code tables.

Each row in a code table is a pair of corresponding values: a display value and a data value. The display values are those users see at runtime. The data values are those that are saved in the database.

Limit on size of data value
The data value you specify for the Checkbox, DropDownListBox, Edit, EditMask, and RadioButtons edit styles is limited to 255 characters.

How code tables are implemented

You can define a code table as a property of the following column edit styles:

Edit
DropDownListBox
RadioButtons
DropDownDataWindow
EditMask, using spin control

The steps to specify the code table property for each edit style are similar: you begin by defining a new edit style in the Database painter. Once you select an edit style, use the specific procedure that follows to define the code table property.

For how to create an edit style, see "About edit styles" on page 608.

Allowing NULL values
An internal PowerBuilder code, NULL!, indicates null values are allowed. To use this code, specify NULL! as the data value, then specify a display format for nulls for the column.

❖ **To define a code table as a property of the Edit edit style:**

1   Select the Use Code Table check box.

2   Enter the display and data values for the code table.

3   If you want to restrict input in the column to values in the code table, select the Validate check box.

For more information, see "Validating user input" on page 624.

❖   **To define a code table as a property of the DropDownListBox edit style:**

1   Enter the display and data values for the code table.

2   If you want to restrict input in the column to values in the code table, clear the Allow Editing check box.

For more information, see "Validating user input" on page 624.

❖   **To define a code table as a property of the RadioButtons edit style:**

•   Enter the display and data values for the code table.

❖   **To define a code table as a property of the DropDownDataWindow edit style:**

1   Specify the column that provides the display values in the Display Column box.

2   Specify the column that provides the data values in the Data Column box.

3   If you want to restrict input to values in the code table, clear the Allow Editing check box.

❖   **To define a code table as a property of the EditMask edit style:**

1   Select the Spin Control check box.

2   Select the Code Table check box.

3   Enter the display and data values for the code table.

How code tables are processed

When data is retrieved into a DataWindow object column with a code table, processing begins at the top of the data value column. If the data matches a data value, the corresponding display value displays. If there is no match, the actual value displays.

Consider the example in Table 22-9.

Table 22-9: Data values and display values

Display values	Data values
Massachusetts	MA
Massachusetts	ma
ma	MA
Mass	MA
Rhode Island	RI
RI	RI

If the data is MA or ma, the corresponding display value (Massachusetts) displays. If the data is Ma, there is no match, so Ma displays.

Case sensitivity
Code table processing is case sensitive.

If the code table is in a DropDownListBox edit style, and if the column has a code table that contains duplicate display values, then each value displays only once. Therefore, if this code table is defined for a column in a DataWindow object that has a DropDownListBox edit style, Massachusetts and Rhode Island display in the ListBox portion of the DropDownListBox.

Validating user input

When users enter data into a column in a DataWindow object, processing begins at the top of the display value column of the associated code table.

If the data matches a display value, the corresponding data value is put in the internal buffer. For each display value, the first data value is used. Using the sample code table, if the user enters Massachusetts, ma, or Mass, the data value is MA.

You can specify that *only* the values in the code table are acceptable:

• For a column using the Edit edit style, select the Validate check box.

  If you have selected the Validate check box for the Edit edit style, an ItemError event is triggered whenever a user enters a value not in the code table. Otherwise, the entered value is validated using the column's validation rule, if any, and put in the DataWindow buffer.

- For the DropDownListBox and DropDownDataWindow edit styles, clear the Allow Editing check box: users cannot type a value.

  Although users cannot type a value when you clear the Allow Editing check box, they can search for a row in the drop-down list or DataWindow by typing in the initial character for the row display value. The search is case sensitive. For the DropDownDataWindow edit style, the initial character for a search cannot be an asterisk or a question mark. This restriction does not apply to the DropDownListBox edit style.

When the code table processing is complete, the ItemChanged or ItemError event is triggered.

Code table data
The data values in the code table must pass validation for the column and must have the same datatype as the column.

About validation rules

When users enter data in a DataWindow object, you want to be sure the data is valid before using it to update the database. Validation rules provide one way to do this.

You usually define validation rules in the Database painter. To use a validation rule, you associate it with a column in the Database painter or DataWindow object painter.

Another technique
You can also perform data validation through code tables, which are implemented through a column's edit style.

For more information, see "About edit styles" on page 608.

Understanding validation rules

Validation rules are criteria that a DataWindow object uses to validate data entered into a column by users. They are PowerBuilder specific and therefore not enforced by the DBMS.

Validation rules assigned in the Database painter are used by default when you place columns in a DataWindow object. You can override the default rules in the DataWindow object painter.

A validation rule is an expression that evaluates to either TRUE or FALSE. If the expression evaluates to TRUE for an entry into a column, PowerBuilder accepts the entry. If the expression evaluates to FALSE, the entry is not accepted and the ItemError event is triggered. By default, PowerBuilder displays a message box to the user. You can customize the message displayed when a value is rejected.

You can also code the ItemError event to cause different processing to happen.

For more information, see the chapter on using DataWindow objects in the *DataWindow Programmer's Guide*.

At runtime

In scripts, you can use the GetValidate method to obtain the validation rule for a column and the SetValidate method to change the validation rule for a column.

For information about the GetValidate and SetValidate methods, see the online help.

Working with validation rules

You work with validation rules in the Database painter and DataWindow object painter.

What you do in the Database painter

In the Database painter, you can:

•   Create, modify, and delete named validation rules

    The validation rules are stored in the extended attribute system tables. Once you define a validation rule, it can be used by any column of the appropriate datatype in the database.

- Assign validation rules to columns and remove them from columns

    These rules are used by default when you place the column in a DataWindow object in the DataWindow object painter.

What you do in the DataWindow painter

In the DataWindow painter, you can:

- Accept the default validation rule assigned to a column in the Database painter

- Create an ad hoc, unnamed rule to use with one specific column

Validation rules and the extended attribute system tables

Once you have placed a column that has a validation rule from the extended attribute system tables in a DataWindow object, there is no longer any link to the named rule in the extended attribute system tables.

If the definition of the validation rule later changes in the extended attribute system tables, the rule for the column in a DataWindow object will not change.

Defining validation rules

Typically, you define validation rules in the Database painter, because validation rules are properties of the data itself. Once defined in the Database painter, the rules are used by default each time the column is placed in a DataWindow object. You can also define a validation rule in the DataWindow object painter that overrides the rule defined in the Database painter.

Defining a validation rule in the Database painter

This section describes the ways you can manipulate validation rules in the Database painter.

❖ **To create a new validation rule**

1   In the Database painter, select Object>Insert>Validation Rule from the menu bar.

    The Validation Rule view displays in the Properties view.

2   Assign a name to the rule, select the datatype of the columns to which it applies, and customize the error message (if desired).

    For information, see "Customizing the error message" on page 630.

3   Click the Definition tab and define the expression for the rule.

For information, see "Defining the expression" on page 629.

You can use this rule with any column of the appropriate datatype in the database.

❖ **To modify a validation rule:**

1   In the Database painter, open the Extended Attributes view.

2   In the Extended Attributes view, open the list of validation rules.

3   Position the pointer on the validation rule you want to modify, display the pop-up menu, and select Properties.

4   In the Validation Rule view, modify the validation rule as desired.

For information, see "Defining the expression" on page 629 and "Customizing the error message" on page 630.

❖ **To associate a validation rule with a column in the Database painter:**

1   In the Database painter (Objects view), position the pointer on the column, select Properties from the column's pop-up menu, and select the Validation tab.

2   Select a validation rule from the Validation Rule drop-down list.

The column now has the selected validation rule associated with it in the extended attribute system tables. Whenever you use this column in a DataWindow object, it will use this validation rule unless you override it in the DataWindow painter.

❖ **To remove a validation rule from a column in the Database painter:**

1   In the Database painter (Objects view), position the pointer on the column, select Properties from its pop-up menu, and select the Validation tab in the Properties view.

2   Select (None) from the list in the Validation Rule drop-down list.

The validation rule is no longer associated with the column.

Defining the expression

A validation rule is a boolean expression. PowerBuilder applies the boolean expression to an entered value. If the expression returns TRUE, the value is accepted. Otherwise, the value is not accepted and an ItemError event is triggered.

What expressions can contain

You can use any valid DataWindow expression in validation rules.

Validation rules can include most DataWindow expression functions. A DataWindow object that will be used in PowerBuilder can also include user-defined functions. DataWindow expression functions are displayed in the Functions list and can be pasted into the definition.

For information about these functions, see the *DataWindow Reference.*

Use the notation @*placeholder* (where *placeholder* is any group of characters) to indicate the current column in the rule. When you define a validation rule in the Database painter, PowerBuilder stores it in the extended attribute system tables with the placeholder name. At runtime, PowerBuilder substitutes the value of the column for *placeholder*.

Pasting the placeholder

The @col can be easily used as the placeholder. A button in the Paste area is labeled with @col. You can click the button to paste the @col into the validation rule.

An example

For example, to make sure that both Age and Salary are greater than zero using a single validation rule, define the validation rule as follows:

```
@col > 0
```

Then associate the validation rule with both the Age and Salary columns. At runtime, PowerBuilder substitutes the appropriate values for the column data when the rule is applied.

Using match values for character columns

If you are defining the validation rule for a character column, you can use the Match button on the Definition page of the Validation Rule view. This button lets you define a match pattern for matching the contents of a column to a specified text pattern (for example, ^[0-9]+$ for all numbers and ^[A-Za-z]+$ for all letters).

❖ **To specify a match pattern for character columns:**

1 Click the Match button on the Definition page of the Validation Rule view.

The Match Pattern dialog box displays.

2 Enter the text pattern you want to match the column to, or select a displayed pattern.

3 (Optional) Enter a test value and click the Test button to test the pattern.

4 Click OK when you are satisfied that the pattern is correct.

For more on the Match function and text patterns, see the *DataWindowReference.*

Customizing the error message

When you define a validation rule, PowerBuilder automatically creates the error message that displays by default when users enter an invalid value:

    'Item ~'' + @*Col* + '~' does not pass validation test.'

You can edit the string expression to create a custom error message.

Different syntax in the DataWindow object painter

If you are working in the DataWindow object painter, you can enter a string expression for the message, but you do not use the @ sign for placeholders. For example, this is the default message:

    'Item ~'' + *ColumnName* + '~' does not pass validation test.'

A validation rule for the Salary column in the Employee table might have the following custom error message associated with it:

    Please enter a salary greater than $10,000.'

If users enter a salary less than or equal to $10,000, the custom error message displays.

Specifying initial values

As part of defining a validation rule, you can supply an initial value for a column.

❖ **To specify an initial value for a column in the Database painter:**

1 Select Properties from the column's pop-up menu and select the Validation tab.

2 Specify a value in the Initial Value box.

Defining a validation rule in the DataWindow painter

Validation rules you assign to a column in the Database painter are used by default when you place the column in a DataWindow object. You can override the validation rule in the DataWindow object painter by defining an ad hoc rule for one specific column.

❖ **To specify a validation rule for a column in the DataWindow painter:**

1 In the DataWindow painter, select View>Column Specifications from the menu bar.

The Column Specification view displays.

2 Create or modify the validation expression. To display the Modify Expression dialog box, display the pop-up menu for the box in which you want to enter a Validation Expression and select Expression. Follow the directions in "Specifying the expression" next.

3 (Optional) Enter a string or string expression to customize the validation error message.

For more information, see "Customizing the error message" on page 630.

4 (Optional) Enter an initial value.

Used for current columns only
If you create a validation rule here, it is used only for the current column and is not saved in the extended attribute system tables.

Specifying the expression

Since a user might have just entered a value in the column, validation rules refer to the current data value, which you can obtain through the GetText DataWindow expression function.

Using GetText ensures that the most recent data entered in the current column is evaluated.

PowerBuilder does the conversion for you
If you have associated a validation rule for a column in the Database painter, PowerBuilder automatically converts the syntax to use GetText when you place the column in a DataWindow object.

GetText returns a string. Be sure to use a data conversion function (such as Integer or Real) if you want to compare the entered value with a datatype other than string.

For more on the GetText function and text patterns, see the *DataWindowReference*.

Referring to other columns

You can refer to the values in other columns by specifying their names in the validation rule. You can paste the column names in the rule using the Columns box.

Examples

Here are some examples of validation rules.

Example 1   To check that the data entered in the current column is a positive integer, use this validation rule:

```
Integer(GetText( )) > 0
```

Example 2   If the current column contains the discounted price and the column named Full_Price contains the full price, you could use the following validation rule to evaluate the contents of the column using the Full_Price column:

```
Match(GetText( ),"^[0-9]+$") AND
Real(GetText( )) < Full_Price
```

To pass the validation rule, the data must be all digits (must match the text pattern ^[0-9]+$) and must be less than the amount in the Full_Price column.

Notice that to compare the numeric value in the column with the numeric value in the Full_Price column, the Real function was used to convert the text to a number.

Example 3   In your company, a product price and a sales commission are related in the following way:

- If the price is greater than or equal to $1000, the commission is between 10 percent and 20 percent

- If the price is less than $1000, the commission is between 4 percent and 9 percent

The Sales table has two columns, Price and Commission. The validation rule for the Commission column is:

```
(Number(GetText( )) >= If(price >= 1000, .10, .04))
AND
(Number(GetText( )) <= If(price >= 1000, .20, .09))
```

A customized error message for the Commission column is:

```
"Price is " + if(price >= 1000,
"greater than or equal to","less than") +
" 1000. Commission must be between " +
If(price >= 1000,".10", ".04") + " and " +
If(price >= 1000, ".20.", ".09.")
```

How to maintain extended attributes

PowerBuilder provides facilities you can use to create, modify, and delete display formats, edit styles, and validation rules independently of their association with columns. The following procedure summarizes how you do this.

❖ **To maintain display formats, edit styles, and validation rules:**

1   Open the Database painter.

2   Select View>Extended Attributes.

The Extended Attributes view displays listing all the entities in the extended attribute system tables.

3 Do one of the following:

- To create a new entity, display the pop-up menu for the type you want to add, then select New.

- To modify an entity, display its pop-up menu, then select Properties.

- To delete an entity, display its pop-up menu, then select Delete.

Caution
If you delete a display format, edit style, or validation rule, it is removed from the extended attribute system tables. Columns in the database are no longer associated with the entity.

Filtering, Sorting, and Grouping Rows

Filtering rows

You can use WHERE and HAVING clauses and retrieval arguments in the SQL SELECT statement for the DataWindow object to limit the data that is retrieved from the database. This reduces retrieval time and space requirements at runtime.

However, you may want to further limit the data that displays in the DataWindow object. For example, you might want to:

- Retrieve many rows and initially display only a subset (perhaps allowing the user to specify a different subset of rows to display at runtime)

- Limit the data that is displayed using DataWindow expression functions (such as If) that are not valid in the SELECT statement

Using filters

In the DataWindow painter, you can define filters to limit the rows that display at runtime. Filters can use most DataWindow expression functions or user-defined functions.

Filters do not affect which rows are retrieved
A filter operates against the retrieved data. It does not re-execute the SELECT statement.

Defining a filter

❖ **To define a filter:**

1 In the DataWindow painter, select Rows>Filter from the menu bar.

The Specify Filter dialog box displays:
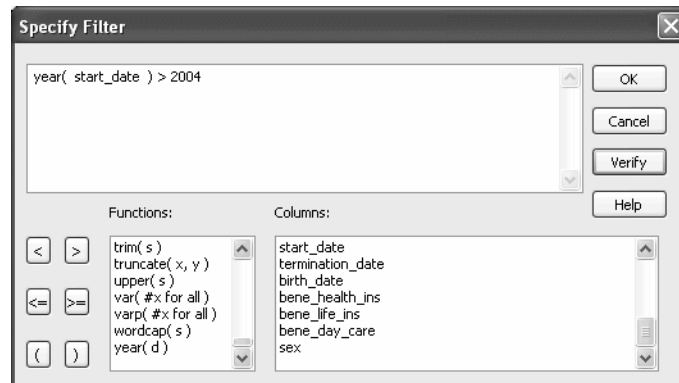
2 In the Specify Filter dialog box, enter a boolean expression that PowerBuilder will test against each retrieved row.

If the expression evaluates to TRUE, the row is displayed. You can specify any valid expression in a filter. Filters can use any non-object-level PowerScript function, including user-defined functions. You can paste commonly used functions, names of columns, computed fields, retrieval arguments, and operators into the filter.

International considerations
For applications to run the same in any country, filter expressions require U.S. notation for numbers. That is, a comma always represents the thousands delimiter and a period always represents the decimal place when you specify expressions in the development environment.

For information about expressions for filters, see the *DataWindow Reference*.

3   (Optional) Click Verify to make sure the expression is valid.

4   Click OK.

Only rows meeting the filter criteria are displayed in the Preview view.

Filtered rows and updates
Modifications of filtered rows are applied to the database when you issue an update request.

Removing a filter

❖   **To remove a filter:**

1   Select Rows>Filter from the menu bar.

2   Delete the filter expression from the Specify Filter dialog box, then click OK.

Examples of filters   Assume that a DataWindow object retrieves employee rows and three of the columns are Salary, Status, and Emp_Lname. Table 23-1shows some examples of filters you might use.

Table 23-1: Sample filters

To display these rows	Use this filter
Employees with salaries over $50,000	`Salary > 50000`
Active employees	`Status = 'A'`
Active employees with salaries over $50,000	`Salary > 50000 AND Status = 'A'`
Employees whose last names begin with H	`left(Emp_Lname, 1) = 'H'`

Setting filters in a script   You can use the SetFilter and Filter methods in a script to dynamically modify a filter that was set in the DataWindow painter. For information about SetFilter and Filter, see the online help.

Sorting rows

You can use an ORDER BY clause in the SQL SELECT statement for the DataWindow object to sort the data that is retrieved from the database. If you do this, the DBMS itself does the sorting and the rows are brought into PowerBuilder already sorted.

However, you might want to sort the rows after they are retrieved. For example, you might want to:

- Offload the processing from the DBMS

- Sort on an expression, which might not be allowed in the SELECT statement but is allowed in PowerBuilder

❖ **To sort the rows:**

1 Select Rows>Sort from the menu bar.
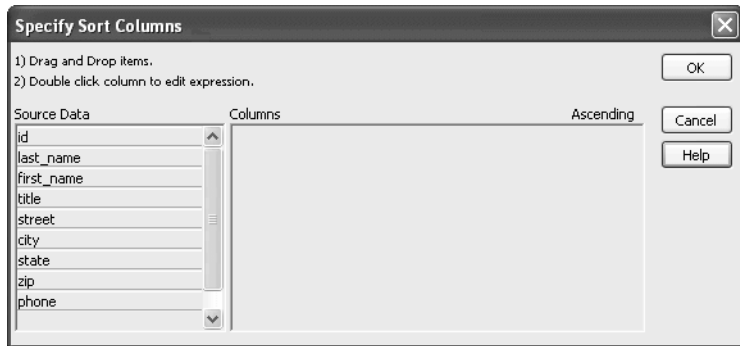
2 Drag to the Columns box the columns on which you want to sort the rows, and specify whether you want to sort in ascending or descending order.

The order of the columns determines the precedence of the sort. To reorder the columns, drag them up or down in the list. To delete a column from the sort columns list, drag the column outside the dialog box.

3 You can also specify expressions to sort on: for example, if you have two columns, Revenues and Expenses, you can sort on the expression *Revenues – Expenses*.

To specify an expression to sort on, double-click a column name in the Columns box, modify the expression in the Modify Expression dialog box, and click OK.

You return to the Specify Sort Columns dialog box with the expression displayed.

If you change your mind
You can remove a column or expression from the sorting specification by simply dragging it and releasing it outside the Columns box.

4   Click OK when you have specified all the sort columns and expressions.

Suppressing repeating values

When you sort on a column, there might be several rows with the same value in one column. You can choose to suppress the repeating values in that column.

When you suppress a repeating value, the value displays at the start of each new page and, if you are using groups, each time a value changes in a higher group.

For example, if you have sorted employees by department ID, you can suppress all but the first occurrence of each department ID in the DataWindow object:

Dept	ID	Name	Salary
300	390	Davidson , Jo Ann	$57,090
	586	Coleman , James	$42,300
	757	Higgins , Denis	$43,700
	879	Coe , Kristen	$36,500
	1293	Shea , Mary Anne	$138,948
	1336	Bigelow , Janet	$31,200
	1390	Litton , Jennifer	$58,930
	1483	Letiecq , John	$75,400
400	184	Espinoza , Melissa	$36,490
	207	Francis , Jane	$53,870
	318	Crow , John	$41,701
	409	Weaver , Bruce	$46,550
	591	Barletta , Irene	$45,450
	888	Charlton , Doug	$28,300
	992	Butterfield , Joyce	$34,011

❖ **To suppress repeating values:**

1 Select Rows>Suppress Repeating Values from the menu bar.

The Specify Repeating Value Suppression List dialog box displays:

2 Drag the columns whose repeated values you want to suppress from the Source Data box to the Suppression List box, and click OK.

If you change your mind
You can remove a column from the suppression list simply by dragging it and releasing it outside the Suppression List box.

Grouping rows

You can group related rows together and, optionally, calculate statistics for each group separately. For example, you might want to group employee information by department and get total salaries for each department.

How groups are defined

Each group is defined by one or more DataWindow object columns. Each time the value in a grouping column changes, a break occurs and a new section begins.

For each group, you can:

• Display the rows in each section

• Specify the information you want to display at the beginning and end of each section

• Specify page breaks after each break in the data

• Reset the page number after each break

Grouping example

The following DataWindow object retrieves employee information. It has one group defined, Dept_ID, so it groups rows into sections according to the value in the Dept_ID column. In addition, it displays:

- Department ID before the first row for that department

- Totals and averages for salary and salary plus benefits (a computed column) for each department

- Grand totals for the company at the end

The following screenshot shows the DataWindow object.

Total Compensation Report **Salary Plus Benefits**					Value of Health Ins. - $4000 Value of Life Ins. - .543% of salary Value of Day Care - $5,200				Page 4 of 4 2/27/2004
Dept. ID	Employee ID	Employee First Name	Employee Last Name	Salary	Health Ins.	Life Ins.	Day Care		Salary Plus Benefits
400									
	1191	Matthew	Bucceri	$45,900.00	☑	☑	☐		$55,449.00
	1507	Ruth	Wetherby	$35,745.00	☑	☑	☐		$45,239.00
	1576	Scott	Evans	$68,940.00	☑	☑	☐		$78,614.00
	1607	Mark	Morris	$61,300.00	☑	☑	☐		$70,932.00
	1643	Elizabeth	Lambert	$29,384.00	☑	☑	☐		$38,843.00
	1684	Janet	Hildebrand	$45,829.00	☑	☑	☐		$55,377.00
	1740	Robert	Nielsen	$34,889.00	☑	☑	☐		$44,378.00
	1751	Alex	Ahmed	$34,992.00	☑	☑	☐		$44,482.00
			Total:	$698,250.75			Total:		$850,842.25
			Average:	$43,640.67			Average:		$53,177.17
500									
	191	Jeannette	Bertrand	$32,780.00	☑	☑	☑		$42,257.00
	703	Jose	Martinez	$61,050.88	☑	☑	☐		$70,681.88
	750	Jane	Braun	$37,730.00	☑	☑	☐		$47,234.00
	868	Felicia	Kuo	$28,200.00	☑	☑	☐		$37,653.00
	921	Charles	Crowley	$41,700.00	☑	☑	☐		$51,226.00
	1013	Joseph	Barker	$27,290.00	☐	☑	☐		$36,738.00
	1570	Anthony	Rebeiro	$34,576.00	☑	☑	☐		$44,063.00
	1615	Sheila	Romero	$27,500.00	☑	☑	☑		$36,949.00
	1658	Michael	Lynch	$24,903.00	☑	☑	☐		$34,338.00
			Total:	$315,729.88			Total:		$401,144.29
			Average:	$35,081.10			Average:		$44,571.10
			Grand Total:	$3,761,106.82			Grand Total:		$4,479,029.63
			Average:	$50,148.09			Average:		$59,719.93

How to do it

You can create a grouped DataWindow object in three ways:

- Use the Group presentation style to create a grouped DataWindow object from scratch ("Using the Group presentation style" next).

- Take an existing tabular DataWindow object and define grouping ("Defining groups in an existing DataWindow object" on page 646).

- Use the TreeView presentation style (Chapter 28, "Working with TreeViews").

Making the DataWindow control large enough

If a DataWindow object has grouped rows, each page contains all group headers (including zero-height headers) at the top of the page. Your DataWindow control must be large enough to accommodate all the group headers that display on each page of the report.

The last row of a group displays on the same page as that row's group trailer and each applicable higher-level group trailer. If the DataWindow object has a summary band, it displays on the same page as the last row of the report. If the control is not large enough, you might see anomalies when scrolling through the DataWindow object, particularly in the last row of the report, which needs room to display the report's header band, all group headers, all group trailers, the summary band, and the footer band.

If you cannot increase the height of the DataWindow control so that it has room for all the headers and trailers, you can change the design of the DataWindow object so that they require less space.

Scrolling through a grouped DataWindow

When you scroll through a grouped DataWindow object, you might see the group header repeated where you do not expect it. This is because the data is paginated in a fixed layout based on the size of the DataWindow control. You can scroll to a point that shows the bottom half of one page and the top of the next. When you use the arrow keys to page through the data, you scroll one row at a time.

Using the Group presentation style

One of the DataWindow object presentation styles, Group, is a shortcut to creating a grouped DataWindow object. It generates a tabular DataWindow object that has one group level and some other grouping properties defined. You can then further customize the DataWindow object.

❖ **To create a basic grouped DataWindow object using the Group presentation style:**

1   Select File>New from the menu bar.

The New dialog box displays.

2   Choose the DataWindow tab page and the Group presentation style, and click OK.

3   Choose a data source and define the data.

You are prompted to define the grouping column(s).

4   Drag the column(s) you want to group on from the Source Data box to the Columns box.

Multiple columns and multiple group levels
You can specify more than one column, but all columns apply to group level one. You can define one group level at this point. Later you can define additional group levels.

In the following example, grouping will be by department, as specified by the dept_id column:
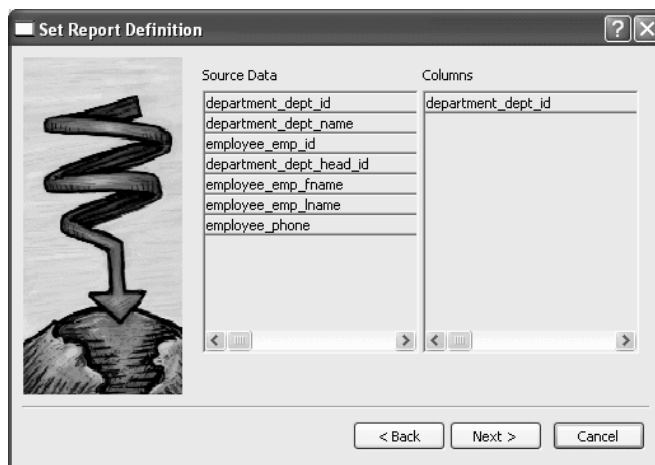
If you want to use an expression, you can define it when you have completed the wizard. See "Using an expression for a group" on page 645.

5   Click Next.

PowerBuilder suggests a header based on your data source. For example, if your data comes from the Employee table, PowerBuilder uses the name Employee in the suggested header.

6   Specify the Page Header text.

7   If you want a page break each time a grouping value changes, select the New Page On Group Break box.

8   If you want page numbering to restart at 1 each time a grouping value changes, select the Reset Page Number On Group Break box *and* the New Page On Group Break box.

9   Click Next.

10   Select Color and Border settings and click Next.

11   Review your specification and click Finish.

The DataWindow object displays with the basic grouping properties set.

This is an example of a Group style DataWindow object:
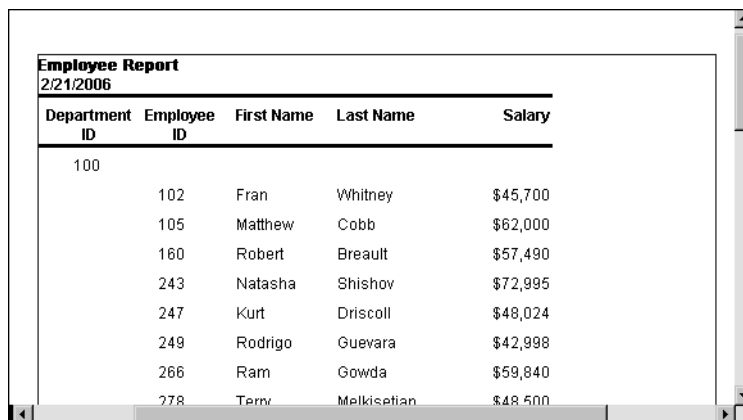
What PowerBuilder does

As a result of your specifications, PowerBuilder generates a tabular DataWindow object and:

• Creates group header and trailer bands

• Places the column you chose as the grouping column in the group header band

• Sorts the rows by the grouping column

- Places the page header and the date (as a computed field) in the header band

- Places the page number and page count (as computed fields) in the footer band

- Creates sum-computed fields for all numeric columns (the fields are placed in the group trailer and summary bands)

Here is the preceding DataWindow object in the Preview view:

Using an expression for a group

If you want to use an expression for one or more column names in a group, you can enter an expression as the Group Definition on the General page in the Properties view after you have finished using the Group wizard.

❖ **To use an expression for a group:**

1 Open the Properties view and select the group header band in the Design view.

2 Double-click the ellipsis button next to the Group Definition box on the General page in the Properties view to open the Specify Group Columns dialog box.

3 In the Columns box, double-click the column that you want to use in an expression.

The Modify Expression dialog box opens. You can specify more than one grouping item expression for a group. A break occurs whenever the value concatenated from each column/expression changes.

What you can do

You can use any of the techniques available in a tabular DataWindow object to modify and enhance the grouped DataWindow object, such as moving controls, specifying display formats, and so on. In particular, see "Defining groups in an existing DataWindow object" next to learn more about the bands in a grouped DataWindow object and how to add features especially suited for grouped DataWindow objects (for example, add a second group level, define additional summary statistics, and so on).

DataWindow Object is not updatable by default
When you generate a DataWindow object using the Group presentation style, PowerBuilder makes it not updatable by default. If you want to be able to update the database through the grouped DataWindow object, you must modify its update characteristics. For more information, see Chapter 21, "Controlling Updates in DataWindow Objects."

Defining groups in an existing DataWindow object

Instead of using the Group presentation style to create a grouped DataWindow object from scratch, you can take an existing tabular DataWindow object and define groups in it.

❖ **To add grouping to an existing DataWindow object:**

1 Start with a tabular DataWindow object that retrieves all the columns you need.

2 Specify the grouping columns.

3 Sort the rows.

4 (Optional) Rearrange the DataWindow object.

5 (Optional) Add summary statistics.

6 (Optional) Sort the groups.

Steps 2 through 6 are described next.

Specifying the grouping columns

❖ **To specify the grouping columns:**

1   In the DataWindow painter, Select Rows>Create Group from the menu bar.

The Specify Group Columns dialog box displays.

2   Specify the group columns, as described in "Using the Group presentation style" on page 642.

3   Set the Reset Page Count and New Page on Group Break properties on the General page in the Properties view.

Creating subgroups

After defining your first group, you can define subgroups, which are groups within the group you just defined.

❖ **To define subgroups:**

1   Select Rows>Create Group from the menu bar and specify the column/expression for the subgroup.

2   Repeat step 1 to define additional subgroups if you want.

You can specify as many levels of grouping as you need.

How groups are identified

PowerBuilder assigns each group a number (or level) when you create the group. The first group you specify becomes group 1, the primary group. The second group becomes group 2, a subgroup within group 1, and so on.

For example, suppose you define two groups. The first group uses the dept_id column and the second group uses the status column.

The rows are grouped first by department (group 1). Within department, rows are grouped by status (group 2). If you specify page breaks for the groups, a page break will occur when any of these values changes.

You use the group's number to identify it when defining summary statistics for the group. This is described in "Adding summary statistics" on page 650.

Sorting the rows

PowerBuilder does not sort the data when it creates a group. Therefore, if the data source is not sorted, you must sort the data by the same columns (or expressions) specified for the groups.

For example, if you are grouping by dept_id then status, select Rows>Sort from the menu bar and specify dept_id and then status as sorting columns:
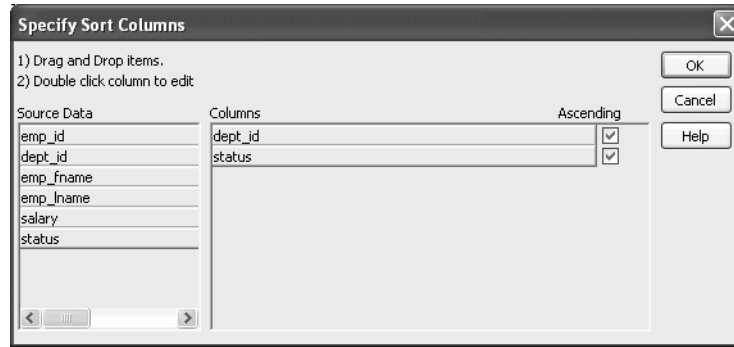
You can also sort on additional rows. For example, if you want to sort by employee ID within each group, specify emp_id as the third sorting column.

For more information about sorting, see "Sorting rows" on page 638.

Rearranging the DataWindow object

When you create a group, PowerBuilder creates two new bands for each group:

- A group header band
- A group trailer band

The bar identifying the band contains:

- The number of the group
- The name of the band
- The name of each column that defines the group
- An arrow pointing to the band
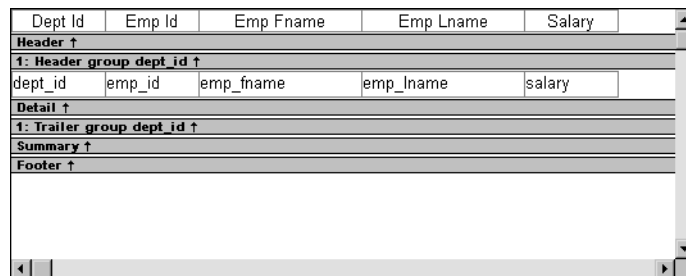
You can include any control in the DataWindow object (such as columns, text, and computed fields) in the header and trailer bands of a group.

Using the group header band

The contents of the group header band display at the top of each page and after each break in the data.

Typically, you use this band to identify each group. You might move the grouping column from the detail band to the group header band, since it now serves to identify one group rather than each row.

For example, if you group the rows by department and include the department in the group header, the department will display before the first line of data each time the department changes.

At runtime, you see this:

Emp Id	Emp Fname	Emp Lname	Salary
Dept Id			
100			
1090	Susan	Smith	51411.0
582	Peter	Samuels	37400.0
529	Dorothy	Sullivan	67890.0
479	Linda	Siperstein	39875.5
266	Ram	Gowda	59840.0
604	Albert	Wang	68400.0
453	Andrew	Rabkin	64500.0
445	Kim	Lull	87900.0

Suppressing group headers

If you do not want a group header to display at the top of each page when you print or display a report, select the Suppress Group Header check box on the General property page for the header. If none of the headers are suppressed, they all display at the top of each page. When a page break coincides with a group break, the group header and any group headers that follow it display even if the Suppress Group Header property is set, but higher level headers are suppressed if the property is set for those headers.

For example, suppose a report has three groups: division, sales region, and sales manager. If all three group headers are suppressed, and a sales region group break coincides with a page break, the division header is suppressed but the sales region and sales manager headers display.

Using the group trailer band

The contents of the group trailer display after the last row for each value that causes a break.

In the group trailer band, you specify the information you want displayed after the last line of identical data for each value in the group. Typically, you include summary statistics here, as described next.

Adding summary statistics

One of the advantages of creating a grouped DataWindow object is that you can have PowerBuilder calculate statistics for each group. To do that, you place computed fields that reference the group. Typically, you place these computed fields in the group's trailer band.

❖ **To add a summary statistic:**

1 Select Insert>Control>Computed Field from the menu bar.

2 Click in the Design view where you want the statistic.

The Modify Expression dialog box displays.

3 Specify the expression that defines the computed field (see below).

4 Click OK.

A shortcut to sum values
If you want to sum a numeric column, select the column in Design view and click the Sum button in the Controls drop-down toolbar. PowerBuilder automatically places a computed field in the appropriate band.

Specifying the expression

Typically, you use aggregate and other functions in your summary statistic. PowerBuilder lists functions you can use in the Functions box in the Modify Expression dialog box. When you are defining a computed field in a group header or trailer band, PowerBuilder automatically lists forms of the functions that reference the group:
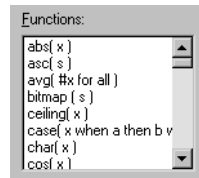
You can paste these templates into the expression, then replace the #x that is pasted in as the function argument with the appropriate column or expression.

For example, to count the employees in each department (group 1), specify this expression in the group trailer band:

```
Count( Emp_Id for group 1 )
```

To get the average salary of employees in a department, specify:

```
Avg( Salary for group 1 )
```

To get the total salary of employees in a department, specify:

```
Sum( Salary for group 1 )
```


At runtime, you see this:

Sorting the groups

You can sort the groups in a DataWindow object. For example, in a DataWindow object showing employee information grouped by department, you might want to sort the departments (the groups) by total salary.

Typically, this involves aggregate functions, as described in "Adding summary statistics" on page 650. In the department salary example, you would sort the groups using the aggregate function Sum to calculate total salary in each department.

❖ **To sort the groups:**

1 Place the mouse pointer on the group header bar (not inside the band) until the pointer becomes a double-headed arrow.

2 Click.

The General property page for the group displays in the Properties view.

3 Click the button next to the Group Sort box.
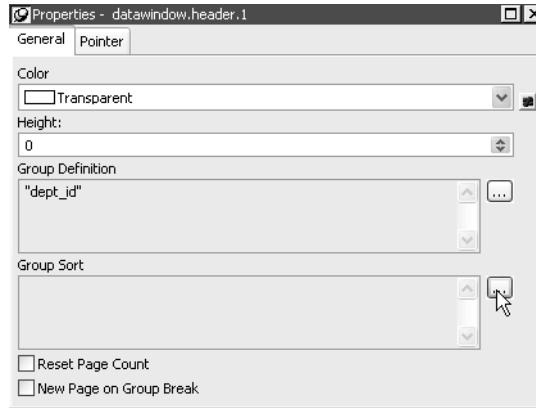
The Specify Sort Columns dialog box displays.

4 Drag the column you want to sort the groups by from the Source Data box into the Columns box.

If you chose a numeric column, PowerBuilder uses the Sum function in the expression; if you chose a non-numeric column, PowerBuilder uses the Count function.

For example, if you chose the Salary column, PowerBuilder specifies that the groups will be sorted by the expression sum(salary for group 1):
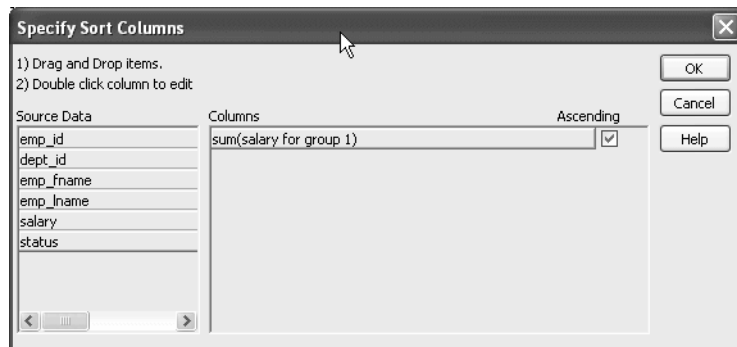
5   Select ascending or descending sort as appropriate.

6   If you want to modify the expression to sort on, double-click the column in the Columns box.

The Modify Expression dialog box displays.

7   Specify the expression to sort on.

For example, to sort the department group (the first group level) on average salary, specify `avg(salary for group 1)`.

8   Click OK.

You return to the Specify Sort Columns dialog box with the expression displayed.

9   Click OK again.

At runtime, the groups will be sorted on the expression you specified.

Highlighting Information in DataWindow Objects

About this chapter

This chapter describes how you modify the way information displays in DataWindow objects and reports based on the conditions you specify. The conditions are usually related to data values, which are not available until runtime.

Contents

Topic	Page
Highlighting information	655
Modifying properties conditionally at runtime	659
Supplying property values	665
Specifying colors	684

Highlighting information

Every control in a DataWindow object has a set of properties that determines what the control looks like and where it is located. For example, the values in a column of data display in a particular font and color, in a particular location, with or without a border, and so on.

Modifying properties when designing

You define the appearance and behavior of controls in DataWindow objects in the DataWindow painter. As you do that, you are specifying the controls' properties. For example, when you place a border around a column, you are setting that column's Border property.

In most cases, the appearance and behavior of controls is fixed; you do not want them to change at runtime. When you make headings bold when designing them, you want them to be bold at all times.

In the following DataWindow object, the Salary Plus Benefits column has a Shadow box border around every data value in the column. To display the border, you set the border property for the column:

Modifying properties at runtime

In some cases, however, you might want some properties of controls in DataWindow objects to be driven by the data, which is not known when you are defining the DataWindow object in the painter. For these situations you can define property conditional expressions, which are expressions that are evaluated at runtime.

You can use these expressions to conditionally and dynamically modify the appearance and behavior of your DataWindow object at runtime. The results of the expressions set the values of properties of controls in the DataWindow object.

In the following DataWindow object, the Salary Plus Benefits column has a Shadow box border highlighting each data value that is greater than $60,000:

To control the display of the border, you define a property conditional expression for the column's Border property. When users run the DataWindow object, PowerBuilder changes the border of individual data values based on the condition (value greater than $60,000).

Defining an expression

The following illustration shows the Salary_Plus_Benefits column selected in the Design view. To the right of the Design view, the Properties view shows properties for the column, including the Border property. Next to the Border property is a button for accessing the dialog box where you enter the expression. The button displays an equals sign with a slash through it when no expression has been entered, and an equals sign without a slash when it has.
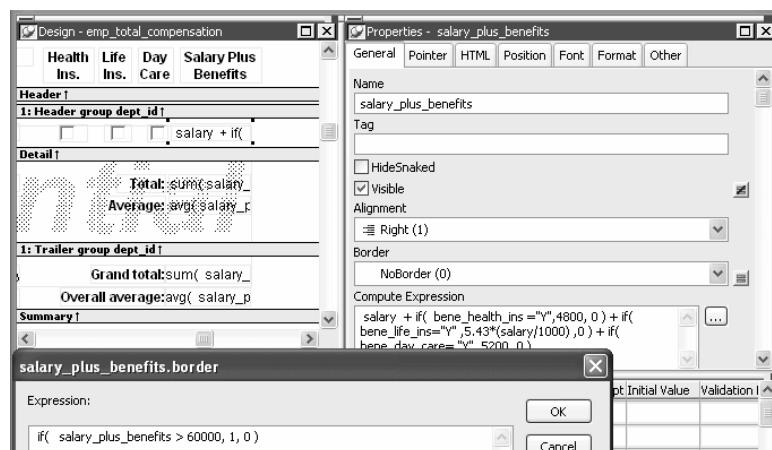
In this example the Border property is set to NoBorder in the Properties view. However, the expression defined for the property overrides that setting at runtime.

A closer look at the expression

The expression you enter almost always begins with If. Then you specify three things: the condition, what happens if it is true, and what happens if it is false. Parentheses surround the three things and commas separate them:

If(*expression, true, false*)

The following expression is used in the example. Because the expression is for the Border property, the values for true and false indicate particular borders. The value 1 means Shadow box border and the value 0 means no border:

```
If(salary_plus_benefits > 60000, 1, 0)
```

When users run the DataWindow object, PowerBuilder checks the value in the computed column called salary_plus_benefits to see if it is greater than 60,000. If it is (true), PowerBuilder displays the value with the Shadow box border. If not (false), PowerBuilder displays the value with no border.

About specifying
properties

Usually you specify a number to indicate what you want for a particular property. For example, the following list shows all of the borders you can specify and the numbers you use. If you want the border property to be Shadow box, you specify 1 in the If statement, for either true or false.

0—None
1—Shadow box
2—Box
3—Resize
4—Underline
5—3D Lowered
6—3D Raised

In the Properties view, the list of choices for setting a property includes the values that correspond to choices in parentheses. This makes it easier to define an expression for a property; you do not need to look up the values. For example, if you want to specify the ResizeBorder in your expression, you use the number 3, as shown in the drop-down list.
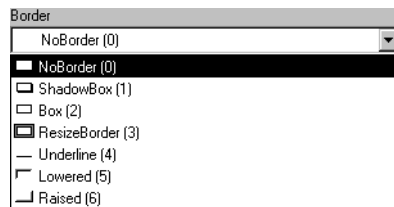
For details on the values of properties that can be set using expressions, see "Supplying property values" on page 665.

For complete information about what the valid values are for all properties associated with a DataWindow object, see the discussion of DataWindow object properties in the *DataWindow Reference* or online Help.

About modifying
properties
programmatically

You can also programmatically modify the properties of controls in a DataWindow object at runtime. For more information, see the *DataWindow Reference* and the *DataWindow Programmer's Guide*.

Modifying properties conditionally at runtime

"Modifying properties at runtime" on page 656 described how you can use conditional expressions that are evaluated at runtime to highlight information in a DataWindow object. This section presents a procedure for modifying properties at runtime and some examples.

❖ **To modify properties conditionally at runtime:**

1 Position the pointer on the control, band, or DataWindow object background whose properties you want to modify at runtime.

2 Select Properties from the pop-up menu, then select the page that contains the property you want to modify at runtime.

3 Click the button next to the property you want to change.

4 Scroll the list of functions in the Functions box until you see the IF function, and then select it:
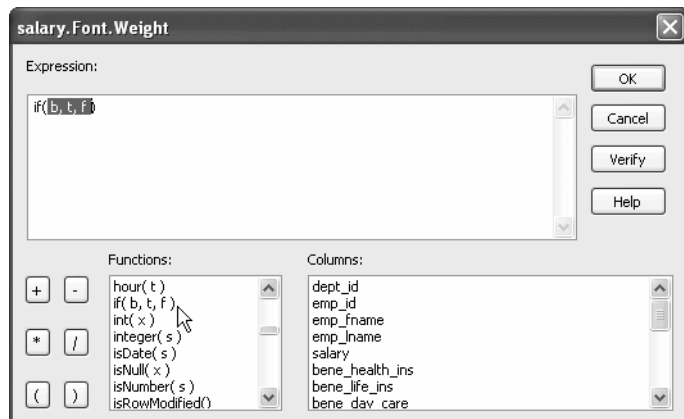
5 Replace the *b* (boolean) with your condition (for example, salary>40000).

You can select columns and functions and use the buttons to add the symbols shown on them.

6 Replace the *t* (true) with the value to use for the property if the condition is true.

Values to use for properties are usually numbers. They are different for each property. For more information about property values that can be set on the Expressions page, see "Supplying property values" on page 665.

For complete information about what the valid values are for all properties of controls in the DataWindow object, see the discussion of DataWindow object properties in the *DataWindow Reference* or online Help.

7   Replace the *f* (false) with the value to use for the property if the condition is false.

8   Click OK.

For examples, see "Example 1: creating a gray bar effect" next, "Example 2: rotating controls" on page 661, "Example 3: highlighting rows of data" on page 662, and "Example 4: changing the size and location of controls" on page 664.

Example 1: creating a gray bar effect

The following DataWindow object shows alternate rows with a light gray bar. The gray bars make it easier to track data values across the row:

To create the gray bar effect:

1   Add a rectangle control to the detail band and size it so that it surrounds the controls you want highlighted.

     To make sure that you have selected the detail band, select the Position tab in the Properties view and select Band from the Layer drop-down list.

2   To make it easier to see what you are doing in the Design view, select the General tab and set the Brush Color to White and the Pen Color to Black. A narrow black line forms a boundary around the rectangle.

3   Select Send to Back from the rectangle's pop-up menu.

4   To hide the border of the rectangle, set the Pen Style to No Visible Line.

5   Click the button next to the Brush Color property on the General page.

6   In the Modify Expression dialog box, enter the following expression for the Brush.Color property:

```
If(mod(getrow(),2 )=1, rgb(255, 255, 255 ), rgb(240,
240, 240 ))
```

The mod function takes the row number (getrow()), divides it by 2, then returns the remainder. The remainder can be either 0 or 1. If the row number is odd, mod returns 1; if the row number is even, mod returns 0.

The expression mod(getrow(),2)=1 distinguishes odd rows from even rows.

The rgb function specifies maximum amounts of red, green, and blue: rgb (255, 255, 255). Specifying 255 for red, green, and blue results in the color white.

If the row number is odd (the condition evaluates as true), the rectangle displays as white. If the row number is even (the condition evaluates as false), the rectangle displays as light gray (rgb (240, 240, 240)).

Example 2: rotating controls

The following DataWindow object shows the column headers for Health Insurance, Life Insurance, and Day Care rotated 45 degrees.

Total Compensation Report
Salary Plus Benefits

Value of health ins. = $4,800
Value of life insurance = $(5.43 × salary)/1,000
Value of day care = $5,200

Page 1 of 19
03/22/03

Department ID	Employee ID	Employee First Name	Employee Last Name	Salary	Health Ins.	Life Ins.	Day Care	Salary Plus Benefits
100	102	Fran	Whitney	$45,700	☑	☑	☐	$50,748
	105	Matthew	Cobb	$62,000	☑	☑	☐	$67,137
	160	Robert	Breault	$57,490	☑	☑	☑	$67,802
	243	Natasha	Shishov	$72,995	☑	☑	☐	$78,191
	247	Kurt	Driscoll	$48,024	☑	☑	☑	$58,284

To rotate each of these three text controls:

1   Select one of the controls, then use Ctrl + click to select the other two controls.

The Properties view changes to show the properties that are common to all selected controls.

2   On the Font page in the Properties view, click the button next to the Escapement property.

3   Enter the number 450 in the Modify Expression dialog box and click OK.

The value entered for font escapement is in tenths of degrees, so the number 450 means 45 degrees. You do not have to specify a condition. Typically, you do not specify a condition for control rotation.

The rotation of the controls does not change in the Design view.

4   To see the change, close and reopen the Preview view.

Example 3: highlighting rows of data

The following DataWindow object is an employee phone list for a company in Massachusetts. Out-of-state (not in Massachusetts) employees are shown in bold and preceded by two asterisks (**):

This DataWindow object uses newspaper columns. To understand how to create this DataWindow object without highlighting data, see "Printing with newspaper-style columns" on page 544.

In the Design view, the detail band includes four controls: the employee last name, a comma, the employee first name, and the phone number:

To make these controls display in bold with two asterisks if the employee is not from Massachusetts:

1 Select one of the controls, then use Ctrl + Click to select the other three controls.

The Properties view changes to show the properties that are common to all selected controls.

2 On the Font page in the Properties view, click the button next to the Bold property.

3 Enter the following expression in the Modify Expression dialog box and click OK:

```
If(state = 'MA', 400, 700)
```

The expression states that if the value of the state column is MA, use 400 as the font weight. This means employees from Massachusetts display in the normal font. For any state except MA, use 700 as the font weight. This means all other employees display in bold font.

Logic that relies on the state column
To use logic that relies on the state column, you need to include the column in the data source. You can add the column after creating the DataWindow object by modifying the data source. Notice that the state column does not actually appear anywhere in the DataWindow object. Values must be available but do not need to be included in the DataWindow object.

---

4 To insert two asterisks (**) in front of the employee name if the employee is not from Massachusetts, add a text control to the left of the employee name with the two asterisks in bold.

5 With the text control selected, click the button next to its Visible property on the General page in the Properties view.

6 In the Modify Expression dialog box that displays, enter the following expression and click OK:

```
If(state = 'MA', 0, 1)
```

This expression says that if the state of the employee is MA (the true condition), the Visible property of the ** control is off (indicated by 0). If the state of the employee is not MA (the false condition), the Visible property of the ** control is on (indicated by 1). The asterisks are visible next to that employee's name.

Tip
You can use underlines, italics, strikethrough, borders, and colors to
highlight information.

Example 4: changing the size and location of controls

The following DataWindow object shows city and state columns enclosed in a
rectangle and underlined. The columns change location if the current row
contains data for a customer from the state of New York. The rectangle and the
line change both location and size.

This example shows how to move the rectangle and line. The process for
columns is similar.

In the Design view, the rectangle and line display in one location, with a single
set of dimensions. The expressions you specify are used only in Preview view
and at runtime and all have the following syntax:

If (state='NY', *true value, false value*)

The *false value* is the same as the value in Design view. All of the values used
in this example are in PowerBuilder Units (PBUs), the default unit of measure
used for the DataWindow object.

To change properties of the rectangle and the line for rows with the state column equal to New York:

1   Select the rectangle, display the Position page in the Properties view, and specify expressions for the following properties:

Property	Expression
X	if (state = 'NY', 2890, 1865)
Width	if (state = 'NY', 500, 1000)
Height	if (state = 'NY', 160, 120)

2   Select the line, display the Position page in the Properties view, and specify expressions for the following properties:

Property	Expression
X1	if (state = 'NY', 2890, 1865)
Y1	if (state = 'NY', 168, 132)
X2	if (state = 'NY', 3400, 2865)
Y2	if (state = 'NY', 168, 132)

3   On the General page for the line, specify this expression for Pen Width:

```
if (state = 'NY', 10, 4)
```

At runtime, the rectangle is taller and narrower, and the line is shorter and has a wider pen width.

Supplying property values

Each property has its own set of property values that you can use to specify the true and false conditions in the If expression. Usually you specify a number to indicate what you want. For example, if you are working with the Border property, you use the number 0, 1, 2, 3, 4, 5, or 6 to specify a border.

Table 24-1 summarizes the properties available. A detailed description of each property follows the table. For a complete list of properties for each control, see the online Help.

Valid values of properties are shown in parentheses in the Properties view wherever possible.
For example, the drop-down list showing border selections includes the correct number for specifying each border in parentheses after the name of the border (ShadowBox (1), Underline (4)).

Table 24-1: Properties for controls in the DataWindow painter

Property	Painter option in Properties view	Description
Background.Color	Background Color on General page or Font page	Background color of a control
Border	Border on General page	Border of a control
Brush.Color	Brush Color on General page	Color of a graphic control
Brush.Hatch	Brush Hatch on General page	Pattern used to fill a graphic control
Color	Text Color on Font page; Color on General page; Line Color on General page	Color of text for text controls, columns, and computed fields; background color for the DataWindow object; line color for graphs
Font.Escapement (for rotating controls)	Escapement on Font page	Rotation of a control
Font.Height	Size on Font page	Height of text
Font.Italic	Italic on Font page	Use of italic font for text
Font.Strikethrough	Strikeout on Font page	Use of strikethrough for text
Font.Underline	Underline on Font page	Use of underlining for text
Font.Weight	Bold on Font page	Weight (for example, bold) of text font
Format	Format on Format page	Display format for columns and computed fields
Height	Height on Position page	Height of a control
Pen.Color	Pen Color on General page	Color of a line or the line surrounding a graphic control
Pen.Style	Pen Style on General page	Style of a line or the line surrounding a graphic control
Pen.Width	Pen Width on General page	Width of a line or the line surrounding a graphic control
Pointer	Pointer on Pointer page	Image to be used for the pointer
Protect	Protect on General page	Whether a column can be edited
Timer_Interval	Timer Interval on General page	How often time fields are to be updated
Visible	Visible on General page	Whether a control is visible
Width	Width on Position page	Width of a control
X	X on Position page	X position of a control

Property	Painter option in Properties view	Description
X1, X2	X1, X2 on Position page	X coordinates of either end of a line
Y	Y on Position page	Y position of a control relative to the band in which it is located
Y1, Y2	Y1, Y2 on Position page	Y coordinates of either end of a line

Background.Color

Description	Setting for the background color of a control.
In the painter	Background Color on the General page or Font page in the Properties view.
Value	A number that specifies the control's background color.
	For information on specifying colors, see "Specifying colors" on page 684.
	The background color of a line is the color that displays between the segments of the line when the pen style is not solid.
	If Background.Mode is transparent (1), Background.Color is ignored.
Example	The following statement specifies that if the person represented by the current row uses the day care benefit, the background color of the control is set to light gray (15790320). If not, the background color is set to white (16777215):

```
If(bene_day_care = 'Y', 15790320, 16777215)
```

In this example, the condition is applied to the Background.Color property for three controls: the emp_id column, the emp_fname column, and the emp_lname column.

The following is a portion of the resulting DataWindow object. Notice that the employee ID, first name, and last name have a gray background if the employee uses the day care benefit:

Employee ID	Employee First Name	Employee Last Name	Salary	Health Ins.	Life Ins.	Day Care	Salary Plus Benefits
102	Fran	Whitney	$45,700	☒	☒	☐	$50,748
105	Matthew	Cobb	$62,000	☒	☒	☐	$67,137
160	Robert	Breault	$57,490	☒	☒	☒	$67,802
243	Natasha	Shishov	$72,995	☒	☒	☐	$78,191
247	Kurt	Driscoll	$48,024	☒	☒	☒	$58,284
249	Rodrigo	Guevara	$42,998	☒	☒	☐	$48,031
266	Ram	Gowda	$59,840	☐	☒	☐	$60,165
278	Terry	Melkisetian	$48,500	☒	☒	☒	$58,763
316	Lynn	Pastor	$74,500	☒	☒	☒	$84,905
445	Kim	Lull	$87,900	☒	☒	☐	$93,177

Border

Description | The type of border for the control.

In the painter | Border on the General page in the Properties view.

Value | A number that specifies the type of border. Values are:

    0—None
    1—Shadow box
    2—Box
    3—Resize
    4—Underline
    5—3D Lowered
    6—3D Raised

Example | The following statement specifies that if the person represented by the current row has a status of L (on leave), the status column displays with a Shadow box border:

```
If(status = 'L', 1, 0)
```

In this example, the condition is applied to the Border property of the status column.

The following is a portion of the resulting DataWindow object. Notice that the status On Leave displays with a Shadow box border:

Emp ID	Employee First Name	Employee Last Name	Street	City	State	Zip Code	Phone	Status	Soc. Sec. No.
102	Fran	Whitney	49 East Washington Street	Needham	MA	02192-	(617) 555-3985	Active	017-34-9033
105	Matthew	Cobb	77 Pleasant Street	Waltham	MA	02154-	(617) 555-3840	Active	052-34-5739
129	Philip	Chin	59 Pond Street	Atlanta	GA	30339-	(404) 555-2341	Active	024-60-8923
148	Julie	Jordan	144 Great Plain Avenue	Winchester	MA	01890-	(617) 555-7835	Active	501-70-4733
160	Robert	Breault	58 Cherry Street	Milton	MA	02186-	(617) 555-3099	Active	025-48-7623
184	Melissa	Espinoza	112 Apple Tree Way	Stow	MA	01775-	(508) 555-2319	Active	025-48-1943
191	Jeannette	Bertrand	209 Concord Street	Acton	MA	01720-	(508) 555-8138	Active	017-34-8821
195	Marc	Dill	89 Hancock Street	Milton	MA	02186-	(617) 555-2144	Active	079-48-6634
207	Jane	Francis	12 Hawthorne Drive	Concord	MA	01742-	(508) 555-9022	Active	501-70-8992
243	Natasha	Shishov	15 Milk Street	Waltham	MA	02154-	(617) 555-2755	Active	043-21-6799
247	Kurt	Driscoll	154 School Street	Waltham	MA	02154-	(617) 555-1234	On Leave	024-60-1768
249	Rodrigo	Guevara	East Main Street	Framingham	MA	01701-	(508) 555-0029	Active	084-32-9990
266	Ram	Gowda	79 Page Street	Natick	MA	01760-	(508) 555-8722	Active	017-34-6122

About the value L and the value On Leave

The status column uses an edit style. The internal value for on leave is L and the display value is On Leave. The conditional expression references the internal value L, which is the actual value stored in the database. The DataWindow object shows the value On Leave, which is the display value assigned to the value L in the code table for the Status edit style.

Brush.Color

Description	Setting for the fill color of a graphic control.
In the painter	Brush Color on the General page in the Properties view.
Value	A number that specifies the color that fills the control.
	For information on specifying colors, see "Specifying colors" on page 684.
Example	See the example for "Brush.Hatch" next.

Brush.Hatch

Description   Setting for the fill pattern of a graphic control.

In the painter   Brush Hatch on the General page in the Properties view.

Value   A number that specifies the pattern that fills the control. Values are:

> 0—Horizontal
> 1—Bdiagonal (lines from lower left to upper right)
> 2—Vertical
> 3—Cross
> 4—Fdiagonal (lines from upper left to lower right)
> 5—DiagCross
> 6—Solid
> 7—Transparent

Example   In this example, statements check the employee's start date to see if the month is the current month or the month following the current month. Properties of a rectangle control placed behind the row of data are changed to highlight employees with months of hire that match the current month or the month following the current month.

The Design view includes columns of data and a rectangle behind the data. The rectangle has been changed to black in the following picture to make it stand out:
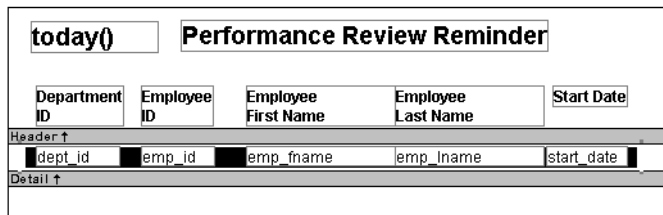
The following statement is for the Brush.Color property of the rectangle. If the month of the start date matches the current month or the next one, Brush.Color is set to light gray (12632256). If not, it is set to white (16777215), which means it will not show:

```
If(month( start_date ) = month(today())
or month( start_date ) = month(today())+1
or (month(today()) = 12 and month(start_date)=1),
12632256, 16777215)
```

The following statement is for the Brush.Hatch property of the rectangle. If the month of the start date matches the current month or the next one, Brush.Hatch is set to Bdiagonal (1). If not, it is set to Transparent (7), which means it will not show:

```
If(month( start_date ) = month(today())
or month( start_date ) = month(today())+1
or (month(today()) = 12 and month(start_date)=1),
1, 7)
```

Expressions are also provided for Pen.Color and Pen.Style.

For more about these properties and a picture, see "Pen.Style" on page 678.

Color

Description	The color of text for text controls, columns, and computed fields; background color for the DataWindow object; line color for graphs.
In the painter	In the Properties view, Text Color on the Font property page; Color on the General property page; Line Color on the General property page.
Value	A number that specifies the color used for text.
	For information on specifying colors, see "Specifying colors" on page 684.
Example	The following statement is for the Color property of the emp_id, emp_fname, emp_lname, and emp_birth_date columns:

```
If(month(birth_date) = month (today()), 255, 0)
```

If the employee has a birthday in the current month, the information for the employee displays in red (255). Otherwise, the information displays in black (0).

The Font.Underline property has the same conditional expression defined for it so that the example shows clearly on paper when printed in black and white.

Font.Escapement (for rotating controls)

Description
:   The angle of rotation from the baseline of the text.

In the painter
:   Escapement on the Font page in the Properties view.

Value
:   An integer in tenths of degrees. For example, 450 means 45 degrees. 0 is horizontal.

Example
:   To enter rotation for a control, select the control in the Design view and click the button next to the Escapement property in the Properties view. In the dialog box that displays, enter the number of tenths of degrees.

The following picture shows the Design view with a number of text controls. Each text control shows the Font.Escapement value entered and the number of degrees of rotation. In the Design view, you do not see rotation; it looks as if the controls are all mixed up. Two controls seem to overlie each other:
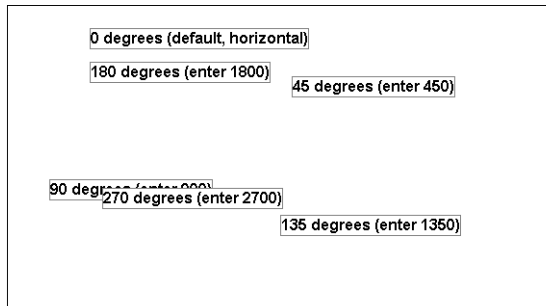
The next picture shows the same controls at runtime. Each control is rotated appropriately:
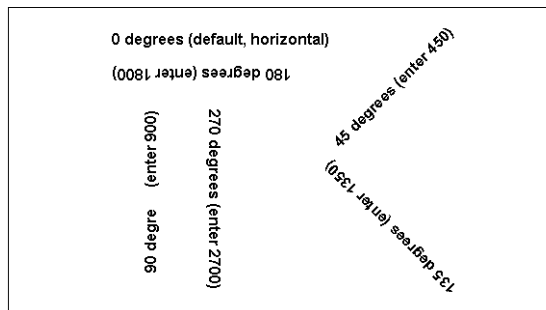
How to position controls that are rotated
Make the controls movable. To do so, display each control and select the
Moveable check box in the Position page. Then in the Preview view, click the
rotated text control until a gray box displays (try the center of the text). Drag
the rotated control where you want it. In the Design view, the controls will be
wherever you dragged them. They may look incorrectly positioned in the
Design view, but they will be correctly positioned when you run the
DataWindow object. When you are satisfied with the positioning, you can clear
the Moveable check box for the controls to ensure that they stay where you
want them.

Font.Height

Description   The height of the text.

In the painter   Size on the Font page in the Properties view.

Value   An integer in the unit of measure specified for the DataWindow object. Units
                             of measure include PowerBuilder units, thousandths of an inch (1000 = 1 inch),
                             thousandths of a centimeter (1000 = 1 centimeter), or pixels. To specify size in
                             points, specify a negative number.

Example   The following statement is specified for the Font.Height property of a text
                             control. Note that the DataWindow object is defined as using thousandths of an
                             inch as its unit of measure. The statement says that if the control is in the first
                             row, show the text 1/2-inch high (500 1/1000ths of an inch) and if it is not the
                             first, show the text 1/5-inch high (200 1/1000ths of an inch):

```
If(GetRow() = 1, 500, 200)
```

The boundaries of the control might need to be extended to allow for the
increased size of the text. At runtime, the first occurrence of the text control is
big (1/2 inch); subsequent ones are small (1/5 inch).

Font.Italic

Description | A number that specifies whether the text should be italic.

In the painter | Italic on the Font page in the Properties view.

Value | Values are:

0—Not italic
1—Italic

Example | The following statements are specified for the Font.Italic, Font.Underline, and Font.Weight properties, respectively. If the employee has health insurance, the employee's information displays in italics. If not, the employee's information displays in bold and underlined:

```
If(bene_health_ins = 'Y', 1, 0)
If(bene_health_ins = 'N', 1, 0)
If(bene_health_ins = 'N', 700, 400)
```

Statements are specified in this way for four controls: the emp_id column, the emp_fname column, the emp_lname column, and the emp_salary column. In the resulting DataWindow object, those with health insurance display in italics. Those without health insurance are emphasized with bold and underlining:

Font.Strikethrough

Description A number that specifies whether the text should be crossed out.

In the painter Strikeout on the Font page in the Properties view.

Value Values are:

0—Not crossed out
1—Crossed out

Example The following statement is for the Font.Strikethrough property of the emp_id,
emp_fname, emp_lname, and emp_salary columns. The status column must be
included in the data source even though it does not appear in the DataWindow
object itself. The statement says that if the employee's status is L, which means
On Leave, cross out the text in the control:

```
If(status = 'L', 1, 0)
```

An extra text control is included to the right of the detail line. It becomes visible
only if the status of the row is L (see "Visible" on page 681).

The following is a portion of the resulting DataWindow object. It shows two
employees who are On Leave. The four columns of information show as
crossed out:

Font.Underline

Description	A number that specifies whether the text should be underlined.
In the painter	Underline on the Font page in the Properties view.
Value	Values are:

        0—Not underlined
        1—Underlined

Example   The following statement, when applied to the Font.Underline property of columns of employee information, causes the information to be underlined if the employee does not have health insurance:

```
If(bene_health_ins = 'N', 1, 0)
```

For pictures of this example, see "Font.Italic" on page 674.

Font.Weight

Description	The weight of the text.
In the painter	Bold on the Font page in the Properties view.
Value	Values are:

        100—Thin
        200—Extra light
        300—Light
        400—Normal
        500—Medium
        600—Semibold
        700—Bold
        800—Extrabold
        900—Heavy

Most commonly used values
The most commonly used values are 400 (Normal) and 700 (Bold). Your printer driver might not support all of the settings.

Example   The following statement, when applied to the Font.Weight property of columns of employee information, causes the information to be displayed in bold if the employee does not have health insurance:

```
If(bene_health_ins = 'N', 700, 400)
```

For pictures of this example, see "Font.Italic" on page 674.

Format

Description   The display format for a column.

In the painter   Format on the Format page in the Properties view.

Values   A string specifying the display format.

Example   The following statement, when applied to the Format property of the Salary
                        column, causes the column to display the word Overpaid for any salary
                        greater than $60,000 and Underpaid for any salary under $60,000:

```
If(salary>60000, 'Overpaid', 'Underpaid')
```

Edit Mask edit style change
The Edit Mask edit style assigned to the salary column had to be changed.
Because edit styles take precedence over display formats, it was necessary to
change the edit style assigned to the salary column (an Edit Mask edit style) to
the Edit edit style.

Height

Description   The height of the column or other control.

In the painter   Height on the Position page in the Properties view.

Value   An integer in the unit of measure specified for the DataWindow object. Units
                        of measure include PowerBuilder units, thousandths of an inch (1000 = 1 inch),
                        thousandths of a centimeter (1000 = 1 centimeter), or pixels.

Example   The following statement causes the height of a rectangle to be 160
                        PowerBuilder units if the state column for the row has the value NY.
                        Otherwise, the rectangle is 120 PowerBuilder units high:

```
if (state = 'NY', 160, 120)
```

For more details and pictures, see "Example 4: changing the size and location
of controls" on page 664.

Pen.Color

Description	The color of the line or the outline of a graphic control.
In the painter	Pen Color on the General page in the Properties view.
Value	A number that specifies the color of the line or outline.
	For information on specifying colors, see "Specifying colors" on page 684.
Example	See the example for the Pen.Style property, next.

Pen.Style

Description	The style of the line or the outline of a graphic control.
In the painter	Pen Style on the General page in the Properties view.
Value	Values are:

0—Solid
1—Dash
2—Dotted
3—Dash-dot pattern
4—Dash-dot-dot pattern
5—Null (no visible line)

Example   In this example, statements check the employee's start date to see if the month is the current month or the month following the current month. Properties of a rectangle control placed behind the row of data are changed to highlight employees with months of hire that match the current month or the month following the current month.

The Design view includes columns of data and a rectangle behind the data. The rectangle has been changed to black in the following picture to make it stand out:
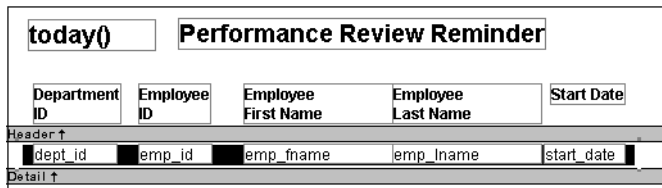
The following statement is for the Pen.Color property of the line around the edge of the rectangle. If the month of the start date matches the current month or the next one, Pen.Color is set to light gray (12632256). If not, it is set to white (16777215), which means it will not show:

```
If(month( start_date ) = month(today())
or month( start_date ) = month(today())+1
or (month(today()) = 12 and month(start_date)=1),
12632256, 16777215)
```

The following statement is for the Pen.Style property of the rectangle. If the month of the start date matches the current month or the next one, Pen.Style is set to Solid (0). If not, it is set to NULL (5), which means it will not show:

```
If(month( start_date ) = month(today())
or month( start_date ) = month(today())+1
or (month(today()) = 12 and month(start_date)=1),
0, 5)
```

Expressions are also defined for Brush.Color and Brush.Hatch.

For more about these properties, see "Brush.Hatch" on page 670.

The following is a portion of the resulting DataWindow object. A rectangle with light gray cross-hatching highlights employees whose reviews are due soon. The line enclosing the rectangle is Light Gray and uses the pen style Solid (0):

01/11/01		**Performance Review Reminder**			
Department ID	**Employee ID**	**Employee First Name**	**Employee Last Name**	**Start Date**	
400	888	Doug	Charlton	03/10/1991	
200	902	Moira	Kelly	04/01/1991	
200	913	Ken	Martel	04/16/1991	
500	921	Charles	Crowley	04/22/1991	
200	930	Ann	Taylor	05/08/1991	
200	949	Pamela	Savarino	05/08/1991	
100	958	Thomas	Sisson	07/16/1991	
400	992	Joyce	Butterfield	08/13/1991	
500	1013	Joseph	Barker	09/10/1991	
200	1021	Paul	Sterling	10/28/1991	
200	1039	Shih Lin	Chao	11/11/1991	
400	1062	Barbara	Blaikie	11/20/1991	
100	1090	Susan	Smith	12/13/1991	
200	1101	Mark	Preston	01/09/1992	*Review due this month*
200	1142	Alison	Clark	01/19/1992	*Review due this month*
100	1157	Hing	Soo	01/29/1992	*Review due this month*
200	1162	Kevin	Goggin	02/03/1992	*Review due next month*
400	1191	Matthew	Bucceri	02/12/1992	*Review due next month*

Pen.Width

Description	The width of the line or the outline of a graphic control.
In the painter	Pen Width on the General page in the Properties view.
Value	An integer in the unit of measure specified for the DataWindow object. Units of measure include PowerBuilder units, thousandths of an inch (1000 = 1 inch), thousandths of a centimeter (1000 = 1 centimeter), or pixels.
Example	The following statement causes the width of a line to be 10 PowerBuilder units if the state column for the row has the value NY. Otherwise, the line is 4 PowerBuilder units wide:

```
If(state = 'NY', 10, 4)
```

For more details and pictures, see "Example 4: changing the size and location of controls" on page 664.

Pointer

Description	The image used for the mouse pointer when the pointer is over the specified control.
In the painter	Pointer on the Pointer page in the Properties view.
Value	A string that specifies a value of the Pointer enumerated data type or the name of a cursor file (CUR) used for the pointer.

Values of the Pointer enumerated data type are:

Arrow!
Cross!
HourGlass!
IBeam!
Icon!
Size!
SizeNESW!
SizeNS!
SizeNWSE!
SizeWE!
UpArrow!

Example

The following condition, entered for the Pointer property of every control in a row of expense data, changes the pointer to a column every time the value in the expense column exceeds $100,000. Note that the pointer has no meaning in a printed report. The pointer is for use on the screen display of a DataWindow object:

```
If(expense 100000, 'pbcolumn.cur', 'arrow!')
```

Protect

Description

The protection setting of a column.

In the painter

Protect on the General page in the Properties view.

Value

Values are:

0—False, the column is not protected
1—True, the column is protected

Timer_Interval

Description

The number of milliseconds between the internal timer events.

In the painter

Timer Interval on the General page in the Properties view.

Value

The default is 0 (which is defined to mean 60,000 milliseconds or one minute).

Visible

Description

Whether the control is visible in the DataWindow object.

In the painter

Visible on the General page in the Properties view.

Value

Values are:

0—Not visible
1—Visible

Example

The following statement is for the Visible property of a text control with the words On Leave located to the right of columns of employee information. The statement says that if the current employee's status is L, which means On Leave, the text control is visible. Otherwise, it is invisible:

```
If(status = 'L', 1, 0)
```

The status column must be retrieved
The status column must be included in the data source even though it does not appear in the DataWindow object itself.

The Design view includes the text control at the right-hand end of the detail line. The text control is visible at runtime only if the value of the status column for the row is L.

In the resulting DataWindow object, the text control is visible only for the two employees on leave. For a picture, see "Font.Strikethrough" on page 675.

Width

Description	The width of the control.
In the painter	Width on the Position page in the Properties view.
Value	An integer in the unit of measure specified for the DataWindow object. Units of measure include PowerBuilder units, thousandths of an inch (1000 = 1 inch), thousandths of a centimeter (1000 = 1 centimeter), or pixels.
Example	The following statement causes the width of a rectangle to be 500 PowerBuilder units if the state column for the row has the value NY. Otherwise, the rectangle is 1000 PowerBuilder units wide:

```
if (state = 'NY', 500, 1000)
```

For more details and pictures, see "Example 4: changing the size and location of controls" on page 664.

X

Description	The distance of the control from the left edge of the DataWindow object. At runtime, the distance from the left edge of the DataWindow object is calculated by adding the margin to the x value.
In the painter	X on the Position page in the Properties view.
Value	An integer in the unit of measure specified for the DataWindow object. Units of measure include PowerBuilder units, thousandths of an inch (1000 = 1 inch), thousandths of a centimeter (1000 = 1 centimeter), or pixels.

Example
The following statement causes a rectangle to be located 6.250 inches from the left if the state column for the row has the value NY. Otherwise, the rectangle is 4.000 inches from the left:

```
If(state = 'NY', 6250, 4000)
```

For more details and pictures, see "Example 4: changing the size and location of controls" on page 664.

X1, X2

Description
The distance of each end of the line from the left edge of the DataWindow object as measured in the Design view. At runtime, the distance from the left edge of the DataWindow object is calculated by adding the margin to the x1 and x2 values.

In the painter
X1, X2 on the Position page in the Properties view.

Value
Integers in the unit of measure specified for the DataWindow object. Units of measure include PowerBuilder units, thousandths of an inch (1000 = 1 inch), thousandths of a centimeter (1000 = 1 centimeter), or pixels.

Example
The following statements for the X1 and X2 properties of a line cause the line to extend from 6.250 to 7.150 inches from the left if the state column for the row has the value NY. Otherwise, the line extends from 4.000 to 6.000 inches from the left:

```
If(state = 'NY', 6250, 4000)
If(state = 'NY', 7150, 6000)
```

For more details and pictures, see "Example 4: changing the size and location of controls" on page 664.

Y

Description
The distance of the control from the top of the band in which the control is located.

In the painter
Y on the Position page in the Properties view.

Value
An integer in the unit of measure specified for the DataWindow object. Units of measure include PowerBuilder units, thousandths of an inch (1000 = 1 inch), thousandths of a centimeter (1000 = 1 centimeter), or pixels.

Example	For information, see "Example 4: changing the size and location of controls" on page 664.

Y1, Y2

Description	The distance of each end of the specified line from the top of the band in which the line is located.
In the painter	Y1, Y2 on the Position page in the Properties view.
Value	Integers in the unit of measure specified for the DataWindow object. Units of measure include PowerBuilder units, thousandths of an inch (1000 = 1 inch), thousandths of a centimeter (1000 = 1 centimeter), or pixels.
Example	The following statements for the Y1 and Y2 properties of a line cause the line to be located .400 inches (Y1 and Y2 equal .400 inches) from the top of the detail band, if the state column for the row has the value NY. Otherwise, the line is located .250 inches (Y1 and Y2 equal .250 inches) from the top of the detail band:

```
If(state = 'NY', 400, 250)
If(state = 'NY', 400, 250)
```

For more details and pictures, see "Example 4: changing the size and location of controls" on page 664.

Specifying colors

You specify a color by specifying a number that represents the color. You can specify the number explicitly or by using an expression that includes the RGB (*r*, *g*, *b*) function.

For the numbers and expressions that specify common colors, see Table 24-2 on page 685.

How the number is calculated

The formula for combining color values into a number is:

red + 256*green + 256*256*blue

where the amount of each primary color (red, green, and blue) is specified as a value from 0 to 255.

The RGB function calculates the number from the amounts of red, green, and blue specified.

Sample numeric calculation	To create cyan, you use blue and green, but no red. If you wanted to create the most saturated (bright) cyan, you would use maximum amounts of blue and green in the formula, which is indicated by the number 255 for each. The following statements show the calculation:

$red + 256 * green + 256 * 256 * blue$

$0 + 256 * 255 + 256 * 256 * 255$

$0 + 65280 + 16711680$

16776960

Sample expression using the RGB function	The following expression specifies the brightest cyan:

RGB (*0,255,255*)

Notice that the expression specifies the maximum for green and blue (255) and 0 for red. The expression returns the value 16776960. To specify cyan, entering the expression RGB(0, 255, 255) is the same as entering the number 16776960.

Numbers and expressions to enter for the common colors	Table 24-2 shows the numbers and expressions to enter for some common colors. The number and expression for a color are equivalent. You can use either.

Table 24-2: Numbers and expressions for common colors

Color	Expression to enter	Number to enter	How the number is calculated
Black	RGB (0, 0, 0)	0	0 (no color)
Blue	RGB (0, 0, 255)	16711680	256*256*255 (blue only)
Cyan	RGB (0, 255, 255)	16776960	256*255 + 256*256*255 (green and blue)
Dark Green	RGB (0, 128, 0)	32768	256*128 (green only)
Green	RGB (0, 255, 0)	65280	256*255 (green only)
Light Gray	RGB (192, 192, 192)	12632256	192 + 256*192 + 256*256*192 (some red, green, and blue in equal amounts)
Lighter Gray	RGB (224, 224, 224)	14737632	224 + 256*224 + 256*256*224 (some red, green, and blue in equal amounts)
Lightest Gray	RGB (240, 240, 240)	15790320	240 + 256*240 + 256*256*240 (some red, green, and blue in equal amounts)
Magenta	RGB (255, 0, 255)	16711935	255 + 256*256*255 (red and blue)
Red	RGB (255, 0, 0)	255	255 (red only)
White	RGB (255, 255, 255)	16777215	255 + 256*255 + 256*256*255 (red, green, and blue in equal amounts at the maximum of 255)
Yellow	RGB (255, 255, 0)	65535	255 + 256*255 (red and green)

CHAPTER 25 # Using Nested Reports

About this chapter

This chapter provides information about creating reports that have other reports nested in them.

Contents

Topic	Page
About nested reports	687
Creating a report using the Composite presentation style	691
Placing a nested report in another report	692
Working with nested reports	696

About reports and
DataWindow objects

A report is the same as a nonupdatable DataWindow object.

This chapter shows the process of nesting reports using the Report painter in InfoMaker, but you can do the same things in the DataWindow painter with the same results.

About nested reports

A nested report is a report within another report.

There are two ways to create reports containing nested reports:

- Create a composite report using the Composite presentation style
- Place a nested report in another report

About creating a
composite report

You can choose the Composite presentation style to create a new report that consists entirely of one or more nested reports. This type of report is called a composite report. A composite report is a container for other reports.

You can use composite reports to print more than one report on a page.

Composite report

For example, the following composite report consists of three tabular reports. One of the tabular reports includes a graph:

Composite report in the Design view

In the Design view, you see three boxes that represent the individual tabular reports that are included in the composite report. The only additional controls in this example are a title, date, and page number:

About placing a nested report within another report

You can place one or more reports within another report. The report you place is called the nested report. You can place a nested report in any type of report except crosstab. Most of the time you will place nested reports in freeform or tabular reports.

Often, the information in the nested report depends on information in the report in which it is placed (the base report). The nested report and the base report are related to each other by some common data. The base report and the nested report have a master/detail relationship.

Freeform report with a related nested report

For example, the following freeform report lists all information about a customer and then includes a related nested report (which happens to be a tabular report). The related nested report lists every order that the customer has ever placed. The base report supplies the customer ID to the nested report, which requires a customer ID as a retrieval argument. This is an example of a master/detail relationship—one customer has many orders:

What you see in the Design view

In the Design view, you see everything in the base report plus a box that represents the related nested report:
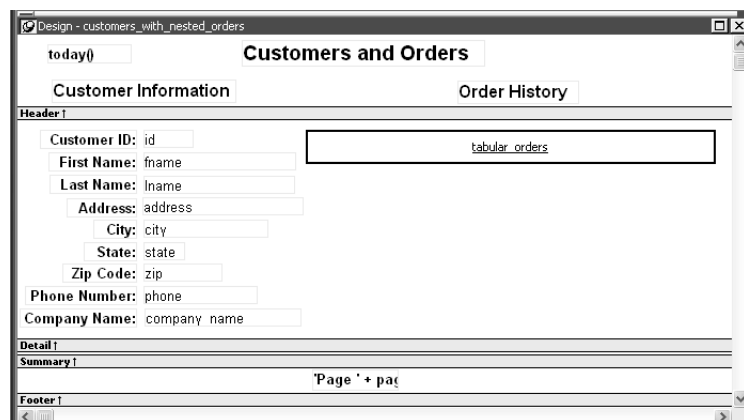
The difference between nested and composite reports

There are two important differences between nesting using the Composite style and nesting a report within a base report.

Data sources   The composite report does not have a data source—it is just a container for nested reports. In contrast, a base report with a nested report in it has a data source. The nested report has its own data source.

Related nesting   The composite report cannot be used to relate reports to each other in the database sense. One report cannot feed a value to another report, which is what happens in a master/detail report. If you want to relate reports to each other so that you can create a master/detail report, you need to place a nested report within a base report.

How retrieval works

When you preview (run) a composite report, PowerBuilder retrieves all the rows for one nested report, and then for another nested report, and so on until all retrieval is complete. Your computer must have a default printer specified, because composite reports are actually displayed in print preview mode.

When you preview (run) a report with another related report nested in it, PowerBuilder retrieves all the rows in the base report first. Then PowerBuilder retrieves the data for all nested reports related to the first row. Next, PowerBuilder retrieves data for nested reports related to the second row, and so on, until all retrieval is complete for all rows in the base report.

For information about efficiency and retrieval, see "Supplying retrieval arguments to relate a nested report to its base report" on page 698.

Limitations on nesting reports

For the most part you can nest the various types of report styles. However, limitations apply to two of them.

Crosstabs   You cannot place a crosstab with retrieval arguments within another report as a related nested report. However, you can include a crosstab in a Composite report.

RichText reports   You cannot nest a RichText report in any way. You cannot place a RichText report in another report, and you cannot include a RichText report in a Composite report.

Creating a report using the Composite presentation style

❖ **To create a report using the Composite presentation style:**

1 Select File>New from the menu bar.

The New Report dialog box displays.

2 Choose the DataWindow tab page and the Composite presentation style, and click OK.

The wizard displays all reports (DataWindow objects) that are in the current target's library search path.

3 Click the reports you want to include in the composite report and then click Next.

The wizard lists your choices.

4 Click Finish.

PowerBuilder places boxes for the selected reports in the Design view. In this example, you see three reports:

5 Select File>Save from the menu bar and assign a name to the composite report.

6   Look at the Preview view of the report:

Product Sales Summary

Product ID	Product Name	Product Description	Quantity Sold	Dollars
300	Tee Shirt	Tank Top	2374	$21,366
301	Tee Shirt	V-neck	2403	$33,642
302	Tee Shirt	Crew Neck	2139	$29,946
400	Baseball Cap	Cotton Cap	3310	$29,790
401	Baseball Cap	Wool cap	2677	$26,770
500	Visor	Cloth Visor	2652	$18,564
501	Visor	Plastic Visor	2508	$17,556
600	Sweatshirt	Hooded Sweatshirt	3060	$73,440
601	Sweatshirt	Zipped Sweatshirt	2748	$65,952
700	Shorts	Cotton Shorts	4536	$68,040

Sales Summary

Products and Current Inventory

Product ID	Product Name	Product Description	Unit Price	Number in Stock
300	Tee Shirt	Tank Top	$9.00	18
301	Tee Shirt	V-neck	$14.00	39
302	Tee Shirt	Crew Neck	$14.00	72
400	Baseball Cap	Cotton Cap	$9.00	92
401	Baseball Cap	Wool cap	$10.00	12
500	Visor	Cloth Visor	$7.00	36
501	Visor	Plastic Visor	$7.00	28
600	Sweatshirt	Hooded Sweatshirt	$24.00	39
601	Sweatshirt	Zipped Sweatshirt	$24.00	32
700	Shorts	Cotton Shorts	$15.00	80

Sales Representatives and Total Number of Orders

Sales Rep ID	Name	Phone	Number of Orders
129	Philip Chin	(404) 555-2341	57
148	Julie Jordan	(617) 555-7835	2
195	Marc Dill	(617) 555-2144	50
299	Rollin Overbey	(510) 555-7255	114
467	James Klobucher	(713) 555-8627	56
667	Mary Garcia	(713) 555-3431	54
690	Kathleen Poitras	(617) 555-3920	52
856	Samuel Singer	(508) 555-3255	55
902	Judy Snow	(508) 555-3769	47
949	Pamela Savarino	(310) 555-1857	52
1039	Shih Lin Chao	(617) 555-5921	1
1142	Alison Clark	(510) 555-9437	57
1596	Catherine Pickett	(617) 555-3478	53

Notice that you are in print preview (which is read-only).

Working with composite reports
Many of the options available for working with reports, such as
Rows>Filter, Rows>Import, and Rows>Sort, are disabled for a composite
report. If you want to use any of these options, you need to access the
nested report(s), where these options are available.

7   Continue to enhance the composite report (for example, add a date and
title).

Placing a nested report in another report

When you place a nested report in another report, the two reports can be
independent of each other, or they can be related in the database sense by
sharing some common data such as a customer number or a department
number. If the reports are related, you need to do some extra things to both the
base report and the related nested report.

Usually, when you place a report within a report rather than create a composite report, you want to relate the reports. Those instructions are first.

Placing a related nested report in another report

Typically, a related nested report provides the details for a master report. For example, a master report might provide information about customers. A related nested report placed in the master report could provide information about all the orders that belong to each customer.

❖ **To place a related nested report in another report:**

1   Create the nested report (DataWindow object) that you plan to place in the base report.

2   Define a retrieval argument for the nested report.

For example, suppose the nested report lists orders and you want to list orders for a particular customer. To define a retrieval argument, you would:

- Select Design>Data Source to go to the Select painter.

- Select Design>Retrieval Arguments from the menu bar in the Select painter.

- Define a retrieval argument in the Specify Retrieval Arguments dialog box. In the example, *customerID* is the name assigned to the retrieval argument.

3   Specify the retrieval argument in a WHERE clause for the SELECT statement.

The WHERE clause in this example tells the DBMS to retrieve rows where the value in the column cust_id equals the value of the argument *:customerid*:
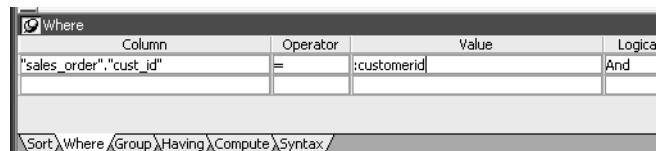
At this point, when you run the report to retrieve data, you are prompted to enter a value for *:customerid*. Later in these steps, you will specify that the base report supply the values for *:customerid* instead of prompting for values.

4   Open or create the report you want to have as the base report.

In the example, the base report is one that lists customers and has a place for the order history of each customer:
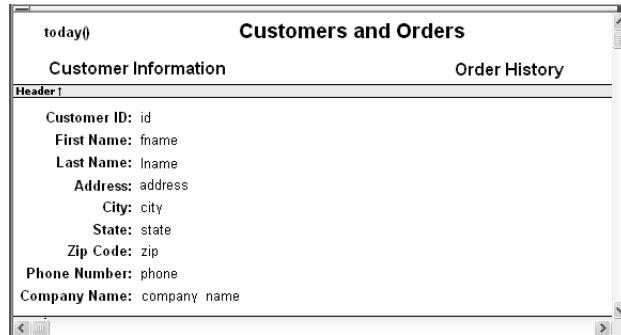
5   Select Insert>Control>Report from the menu bar.

6   In the Design view, click where you want to place the report.

The Select Report dialog box displays, listing defined reports (DataWindow objects) in the current target's library search path.

7   Select the report you want, and click OK.

A box representing the report displays in the Design view.

8   With the report still selected, select the General page of the Properties view.

The Arguments box lists arguments defined for the nested report and provides a way for you to specify how information from the base report will be used to supply the values of arguments to the nested report.
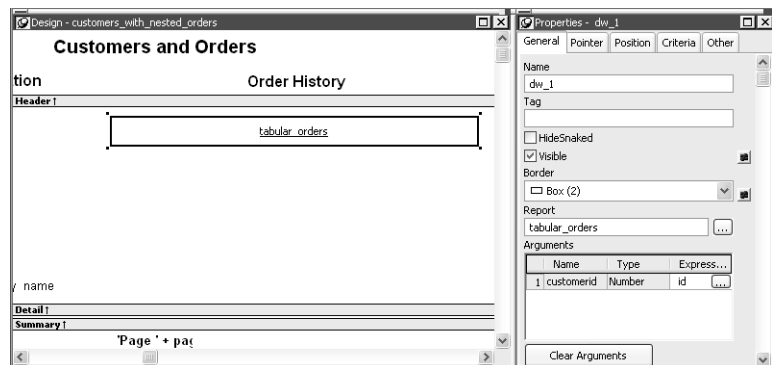
9   Supply the base report column or the expression that will supply the argument's value. To do this, click the button in the Expression column.

The Modify Expression dialog box displays. In this dialog box, you can easily select one of the columns or develop an expression. In the example, the column named id from the base report will supply the value for the argument *:customerid* in the nested report.

10   Select File>Save from the menu bar and assign a name to the report.

11   In the Preview view, you can see what your report looks like:

Placing an unrelated nested report in another report

When you place an unrelated nested report in a base report, the entire nested report appears with *each* row of the base report.

❖   **To place an unrelated nested report in another report:**

1   Create or open the report you want as the base report.

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

3   In the Design view, click where you want to place the report.

The Select Report dialog box displays, listing defined reports (DataWindow objects) in the current target's library search path.

4   Select the report you want to nest in the base report, and click OK.

A box representing the nested report displays in the Design view.

5   Select File>Save from the menu bar and if the base report is newly created, assign a name to it.

Working with nested reports

When you use nested reports either in composite reports or in other base reports, several enhancements and options are available. An easy way to see what you can do is to select the nested report and look at the Properties view for it.

Many of the options in the Properties view are described in Chapter 19, "Enhancing DataWindow Objects." For example, using borders on nested reports is like using borders on any control.

This section describes activities that apply only to nested reports or that have special meaning for nested reports. It covers:

- "Adjusting nested report width and height" next
- "Changing a nested report from one report to another" on page 697
- "Modifying the definition of a nested report" on page 697
- "Adding another nested report to a composite report" on page 698
- "Supplying retrieval arguments to relate a nested report to its base report" on page 698
- "Specifying criteria to relate a nested report to its base report" on page 700
- "Using options for nested reports" on page 701

Adjusting nested report width and height

When you preview a report with nested reports, the width of the nested report may be unacceptable. This can happen, for example, if you change the design of the nested report or if you use newspaper columns in a nested report. The width of the nested report is not adjusted to fit its contents at runtime; if the report is too narrow, some columns may be truncated. For example, if the size of the nested report is set to 6 inches wide in the parent report, columns in the nested report that exceed that width are not displayed in the parent report.

❖ **To adjust report width:**

1   In the Design view, position the pointer near a vertical edge of the nested report and press the left mouse button.

2   Drag the edge to widen the nested report.

3   Check the new width in the Preview view.

When you Print preview a DataWindow that contains a nested N-Up report with newspaper columns across the page, you might find that blank pages display (and print) when the nested report in the detail band fills the page. This is because any white space at the bottom of the band is printed to a second page. You can usually solve this problem by dragging up the detail band to eliminate the white space between the nested report and the band, or even to overlap the bottom of the representation of the nested report.

Changing a nested report from one report to another

You can change the nested report that is used. For example, you may work on several versions of a nested report and need to update the version of the nested report that the composite or base report uses.

❖   **To change the nested report to a different report:**

1   Select the nested report in the Design view.

2   In the Properties view, General property page, click the button next to the Report box.

3   Select the report you want to use, and click OK.

The name of the report that displays in the box in the Design view changes to the new one.

Modifying the definition of a nested report

You can modify the definition of the nested report. You can do this directly from the composite report or base report that contains the nested report.

❖   **To modify the definition of a nested report from the composite report or base report:**

1   Position the pointer on the nested report whose definition you want to modify, and display the pop-up menu.

2   Select Modify Report from the pop-up menu.

The nested report opens and displays in the painter. Both the composite or base report and the nested report are open.

3   Modify the report.

4   Select File>Close from the menu bar.

You are prompted to save your changes.

5   Click OK.

You return to the composite report or to the base report that includes the nested report.

Adding another nested report to a composite report

After you have created a composite report, you might want to add another report. The following procedure describes how. For information on adding a nested report to a report that is *not* a composite report, see "Placing a related nested report in another report" on page 693 or "Placing an unrelated nested report in another report" on page 695.

❖   **To add another nested report to a composite report:**

1   Open the composite report.

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

3   Click in the Design view where you want to place the report.

The Select Report dialog box displays, listing defined reports (DataWindow objects) in the current target's library search path.

4   Select the report you want and click OK.

A box representing the report displays in the Design view.

Supplying retrieval arguments to relate a nested report to its base report

The most efficient way to relate a nested report to its base report is to use retrieval arguments. If your nested report has arguments defined, you use the procedure described in this section to supply the retrieval argument value from the base report to the nested report. (The procedure described is part of the whole process covered in "Placing a related nested report in another report" on page 693.)

Why retrieval arguments are efficient	Some DBMSs have the ability to bind input variables in the WHERE clause of the SELECT statement. When you use retrieval arguments, a DBMS *with this capability* sets up placeholders in the WHERE clause and compiles the SELECT statement *once*. PowerBuilder retains this compiled form of the SELECT statement for use in subsequent retrieval requests.
Requirements for reusing the compiled *SELECT* statement	To enable PowerBuilder to retain and reuse the compiled SELECT statement:

- The database interface must support binding of input variables.

- You must enable binding support by setting the DisableBind DBParm to 0, which is the default.

- You must enable caching in the database profile. Set the SQLCache DBParm parameter to the number of levels of nesting plus 5.

    For more information, see the description of the SQLCache and DisableBind DBParm parameters in the online Help.

Nested reports in composite reports

If the base report is a composite report, you need to define retrieval arguments for the composite report before you can supply them to the nested report.

In the Properties view for the composite report, select the General page. Then define the retrieval arguments that the nested report needs, taking care to specify the correct type.

❖ **To supply a retrieval argument value from the base report to the nested report:**

1   Make sure that the nested report has been set up to take one or more retrieval arguments.

    See "Placing a nested report in another report" on page 692.

2   Select the nested report and then select the General page of the Properties view.

The Arguments box lists arguments defined for the nested report and provides a way for you to specify how information from the base report will supply the value of the argument to the nested report.
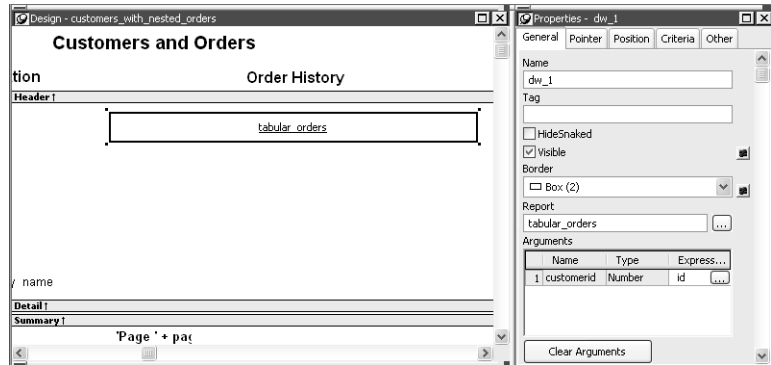
3   Supply the base report column that will supply the argument's value. To do this, click the button in the Expression column.

The Modify Expression dialog box displays. In this dialog box, you can easily select one of the columns or develop an expression. In the example, the column named id from the base report will supply the value for the argument *:customerid* in the nested report.

When you run the report now, you are not prompted for retrieval argument values for the nested report. The base report supplies the retrieval argument values automatically.

Specifying criteria to relate a nested report to its base report

If you do not have arguments defined for the nested report and if database efficiency is not an issue, you can place a nested report in another report and specify criteria to pass values to the related nested report.

How the DBMS processes SQL if you use the specify criteria technique

If you use the specify criteria technique, the DBMS repeatedly recompiles the SELECT statement and then executes it. The recompilation is necessary for each possible variation of the WHERE clause.

❖ **To specify criteria to relate a nested report to its base report:**

1 Select the nested report and then select the Criteria page in the Properties view.

   The Criteria property page provides a way for you to specify how information from the base report will supply the retrieval criteria to the nested report.

2 Click the button next to the criteria box.

   The Specify Retrieval Criteria dialog box displays.

3 Enter the retrieval criteria and click OK.

   The rules for specifying criteria are the same as for specifying criteria in the Quick Select data source. Multiple criteria in one line are ANDed together. Criteria entered on separate lines are ORed together.

   In this example, the customer ID (the id column) is the retrieval criterion being supplied to the nested report. Notice that the id column is preceded by a colon (:), which is required:
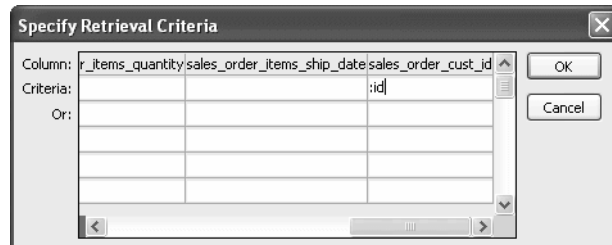
   When you run the report now, PowerBuilder retrieves rows in the nested report based on the criteria you have specified. In the example, the customer ID column in the base report determines which rows from the sales_order table are included for each customer.

Using options for nested reports

Using the Autosize Height option

Autosize Height should be on for all nested reports except graphs. This option ensures that the height of the nested report can change to accommodate the rows that are returned.

This option is on by default for all nested reports except graphs. Usually there is no reason to change it. If you do want to force a nested report to have a fixed height, you can turn this option off.

Note that all bands in the DataWindow also have an Autosize Height option. The option is off by default and must be on for the Autosize Height option for the nested report to work properly.

❖ **To change the Autosize Height option for a nested report:**

1　In the Design view, select the nested report.

2　In the Properties view, select the Position properties page.

3　Select/clear the Autosize Height check box.

Handling large rows
To avoid multiple blank pages or other anomalies in printed reports, never create a DataWindow object with a data row greater than the size of the target page. To handle large text-string columns, break the large string into a series of small strings. The smaller strings are used to populate individual data rows within a nested report instead of using a single AutoSize Height text column.

Using the Slide option

PowerBuilder determines the appropriate Slide options when positioning the nested report(s) and assigns default values. Usually, you should not change the default values:

• The Slide Left option is on by default for grid and crosstab style reports and off by default for all others. Having Slide Left on for grid and crosstab ensures that these reports break horizontally on whole columns and not in the middle of a column.

• The Slide Up All Above/Directly Above options ensure that the nested report uses just as much vertical space as it needs. One of these options is on by default for all nested reports.

For more information, see "Sliding controls to remove blank space in a DataWindow object" on page 579.

Using the New Page option (composite only)

The New Page option forces a new page for a nested report used *in a composite report*. By default, this option is off.

❖ **To specify that a nested report in a composite report should begin on a new page:**

1　In the Design view, select the nested report.

2　In the Properties view, select the General page.

3　Select the New Page check box.

A check mark displays, indicating the option is selected.

Using the Trail Footer option (composite only)

The Trail Footer option controls the placement of the footer for the last page of a nested report *in a composite report*. By default, this option is on. The footer appears directly under the contents of the nested report and not at the bottom of the page.

❖ **To specify that the footer should appear at the bottom of the page:**

1   In the Design view, select the nested report.

2   In the Properties view, select the General page.

3   Clear the Trail Footer check box.

   The check mark next to the option disappears, indicating the option is no longer selected. The footer appears at the bottom of the page on all pages of the nested report, including the last page. Note that if another nested report begins on the same page, the footer from the earlier report might be misleading or confusing.

Working with Graphs

About this chapter

This chapter describes how to build and use graphs in PowerBuilder.

Contents

About graphs

Often the best way to display information is graphically. Instead of showing users a series of rows and columns of data, you can present information as a graph in a DataWindow object or window. For example, in a sales application, you might want to present summary information in a column graph.

PowerBuilder provides many types of graphs and allows you to customize your graphs in many ways. Probably most of your use of graphs will be in a DataWindow object—the source of the data for your graphs will be the database.

You can also use graphs as standalone controls in windows (and user objects) and populate the graphs with data through scripts.

The way you define graphs is the same whether you are using them in a DataWindow object or directly in a window. However, the way you manipulate graphs in a DataWindow object is different from the way you manipulate them in a window.

Before using graphs in an application, you need to understand the parts of a graph and the kinds of graphs that PowerBuilder provides.

Parts of a graph

Here is a column graph created in PowerBuilder that contains most major parts of a graph. It shows quarterly sales of three products: Stellar, Cosmic, and Galactic printers:
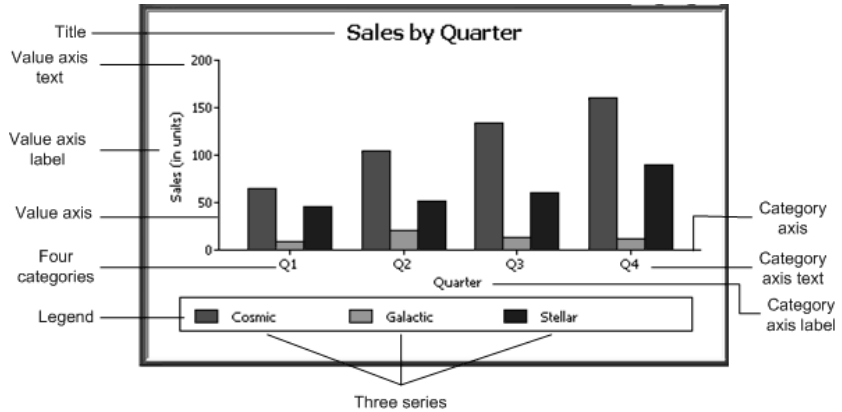
How data is represented

Graphs display data points. To define graphs, you need to know how the data is represented. PowerBuilder organizes data into three components.

Table 26-1: Components of a graph

Component	Meaning
Series	**A set of data points** Each set of related data points makes up one series. In the preceding graph, there is a series for Stellar sales, another series for Cosmic sales, and another series for Galactic sales. Each series in a graph is distinguished by color, pattern, or symbol.
Categories	**The major divisions of the data** Series data are divided into categories, which are often non-numeric. In the preceding graph, the series are divided into four categories: Q1, Q2, Q3, and Q4. Categories represent values of the independent variable(s).
Values	The values for the data points (dependent variables).

Organization of a graph

Table 26-2 lists the parts of a typical graph.

Table 26-2: Organization of a graph

Part of graph	What it is
Title	An optional title for the graph. The title appears at the top of the graph.
Value axis	The axis of the graph along which the values of the dependent variable(s) are plotted. In a column graph, as shown in the preceding graph, the Value axis corresponds to the y axis in an XY presentation. In other types of graphs, such as a bar graph, the Value axis can be along the x dimension.
Category axis	The axis along which are plotted the major divisions of the data, representing the independent variable(s). In the preceding graph, the Category axis corresponds to the x axis. It plots four categories: Q1, Q2, Q3, and Q4. These form the major divisions of data in the graph.
Series	A set of data points. There are three series in the preceding graph: Stellar, Cosmic, and Galactic. In bar and column charts, each series is represented by bars or columns of one color or pattern.
Series axis	The axis along which the series are plotted in three-dimensional (3D) graphs.
Legend	An optional listing of the series. The preceding graph contains a legend that shows how each series is represented in the graph.

Types of graphs

PowerBuilder provides many types of graphs for you to choose from. You choose the type on the Define Graph Style page in the DataWindow wizard or in the General page in the Properties view for the graph.
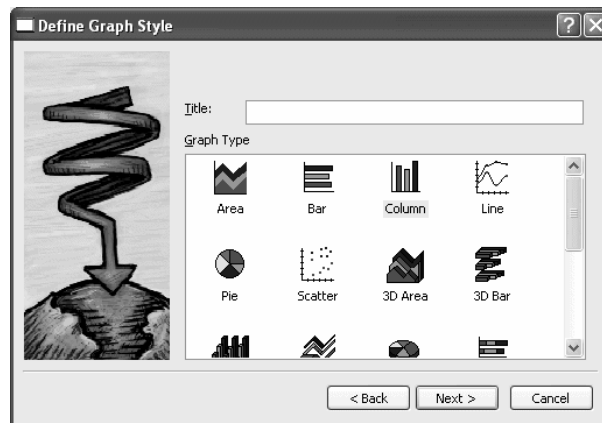
Area, bar, column, and line graphs

Area, bar, column, and line graphs are conceptually very similar. They differ only in how they physically represent the data values—whether they use areas, bars, columns, or lines to represent the values. All other properties are the same. Typically you use area and line graphs to display continuous data and use bar and column graphs to display noncontinuous data.

The only difference between a bar graph and a column graph is the orientation: in column graphs, values are plotted along the y axis and categories are plotted along the x axis. In bar graphs, values are plotted along the x axis and categories are plotted along the y axis.

Pie graphs

Pie graphs typically show one series of data points with each data point shown as a percentage of a whole. The following pie graph shows the sales for Stellar printers for each quarter. You can easily see the relative values in each quarter. (PowerBuilder automatically calculates the percentages of each slice of the pie.)
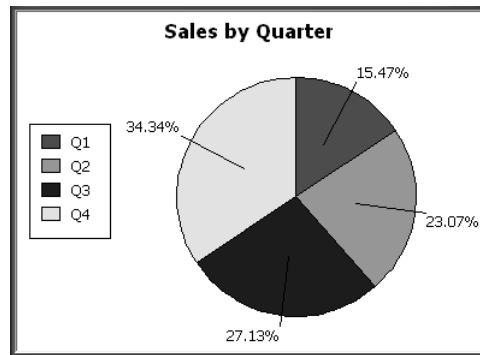
You can have pie graphs with more than one series if you want; the series are shown in concentric circles. Multiseries pie graphs can be useful in comparing series of data.

Scatter graphs

Scatter graphs show xy data points. Typically you use scatter graphs to show the relationship between two sets of numeric values. Non-numeric values, such as string and DateTime datatypes, do not display correctly.

Scatter graphs do not use categories. Instead, numeric values are plotted along both axes—as opposed to other graphs, which have values along one axis and categories along the other axis.

For example, the following data shows the effect of speed on the mileage of a sedan:

Speed	Mileage
10	12
20	18
30	21
40	23
50	26
60	26
70	24
80	20

Here is the data in a scatter graph:
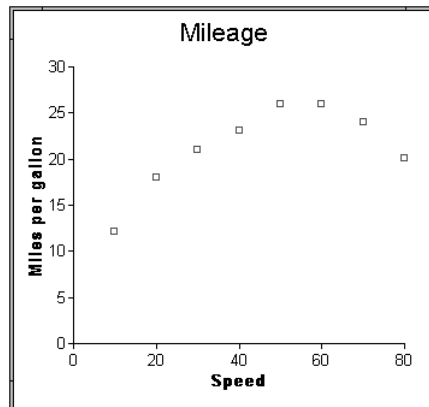
You can have multiple series of data in a scatter graph. You might want to plot mileage versus speed for several makes of cars in the same graph.

Three-dimensional graphs

You can also create 3-dimensional (3D) graphs of area, bar, column, line, and pie graphs. In 3D graphs (except for 3D pie graphs), series are plotted along a third axis (the Series axis) instead of along the Category axis. You can specify the perspective to use to show the third dimension:
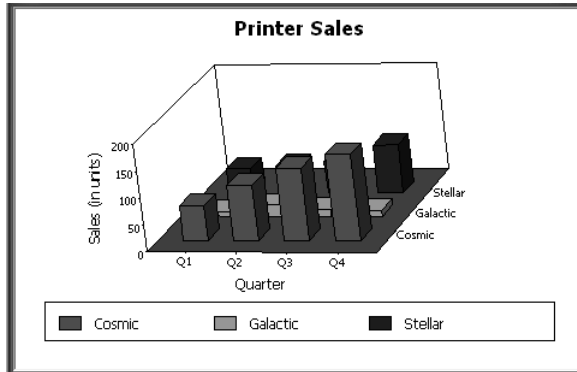
Stacked graphs

In bar and column graphs, you can choose to stack the bars and columns. In stacked graphs, each category is represented as one bar or column instead of as separate bars or columns for each series:
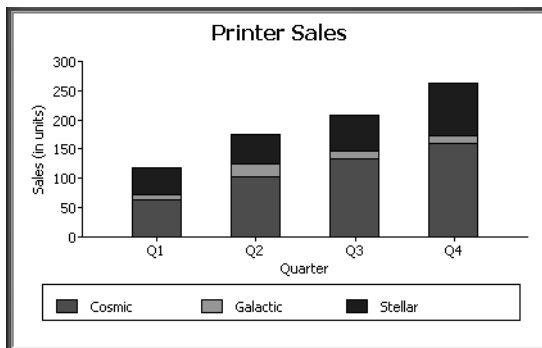
Using graphs in applications

You can use graphs in DataWindow objects and in windows. You specify the properties of a graph, such as its type and title, the same way in a DataWindow object as in a window.

Using graphs in user objects
You can also use graphs in user objects. Everything in this chapter about using graphs in windows also applies to using graphs in user objects.

The major differences between using a graph in a DataWindow object and using a graph in a window (or user object) are:

- Specifying the data for the graph

  In DataWindow objects, you associate columns in the database with the axes of a graph. In windows, you write scripts containing PowerScript functions to populate a graph.

- Specifying the location of the graph

  In DataWindow objects, you can place a graph in the foreground and allow users to move and resize the graph at runtime, or you can place a graph in a band and prevent movement. In windows, graphs are placed like all other window controls.

Using graphs in DataWindow objects

Graphs are used most often in DataWindow objects—the data for the graph comes from tables in the database.

Graphs in DataWindow objects are dynamic

Graphs in DataWindow objects are tied directly to the data that is in the DataWindow object. As the data changes, the graph is automatically updated to reflect the new values.

Two techniques

You can use graphs in DataWindow objects in two ways:

- By including a graph as a control in a DataWindow object

  The graph enhances the display of information in a DataWindow object, such as a tabular or freeform DataWindow object. This technique is described in "Placing a graph in a DataWindow object" next.

- By using the Graph presentation style

  The entire DataWindow object is a graph. The underlying data is not visible. This technique is described in "Using the Graph presentation style" on page 725.

Placing a graph in a DataWindow object

❖ **To place a graph in a DataWindow object:**

1 Open or create the DataWindow object that will contain the graph.

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

3 Click where you want the graph.

  PowerBuilder displays the Graph Data dialog box:
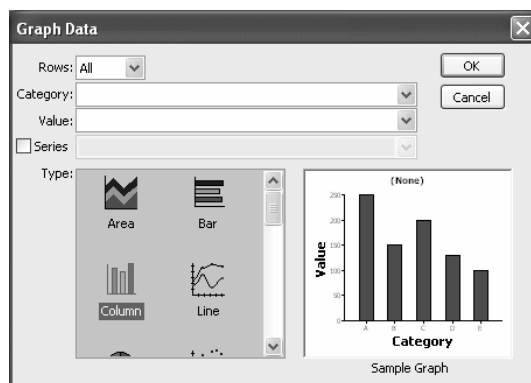
4 Specify which columns contain the data and the type of graph you want, and click OK.

  For more information, see "Associating data with a graph" on page 715.

  The Design view now contains a representation of the graph:
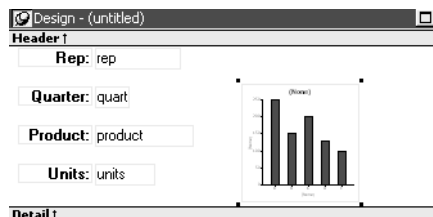
5 Specify the graph's properties in the Properties view.

Using the graph's Properties view

A graph has a Properties view in which you can specify the data as well as the other properties of the graph.

❖ **To display the graph's Properties view:**

• Select Properties from the graph's pop-up menu.

The Properties view for a graph has seven property pages in which you specify information about the graph. Table 26-3 lists the property pages and describes what each property page specifies.

Table 26-3: Property pages for graphs

Property page	What it specifies
Axis	Labels, scale, information about major and minor divisions for the category axes.
Data	Where to get the graph's data.
General	Various general graph properties, including border, graph colors, whether to size the graph to the full screen display, suppression in newspaper columns.
	Graph type, title, legend location.
	For 3D graphs, perspective, rotation, and elevation.
	For bar graphs, overlap, spacing and depth of bars.
Pointer	The pointer to use when the mouse is positioned over the graph.
Position	The x,y location of the upper left corner of the graph, its width and height, sliding options, the layer in which the graph is to be positioned.
	Whether the graph can be resized and moved at runtime.
Text	Text properties for text controls that display on the graph, including title, axis text, axis label, and legend.
	Text properties include font, font style, font size, alignment, rotation, color, display expression, display format.
Other	Descriptions and label for use by assistive technology tools.

Changing a graph's position and size

When you first place a graph in a DataWindow object, it is in the foreground—it sits above the bands in the DataWindow object. Unless you change this setting, the graph displays in front of any retrieved data.

The initial graph is also moveable and resizable, so users have complete flexibility as to the size and location of a graph at runtime. You can change these properties.

❖ **To specify a graph's position and size:**

1   Select Properties from the graph's pop-up menu and then select the Position page or the General page in the Properties view.

2   Select the settings for the following options on the Position property page:

Table 26-4: Settings on the Position property page for graphs

Setting	Meaning
Layer	*Background* — The graph displays behind other elements in the DataWindow object.
	Band — The graph displays in one particular band. If you choose this setting, you should resize the band to fit the graph. Often you will want to place a graph in the Footer band. As users scroll through rows in the DataWindow object, the graph remains at the bottom of the screen as part of the footer.
	Foreground — (Default) The graph displays above all other elements in the DataWindow object. Typically, if you choose this setting, you also make the graph movable so it will not obscure data while users display the DataWindow object.
Moveable	The graph can be moved in the Preview view and at runtime.
Resizable	The graph can be resized in the Preview view and at runtime.
Slide Left, Slide Up	The graph slides to the left or up to remove extra white space. For more information, see "Sliding controls to remove blank space in a DataWindow object" on page 579.
X, Y	The location of the upper-left corner of the graph.
Width, Height	The width and height of the graph.

3   Select the settings for the following options on the General property page:

Table 26-5: Size and position settings on the General property page

Setting	Meaning
Size To Display	The graph fills the DataWindow object and resizes when users resize the DataWindow object. This setting is used with the Graph presentation style.
HideSnaked	Do not repeat graph after the first column in a DataWindow object using newspaper-style columns.

Associating data with a graph

When using a graph in a DataWindow object, you associate axes of the graph with columns in the DataWindow object.

The only way to get data into a graph in a DataWindow object is through columns in the DataWindow object. You cannot add, modify, or delete data in the graph except by adding, modifying, or deleting data in the DataWindow object.

You can graph data from any columns retrieved into the DataWindow object. The columns do not have to be displayed.

About the examples

The process of specifying data for a graph is illustrated below using the Printer table in the EAS Demo DB.

❖   **To specify data for a graph:**

1   If you are creating a new graph, the Graph Data dialog box displays. Otherwise, select Properties from the graph's pop-up menu and select the Data tab in the Properties view.

2   Fill in the boxes as described in the sections that follow, and click OK.

Specifying which rows to include in a graph

The Rows drop-down list allows you to specify which rows of data are graphed at any one time:

Table 26-6: Specifying which rows to include in a graph

Setting	Meaning
All	Graphs the data from all the rows that have been retrieved but not filtered or deleted (that is, the rows in the primary buffer of the DataWindow object)
Page	Graphs only the data from the rows that are currently displayed on the page
Group n	Graphs only the data in the specified group (in a grouped DataWindow object)

Specifying the categories

Specify the column or expression whose values determine the categories. In the Graph Data page in the Graph dialog box and on the Data page in the Properties view, you can select a column name from a drop-down list. In the Data tab, you can also type an expression:
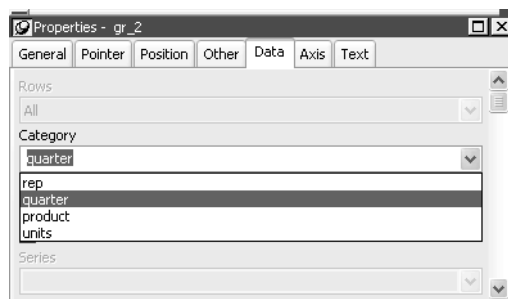
There is an entry along the Category axis for each different value of the column or expression you specify.

    LookupDisplay (*column*)

For more about code tables, see "Defining a code table" on page 622. For more about LookupDisplay, see the *DataWindow Reference*.

Specifying the values

PowerBuilder populates the Value drop-down list. The list includes the names of all the retrieved columns as well as the following aggregate functions:

• Count for all non-numeric columns

• Sum for all numeric columns

Select an item from the drop-down list or type an expression (in the Properties view). For example, if you want to graph the sum of units sold, you can specify:

```
sum(units for graph)
```

To graph 110 percent of the sum of units sold, you can specify:

```
sum(units*1.1 for graph)
```

Specifying the series

Graphs can have one or more series.

Single-series graphs   If you want only one series (that is, if you want to graph all retrieved rows as one series of values), leave the Series box empty.

Multiple-series graphs   If you want to graph more than one series, select the Series check box and specify the column that will provide the series values.

You can select column names from the drop-down list:
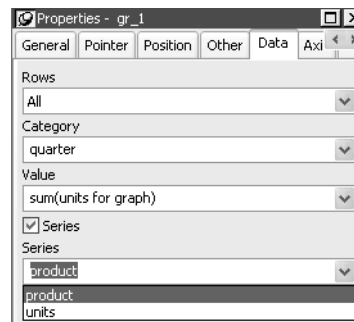
There is a set of data points for each different value of the column you specify here. For example, if you specify in the Series box a column that has 10 values, then your graph will have 10 series: one set of data points for each different value of the column.

| Using expressions | You can also specify expressions for Series (on the Data page of the Properties view). For example, you could specify the following for Series: |

```
Units / 1000
```

In this case, if a table had unit values of 10,000, 20,000, and 30,000, the graph would show series values of 10, 20, and 30.

| Specifying multiple entries | You can specify more than one of the retrieved columns to serve as series. Separate multiple entries by commas. |

You must specify the same number of entries in the Value box as you do in the Series box. The first value in the Value box corresponds to the first series identified in the Series box, the second value corresponds to the second series, and so on. The example about graphing actual and projected sales in "Examples" on page 718 illustrates this technique.

Examples

This section shows how to specify the data for several different graphs of the data in the Printer table in the EAS Demo DB. The table records quarterly unit sales of three printers by three sales representatives.

Table 26-7: The Printer table in the EAS Demo DB

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
Jones	Q1	Galactic	2
Perez	Q1	Galactic	1
…	…	…	…
Simpson	Q4	Stellar	30
Jones	Q4	Stellar	24
Perez	Q4	Stellar	36
Simpson	Q4	Cosmic	60
Jones	Q4	Cosmic	52
Perez	Q4	Cosmic	48
Simpson	Q4	Galactic	3

Rep	Quarter	Product	Units
Jones	Q4	Galactic	3
Perez	Q4	Galactic	6

Graphing total sales

To graph total sales of printers in each quarter, retrieve all the columns into a DataWindow object and create a graph with the following settings on the Data page in the Properties view:

- Set Rows to `All`

- Set Category to `quarter`

- Set Value to `sum(units for graph)`

Leave the Series check box and text box empty.

The Quarter column serves as the category. Because the Quarter column has four values (Q1, Q2, Q3, and Q4), there will be four categories along the Category axis. You want only one series (total sales in each quarter), so you can leave the Series box empty, or type a string literal to identify the series in a legend. Setting Value to `sum(units for graph)` graphs total sales in each quarter.

Here is the resulting column graph. PowerBuilder automatically generates the category text based on the data in the table:
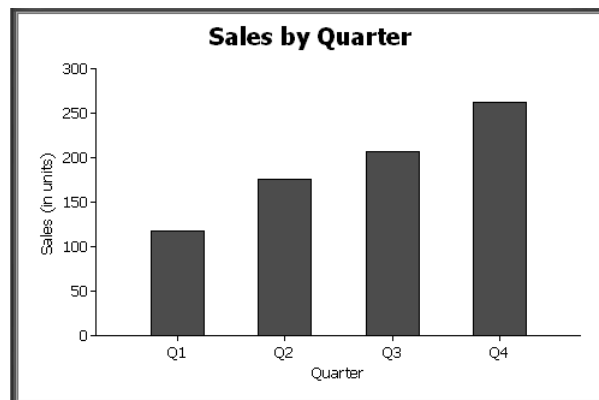
In the preceding graph, there is one set of data points (one series) across four quarters (the category values).

The following is a pie graph, which has exactly the same properties as the preceding column graph except for the type, which is 3D Pie:
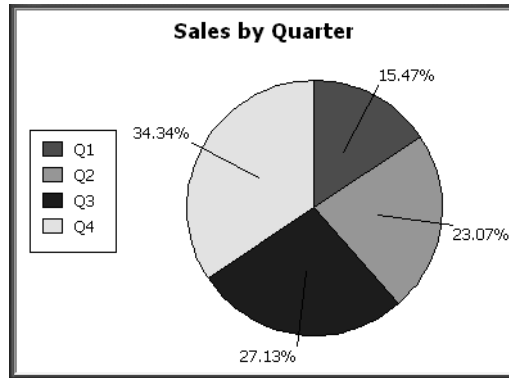
In pie graphs, categories are shown in the legend.

Graphing unit sales of each printer

To graph total quarterly sales of each printer, retrieve all the columns into a DataWindow object and create a graph with the following settings on the Data page in the Properties view:

- Set Rows to `All`
- Set Category to `quarter`
- Set Value to `sum(units for graph)`
- Select the Series check box
- Set Series to `product`

You want a different series for each printer, so the column Product serves as the series. Because the Product column has three values (Cosmic, Galactic, and Stellar), there will be three series in the graph. As in the first example, you want a value for each quarter, so the Quarter column serves as the category, and you want to graph total sales in each quarter, so the Value box is specified as `sum(units for graph)`.

Here is the resulting graph. PowerBuilder automatically generates the category and series labels based on the data in the table. The series labels display in the graph's legend:
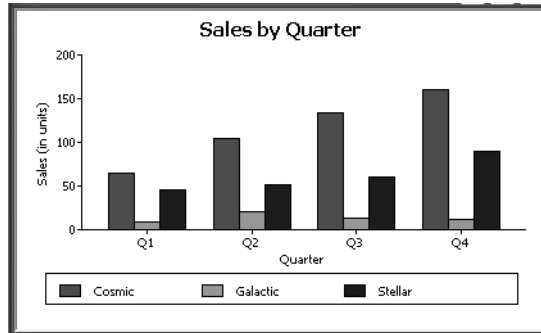
Graphing unit sales by representative

To graph quarterly sales made by each representative, create a graph with the following settings on the Data page in the Properties view:

• Set Rows to `All`

• Set Category to `quarter`

• Set Value to `sum(units for graph)`

• Select the Series check box

• Set Series to `rep`

Here is the resulting graph:
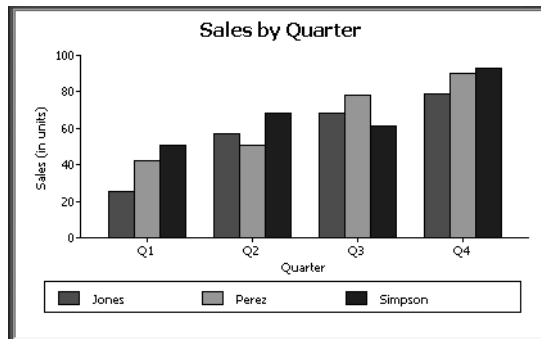
Graphing unit sales by representative and total sales

To graph quarterly sales made by each representative, plus total sales for each printer, create a graph with the following settings on the Data page in the Properties view:

- Set Rows to `All`

- Set Category to `quarter, "Total"`

- Set Value to `sum(units for graph), sum(units for graph)`

- Select the Series check box

- Set Series to `rep, rep`
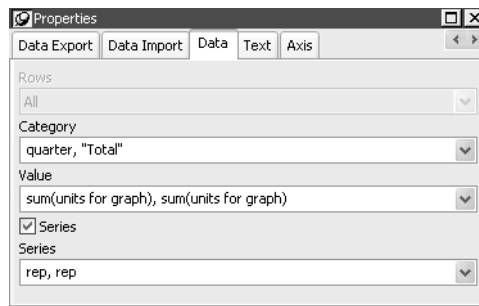
Here you have two types of categories: the first is Quarter, which shows quarterly sales, as in the previous graph. You also want a category for total sales. There is no corresponding column in the DataWindow object, so you can simply type the literal "Total" to identify the category. You separate multiple entries with a comma.

For each of these category types, you want to graph the sum of units sold for each representative, so the Value and Series values are repeated.

Here is the resulting graph:
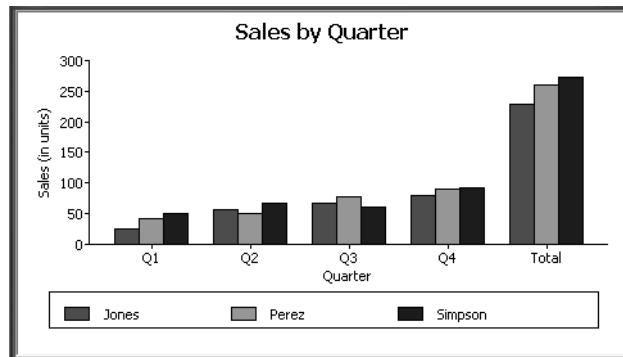
Notice that PowerBuilder uses the literal "Total" supplied in the Category box in the Graph Data window as a value in the Category axis.

Graphing actual and projected sales

To graph total quarterly sales of all printers and projected sales for next year, create a graph with the following settings on the Data page in the Properties view (you assume that sales will increase by 10% next year):

- Set Rows to `All`

- Set Category to `quarter`

- Set Value to `sum(units for graph), sum(units*1.1 for graph)`

- Select the Series check box

- Set Series to `'Actual','Projected'`

You are using labels to identify two series, Actual and Projected. Note the single quotation marks around the literals. For Values, you enter the expressions that correspond to Actual and Projected sales. For Actual, you use the same expression as in the examples above, `sum(units for graph)`. For Projected sales, you multiply each unit sale by 1.1 to get the 10 percent increase. Therefore, the second expression is `sum(units*1.1 for graph)`.

Here is the resulting graph. PowerBuilder uses the literals you typed for the series as the series labels in the legend:
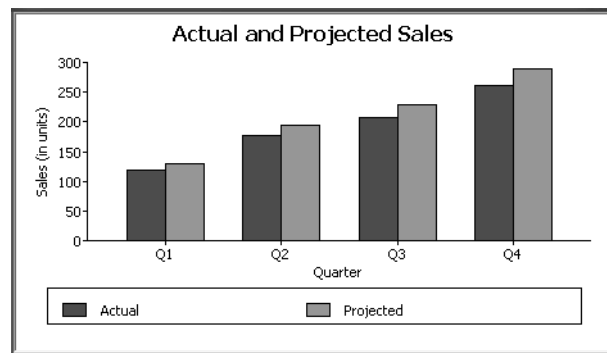
Using overlays

It is often useful to call special attention to one of the series in a graph, particularly in a bar or column graph. You can do that by defining the series as an overlay. An overlay series is graphed as a line on top of the other series in the graph. To define a series as an overlay, define it as follows:

•   If specifying a column name to identify the series, specify this for the series:

> "@overlay~t" + *ColumnName*

•   If using a label to identify the series, specify this for the series:

> "@overlay~t*SeriesLabel*"

Examples

To graph sales in each quarter and overlay the sales of each individual printer, specify the graph's data as in "Graphing unit sales of each printer" on page 720, but use the following expression in the Series box:

```
"Total Sales", "@overlay~t" + product
```

Here is the resulting graph:
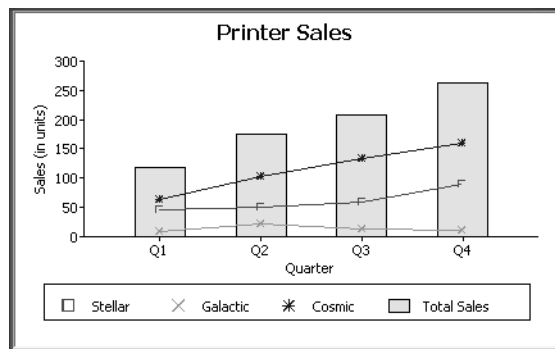
To graph unit sales of printers by quarter and overlay the largest sale made in each quarter, change the Value expression to this:

```
sum(units for graph), max(units for graph)
```

Change the Series expression to this:

```
"Total Sales", "@overlay~tLargets Sale"
```

Here is the resulting graph:
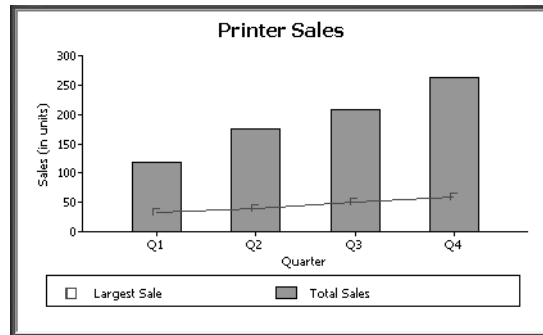
Using the Graph presentation style

Instead of embedding a graph in a DataWindow object, you can use the Graph presentation style to create a DataWindow object that is only a graph—the underlying data is not displayed.

One advantage of the Graph presentation style is that the graph resizes automatically if users resize the DataWindow control associated with the graph DataWindow object at runtime.

❖ **To use the Graph presentation style:**

1   Select File>New from the menu bar.

The New dialog box displays.

2   Select the DataWindow tab, select the Graph presentation style, and click OK.

3   Specify the data source for the DataWindow object. If you want data to be retrieved into the Preview view automatically, select the Retrieve on Preview check box.

For more information, see Chapter 18, "Defining DataWindow Objects."

4   Enter the definitions for the series, categories, and values, as described in "Associating data with a graph" on page 715, and click Next.

Note that when using the Graph presentation style, the graph always graphs all rows; you cannot specify page or group.

5   Enter a title for the graph, select a graph type, and click Next.

6   Review your specifications and click Finish.

A model of the graph displays in the Design view.

7   Specify the properties of the graph, as described in "Defining a graph's properties" next.

8   Save the DataWindow object in a library.

9   Associate the graph DataWindow object with a DataWindow control on a window or user object.

At runtime, the graph fills the entire control and resizes when the control is resized.

Defining a graph's properties

This section describes properties of a graph that are used regardless of whether the graph is in a DataWindow object or in a window. To define the properties of a graph, you use the graph's Properties view. For general information about the property pages, see "Using the graph's Properties view" on page 713.

Using the General property page in the graph's Properties view

You name a graph and define its basic properties on the General property page in the graph's Properties view.

❖   **To specify the basic properties of a graph:**

•   Select Properties from the graph's pop-up menu and then select the General page in the Properties view.

About the model graph in the Design view

As you modify a graph's properties, PowerBuilder updates the model graph shown in the Design view so that you can get an idea of the graph's basic layout:

•   PowerBuilder uses the graph title and axis labels you specify.

•   PowerBuilder uses sample data (not data from your DataWindow object) to illustrate series, categories, and values.

In Preview view, PowerBuilder displays the graph with data.

Naming a graph

You can modify graphs at runtime. To reference a graph in code, you use its name.

❖ **To name a graph:**

- On the General properties page for the graph, assign a meaningful name to the graph in the Name box.

Defining a graph's title

The title displays at the top of the graph.

❖ **To specify a graph's title:**

- On the General properties page for the graph, enter a title in the Title box.

Multiline titles
You can force a new line in a title by embedding ~n.

For information about specifying properties for the title text, see "Specifying text properties for titles, labels, axes, and legends" on page 728.

Specifying the type of graph

You can change the graph type at any time in the development environment. (To change the type at runtime, modify a graph's GraphType property.)

❖ **To specify the graph type:**

- On the General properties page for the graph, select a graph type from the Graph Type drop-down list.

Using legends

A legend provides a key to your graph's series.

❖ **To include a legend for a series in a graph:**

- On the General properties page for the graph, specify where you want the legend to appear by selecting a value in the Legend drop-down list.

For information on specifying text properties for the legend, see "Specifying text properties for titles, labels, axes, and legends" on page 728.

Specifying a border

For a graph control in a DataWindow object, you can specify the border that PowerBuilder places around a graph.

❖ **To specify a border for a graph:**

- On the General properties page for the graph, select the type of border to use from the Border drop-down list.

Specifying point of
view in 3D graphs

If you are defining a 3D graph, you can specify the point of view that
PowerBuilder uses when displaying the graph.

❖ **To specify a 3D graph's point of view:**

1   On the General properties page for the graph, adjust the point of view
    along the three dimensions of the graph:

    •   To change the perspective, move the Perspective slider.

    •   To rotate the graph, move the Rotation slider.

    •   To change the elevation, move the Elevation slider.

2   Define the depth of the graph (the percent the depth is of the width of the
    graph) by using the Depth slider.

Sorting data for series and categories

You can specify how to sort the data for series and categories. By default, the
data is sorted in ascending order.

❖ **To specify how to sort the data for series and categories in a graph:**

1   Select Properties from the graph's pop-up menu and then select the Axis
    page in the Properties view.

2   Select the axis for which you want to specify sorting.

3   Scroll to Sort, the last option on the Axis page, and select Ascending
    (order), Descending (order), or Unsorted.

Specifying text properties for titles, labels, axes, and legends

A graph can have four text elements:

Title
Labels for the axes
Text that shows the values along the axes
Legend
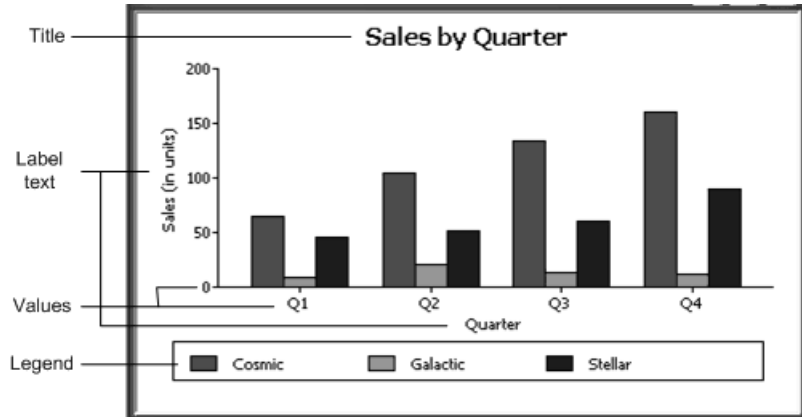
You can specify properties for each text element.

❖ **To specify text properties for the title, labels, axis values, and legend of a graph:**

1   Select Properties from the graph's pop-up menu and then select the Text page in the Properties view.

2   Select a text element from the list in the Text Object drop-down list:
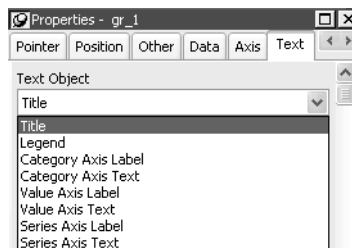
3   Specify the font and its characteristics.

Using Auto Size   With Auto Size in effect, PowerBuilder resizes the text appropriately whenever the graph is resized. With Auto Size disabled, you specify the font size of a text element explicitly.

❖ **To have PowerBuilder automatically size a text element in a graph:**

1   On the Text properties page for the graph, select a text element from the list in the Text Object drop-down list.

2   Select the Autosize check box (this is the default).

❖ **To specify a font size for a text element in a graph:**

1 On the Text properties page for the graph, select a text element from the list in the Text Object drop-down list.

2 Clear the Autosize check box.

3 Select the Font size in the Size drop-down list.

Rotating text

For all the text elements, you can specify the number of degrees by which you want to rotate the text.

❖ **To specify rotation for a text element in a graph:**

1 On the Text properties page for the graph, select a text element from the list in the Text Object drop-down list.

2 Specify the rotation you want in the Escapement box using tenths of a degree (450 means 45 degrees).

Changes you make here are shown in the model graph in the Design view and in the Preview view.

Using display formats

❖ **To use a display format for a text element in a graph:**

1 On the Text properties page for the graph, select a text element from the list in the Text Object drop-down list.

2 Type a display format in the Format box or choose one from the pop-up menu. To display the pop-up menu, click the button to the right of the Format box.

Modifying display expressions

You can specify an expression for the text that is used for each graph element. The expression is evaluated at execution time.

❖ **To specify an expression for a text element in a graph:**

1 On the Text properties page for the graph, select a text element from the list in the Text Object drop-down list.

2 Click the button next to the Display Expression box.

The Modify Expression dialog box displays.

3 Specify the expression.

You can paste functions, column names, and operators. Included with column names in the Columns box are statistics about the columns, such as counts and sums.

4 Click OK to return to the graph's Properties view.

Example   By default, when you generate a pie graph, PowerBuilder puts the title at the top and labels each slice of the pie with the percentage each slice represents of the whole. Percentages are accurate to two decimal places.

The following graph has been enhanced as follows:

- The current date displays in the title

- The percentages are rounded to integers

- The raw data for each slice is shown in addition to the percentages
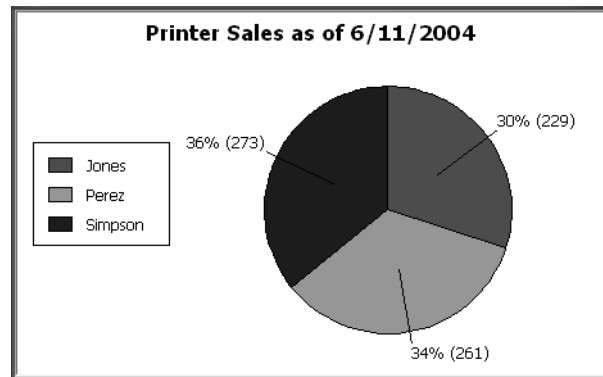
To accomplish this, the display expressions were modified for the title and pie graph labels:

Element	Original expression	Modified expression
Title	`title`	`title + " as of " + date(today())`
Pie graph labels	`if(seriescount > 1, series, string (percentofseries, "0.00%"))`	`if(seriescount > 1, series, string(percentofseries,"0%") + " (" + value + ")" )`

Specifying overlap and spacing

With bar and column charts, you can specify the properties in Table 26-8.

Table 26-8: Overlap and spacing properties for bar and column charts

Property	Meaning
Overlap	The percentage by which bars or columns overlap each other. The default is 0 percent, meaning no overlap.
Spacing	The amount of space to leave between bars or columns. The default is 100 percent, which leaves a space equal to the width of a bar or column.

❖ **To specify overlap and spacing for the bars or columns in a graph:**

1   Select Properties from the graph's pop-up menu and then select the Graph tab.

2   Specify a percentage for Overlap (% of width) and Spacing (% of width).

Specifying axis properties

Graphs have two or three axes. You specify the axes' properties in the Axis page in the graph's Properties view.

❖ **To specify properties for an axis of a graph:**

1   Select Properties from the graph's pop-up menu and then select the Axis page in the Properties view.

2   Select the Category, the Value, or the Series axis from the Axis drop-down list.

If you are not working with a 3D graph, the Series Axis options are disabled.

3   Specify the properties as described next.

Specifying text properties

You can specify the characteristics of the text that displays for each axis. Table 26-9 shows the two kinds of text associated with an axis.

Table 26-9: Text types associated with each axis of a graph

Type of text	Meaning
Text	Text that identifies the values for an axis.
Label	Text that describes the axis. You specify the label text in a painter. You can use ~n to embed a new line within a label.

For information on specifying properties for the text, see "Specifying text properties for titles, labels, axes, and legends" on page 728.

Specifying datatypes

The data graphed along the Value, Category, and Series axes has an assigned datatype. The Series axis always has the datatype String. The Value and Category axes can have the datatypes listed in Table 26-10.

Table 26-10: Datatypes for Value and Category axes

Axis	Possible datatypes
Both axes (for scatter graph)	Number, Date, Time
Value (other graph types)	Number, Date, DateTime, Time
Category (other graph types)	String, Number, Date, DateTime, Time

For graphs in DataWindow objects, PowerBuilder automatically assigns the datatypes based on the datatype of the corresponding column; you do not specify them.

For graphs in windows, you specify the datatypes yourself. Be sure you specify the appropriate datatypes so that when you populate the graph (using the AddData method), the data matches the datatype.

Scaling axes

You can specify the properties listed in Table 26-11 to define the scaling used along numeric axes.

Table 26-11: Properties for scaling on numeric axes

Property	Meaning
Autoscale	If selected (the default), PowerBuilder automatically assigns a scaling for the numbers along the axis.
Round To, Round To Unit	Specifies how to round the end points of the axis (note that this just rounds the range displayed along the axis; it does not round the data itself).
	You can specify a number and a unit. The unit is based on the datatype; you can specify Default as the unit to have PowerBuilder decide for you. For example, if the Value axis is a Date column, you can specify that you want to round the end points of the axis to the nearest five years. In this case, if the largest data value is the year 1993, the axis extends up to 1995, which is 1993 rounded to the next highest five-year interval.
Minimum Value, Maximum Value	The smallest and largest numbers to appear on the axis (disabled if you have selected Autoscale).
Scale Type	Specifies linear or logarithmic scaling (common or natural).
Scale Value	Specifies whether values are displayed as actual values or as a cumulative value, a percentage, or a cumulative percentage.

Using major and minor divisions

You can divide axes into divisions. Each division is identified by a tick mark, which is a short line that intersects an axis. In the Sales by Printer graphs shown in "Examples" on page 718, the graph's Value axis is divided into major divisions of 50 units each. PowerBuilder divides the axes automatically into major divisions.

❖ **To define divisions for an axis of a graph:**

1 To divide an axis into a specific number of major divisions, type the number of divisions you want in the MajorDivisions box.

Leave the number 0 to have PowerBuilder automatically create divisions. PowerBuilder labels each tick mark in major divisions. If you do not want each tick mark labeled, enter a value in the DisplayEveryNLabels box. For example, if you enter 2, PowerBuilder labels every second tick mark for the major divisions.

2 To use minor divisions, which are divisions within each major division, type the appropriate number in the MinorDivisions box. To use no minor divisions, leave the number 0.

When using logarithmic axes
If you want minor divisions, specify 1; otherwise, specify 0.

Representing divisions with grid and drop lines

You can specify lines to represent the divisions as described in Table 26-12 and illustrated in Figure 26-1.

Table 26-12: Representing graph divisions with grid and drop lines

Line	Meaning
Grid line	A line that extends from a tick mark across the graph. Grid lines make graphs easier to read.
Drop line	A line that extends vertically from a data point to its axis (not available for all graph types).

Figure 26-1: Grid and drop lines in a graph
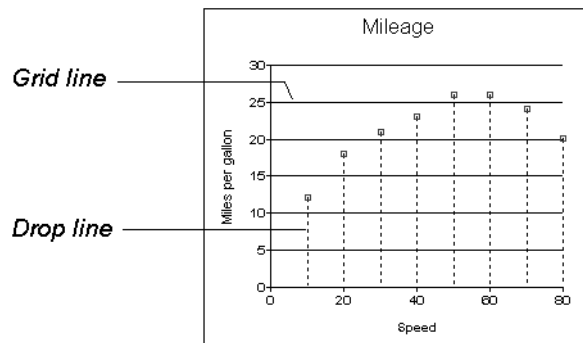
Using line styles

You can define line styles for the components of a graph listed in Table 26-13.

Table 26-13: Components of a graph that can have line styles

Component	Meaning
PrimaryLine	The axis itself
SecondaryLine	The axis parallel to and opposite the primary axis
OriginLine	A grid line that represents the value zero
Frame	The frame for the axis in 3D graphs (disabled for 2D graphs)

Specifying a pointer

You can specify a pointer to use when the mouse is over a graph at runtime.

❖ **To specify a pointer for a graph:**

1   Select Properties from the graph's pop-up menu and then select the Pointer page in the Properties view.

2   Select a stock pointer from the list, or select a *CUR* file containing a pointer.

Using graphs in windows

In addition to using graphs in DataWindow objects, you can also place graphs directly in windows and visual user objects. You define properties for a graph control in the Window painter and use scripts to populate the graph with data and to modify properties for the graph at runtime.

This section describes procedures unique to using graphs in windows and visual user objects. For general graph properties, see "Defining a graph's properties" on page 726.

Placing a graph in a window

This procedure for placing a graph in a window in the Window painter can also be used for placing a graph on a user object in the User Object painter.

❖ **To place a graph in a window:**

1 Open the Window painter and select the window that will contain the graph.

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

3 Click where you want the graph.

PowerBuilder displays a model of the graph in the window.

4 Specify properties for the graph, such as type, title, text properties, and axis properties.

See "Defining a graph's properties" on page 726.

5 Write one or more scripts to populate the graph with data.

See the chapter on manipulating graphs in *Application Techniques*.

Using the graph's Properties view in the Window painter

A graph's Properties view in the Window and User Object painters is similar to the one in the DataWindow painter except that the Properties view in the Window and User Object painter:

• Does not have buttons for specifying property conditional expressions next to properties

• Does not have Data, Position, and Pointer property pages

• Does have an Other page, which you use to specify drag-and-drop, position, and pointer properties for the graph control

For more information, see "Using the graph's Properties view" on page 713.

Working with Crosstabs

About this chapter

This chapter describes how to build crosstabs.

Contents

About crosstabs

Cross tabulation is a useful technique for analyzing data. By presenting data in a spreadsheet-like grid, a crosstab lets users view summary data instead of a long series of rows and columns. For example, in a sales application you might want to summarize the quarterly unit sales of each product.

In PowerBuilder, you create crosstabs by using the Crosstab presentation style. When data is retrieved into the DataWindow object, the crosstab processes all the data and presents the summary information you have defined for it.

An example

Crosstabs are easiest to understand through an example. Consider the Printer table in the EAS Demo DB. It records quarterly unit sales of printers made by sales representatives in one year. (This is the same data used to illustrate graphs in Chapter 26, "Working with Graphs.")

Table 27-1: The Printer table in the EAS Demo DB

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
Jones	Q1	Galactic	2
Perez	Q1	Galactic	1
.	.	.	.
.	.	.	.
.	.	.	.
Simpson	Q4	Stellar	30
Jones	Q4	Stellar	24
Perez	Q4	Stellar	36
Simpson	Q4	Cosmic	60
Jones	Q4	Cosmic	52
Perez	Q4	Cosmic	48
Simpson	Q4	Galactic	3
Jones	Q4	Galactic	3
Perez	Q4	Galactic	6

This information can be summarized in a crosstab. Here is a crosstab that shows unit sales by printer for each quarter:

This report summarizes the unit sales of printers in each quarter.
11/9/98

Sum Of Units	Quarter				
Product	**Q1**	**Q2**	**Q3**	**Q4**	**Grand Total**
Cosmic	64	104	134	160	**462**
Galactic	9	21	13	12	**55**
Stellar	45	51	60	90	**246**
Grand Total	**118**	**176**	**207**	**262**	**763**

The first-quarter sales of Cosmic printers displays in the first data cell. (As you can see from the data in the Printer table shown before the crosstab, in Q1 Simpson sold 33 units, Jones sold 5 units, and Perez sold 26 units—totaling 64 units.) PowerBuilder calculates each of the other data cells the same way.

To create this crosstab, you only have to tell PowerBuilder which database columns contain the raw data for the crosstab, and PowerBuilder does all the data summarization automatically.

What crosstabs do

Crosstabs perform two-dimensional analysis:

- The first dimension is displayed as columns across the crosstab.

  In the preceding crosstab, the first dimension is the quarter, whose values are in the Quarter column in the database table.

- The second dimension is displayed as rows down the crosstab.

  In the preceding crosstab, the second dimension is the type of printer, whose values are in the Product column in the database table.

Each cell in a crosstab is the intersection of a column (the first dimension) and a row (the second dimension). The numbers that appear in the cells are calculations based on both dimensions. In the preceding crosstab, it is the sum of unit sales for the quarter in the corresponding column and printer in the corresponding row.

Crosstabs also include summary statistics. The preceding crosstab totals the sales for each quarter in the last row and the total sales for each printer in the last column.

How crosstabs are implemented in PowerBuilder

Crosstabs in PowerBuilder are implemented as grid DataWindow objects. Because crosstabs are grid DataWindow objects, users can resize and reorder columns at runtime (if you let them).

Import methods return empty result
A crosstab report takes the original result set that was retrieved from the database, sorts it, summarizes it, and generates a new summary result set to fit the design of the crosstab. The ImportFile, ImportClipboard, and ImportString methods can handle only the original result set, and they return an empty result when used with a crosstab report.

Two types of crosstabs

There are two types of crosstabs:

- Dynamic

- Static

Dynamic crosstabs

With dynamic crosstabs, PowerBuilder builds all the columns and rows in the crosstab dynamically when you run the crosstab. The number of columns and rows in the crosstab match the data that exists at runtime.

Using the preceding crosstab as an example, if a new printer was added to the database after the crosstab was saved, there would be an additional row in the crosstab when it is run. Similarly, if one of the quarter's results was deleted from the database after the crosstab was saved, there would be one less column in the crosstab when it is run.

By default, crosstabs you build are dynamic.

Static crosstabs

Static crosstabs are quite different from dynamic crosstabs. With static crosstabs, PowerBuilder establishes the columns in the crosstab based on the data in the database when you *define* the crosstab. (It does this by retrieving data from the database when you initially define the crosstab.) No matter what values are in the database when you later run the crosstab, the crosstab always has the same columns as when you defined it.

Using the preceding crosstab as an example, if there were four quarters in the database when you defined and saved the crosstab, there would always be four columns (Q1, Q2, Q3, and Q4) in the crosstab at runtime, even if the number of columns changed in the database.

Advantages of dynamic crosstabs

Dynamic crosstabs are used more often than static crosstabs, for the following reasons:

- You can define dynamic crosstabs very quickly because no database access is required at definition time.

- Dynamic crosstabs always use the current data to build the columns and rows in the crosstab. Static crosstabs show a snapshot of columns as they were when the crosstab was defined.

- Dynamic crosstabs are easy to modify: all properties for the dynamically built columns are replicated at runtime automatically. With static crosstabs, you must work with one column at a time.

Creating crosstabs

❖ **To create a crosstab:**

1 Select File>New from the menu bar.

   The New dialog box displays.

2 Select the DataWindow tab.

3 Select the Crosstab presentation style, then click OK.

   You are prompted to specify the data.

4 Specify the data you want retrieved into the DataWindow object.

   For more information, see Chapter 18, "Defining DataWindow Objects."

   You are prompted to specify the data for the columns, rows, and cell values in the crosstab.

5 Enter the definitions for the columns, rows, and cell values in the crosstab.

   See "Associating data with a crosstab" on page 742.

6 Click Next.

7 Click Color and Border settings and click Next.

8 Review your specifications and click Finish.

   PowerBuilder creates the crosstab.

9 (Optional) Specify other properties of the crosstab.

   See "Enhancing crosstabs" on page 748.

10 Save the DataWindow object in a library.

Associating data with a crosstab

You associate crosstab columns, rows, and cell values with columns in a database table or other data source.

❖ **To associate data with a crosstab:**

1   If you are defining a new crosstab, the Define Crosstab Rows, Columns, Values dialog box displays after you specify the data source.
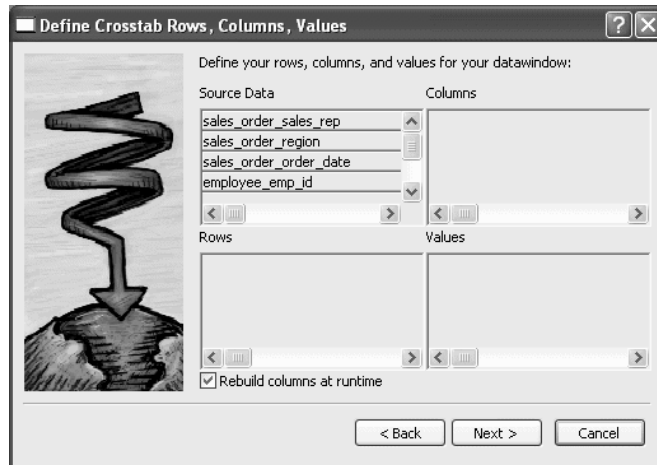
2   Specify the database columns that will populate the columns, rows, and values in the crosstab, as described below.

3   To build a dynamic crosstab, make sure the Rebuild Columns At Runtime box is selected.

For information about static crosstabs, see "Creating static crosstabs" on page 757.

4   Click Next.
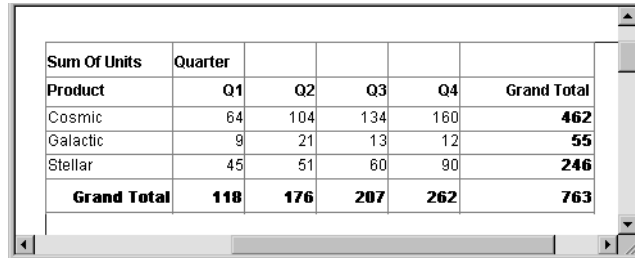
Specifying the information

To define the crosstab, drag the column names from the Source Data box in the Crosstab Definition dialog box (or Wizard page) into the Columns, Rows, or Values box, as appropriate.

If you change your mind or want to edit the DataWindow object later, select Design>Crosstab from the menu bar and drag the column name out of the Columns, Row, or Values box and drop it. Then specify a different column.

Dynamic crosstab example	The process is illustrated using the following dynamic crosstab. The columns in the database are Rep, Quarter, Product, and Units. The crosstab shows the number of printers sold by Quarter:

Sum Of Units	Quarter				
Product	Q1	Q2	Q3	Q4	Grand Total
Cosmic	64	104	134	160	**462**
Galactic	9	21	13	12	**55**
Stellar	45	51	60	90	**246**
Grand Total	**118**	**176**	**207**	**262**	**763**

Specifying the columns	You use the Columns box to specify one or more of the retrieved columns to provide the columns in the crosstab. When users run the crosstab, there is one column in the crosstab for each unique value of the database column(s) you specify here.

❖ **To specify the crosstab's columns:**

• Drag the database column from the Source Data box into the Columns box.

Using the printer example, to create a crosstab where the quarters form the columns, specify Quarter as the Columns value. Because there are four values in the table for Quarter (Q1, Q2, Q3, and Q4), there are four columns in the crosstab.

Specifying the rows	You use the Rows box to specify one or more of the retrieved columns to provide the rows in the crosstab. When users run the crosstab, there is one row in the crosstab for each unique value of the database column(s) you specify here.

❖ **To specify the crosstab's rows:**

• Drag the database column from the Source Data box into the Rows box.

Using the printer example, to create a crosstab where the printers form the rows, specify Product as the Rows value. Because there are three products (Cosmic, Galactic, and Stellar), at runtime there are three rows in the crosstab.

Columns that use code tables
If you specify columns in the database that use code tables, where data is stored with a data value but displayed with more meaningful display values, the crosstab uses the column's display values, not the data values. For more information about code tables, see Chapter 22, "Displaying and Validating Data."

Specifying the values

Each cell in a crosstab holds a value. You specify that value in the Values box. Typically you specify an aggregate function, such as Sum or Avg, to summarize the data. At runtime, each cell has a calculated value based on the function you provide here and the column and row values for the particular cell.

❖ **To specify the crosstab's values:**

1   Drag the database column from the Source Data box into the Values box.

    PowerBuilder displays an aggregate function for the value. If the column is numeric, PowerBuilder uses Sum. If the column is not numeric, PowerBuilder uses Count.

2   If you want to use an aggregate function other than the one suggested by PowerBuilder, double-click the item in the Values box and edit the expression. You can use any of the other aggregate functions supported in the DataWindow painter, such as Max, Min, and Avg.

Using the printer example, you would drag the Units column into the Values box and accept the expression `sum(units for crosstab)`.

Using expressions

Instead of simply specifying database columns, you can use any valid DataWindow expression to define the columns, rows, and values used in the crosstab. You can use any non-object-level DataWindow expression function in the expression.

For example, say a table contains a date column named SaleDate, and you want a column in the crosstab for each month. You could enter the following expression for the Columns definition:

```
Month(SaleDate)
```

The Month function returns the integer value (1–12) for the specified month. Using this expression, you get columns labeled 1 through 12 in the crosstab. Each database row for January sales is evaluated in the column under 1, each database row for February sales is evaluated in the column under 2, and so on.

❖ **To specify an expression for columns, rows, or values:**

1   In the Crosstab Definition dialog box (or wizard page), double-click the item in the Columns, Rows, or Values box.

The Modify Expression dialog box displays.

2   Specify the expression and click OK.

Viewing the crosstab

After you have specified the data for the crosstab's columns, rows, and values, PowerBuilder displays the crosstab definition in the Design view.

For example, to create the dynamic crosstab shown as the "Dynamic crosstab example" on page 743, you would:

1   Drag the quarter column from the Source Data box to the Columns box.

2   Drag the product column from the Source Data box to the Rows box.

3   Drag the units column from the Source Data box to the Values box and accept the expression sum(units for crosstab).

4   Select the Rebuild Columns at Runtime check box.

In the Design view, the crosstab looks like this:

Notice that in the Design view, PowerBuilder shows the quarter entries using the symbolic notation @quarter (with dynamic crosstabs, the actual data values are not known at definition time). @quarter is resolved into the actual data values (in this case, Q1, Q2, Q3, and Q4) when the crosstab runs.

The crosstab is generated with summary statistics: the rows and columns are totaled for you.

At this point, the crosstab looks like this in the Preview view with data retrieved:

- Because quarter was selected as the Columns definition, there is one column in the crosstab for each unique quarter (Q1, Q2, Q3, and Q4).

- Because product was selected as the Rows definition, there is one row in the crosstab for each unique product (Cosmic, Galactic, and Stellar).

- Because sum(units for crosstab) was selected as the Values definition, each cell contains the total unit sales for the corresponding quarter (the Columns definition) and product (the Rows definition).

- PowerBuilder displays the grand totals for each column and row in the crosstab.

Specifying more than one row or column

Typically you specify one database column as the Columns definition and one database column for the Rows definition, as in the printer crosstab. But you can specify as many columns (or expressions) as you want.

For example, consider a crosstab that has the same specification as the crosstab in "Viewing the crosstab" on page 745, except that two database columns, quarter and rep, have been dragged to the Columns box.

PowerBuilder displays this in the Design view:

Sum Of Units	Quarter	Rep		
Header [1] ↑				
	@quarter	@quarter Sum Of Units		
Header [2] ↑				
Product	@rep		Grand Total	
Header [3] ↑				
product	units	crosstabsum(1, 2, "@quar	crosstabsum(1)	
Detail ↑				
"Grand Total"	sum(units	sum(sum_units for all)	sum(grand_sum_units for	
Summary ↑				
Footer ↑				

This is what you see at runtime:

Sum Of Units	Quarter	Rep							
	Q1			Q1 Sum Of Units	Q2				Q2 Sum Of Units
Product	Jones	Perez	Simpson		Jones	Perez	Simpson		
Cosmic	5	26	33	64	36	28	40		104
Galactic	2	1	6	9	6	3	12		21
Stellar	18	15	12	45	15	20	16		51
Grand Total	25	42	51	118	57	51	68		176

For each quarter, the crosstab shows sales of each printer by each sales representative.

Previewing crosstabs

When you have defined the crosstab, you can see it with data in the Preview view.

❖ **To preview the crosstab:**

1 Select View>Preview from the menu bar to display the Preview view.

2 Click on the Preview view to be sure it is current.

3 Select Rows>Retrieve from the menu bar.

PowerBuilder retrieves the rows and performs the cross tabulation on the data.

Retrieve on Preview makes retrieval happen automatically
If the crosstab definition specifies Retrieve on Preview, retrieval happens automatically when the Preview view first displays.

---

4 Continue enhancing your DataWindow object and retrieve again when necessary to see the results of your enhancements.

Enhancing crosstabs

When you have provided the data definitions, the crosstab is functional, but you can enhance it before using it. Because a crosstab is a grid DataWindow object, you can enhance a crosstab using the same techniques you use in other DataWindow objects. For example, you can:

• Sort or filter rows

• Change the column headers

• Specify fonts, colors, mouse pointers, and borders

• Specify column display formats

For more on these and the other standard enhancements you can make to DataWindow objects, see Chapter 19, "Enhancing DataWindow Objects."

The rest of this section covers topics either unique to crosstabs or especially important when working with crosstabs:

- "Specifying basic properties" next

- "Modifying the data associated with the crosstab" on page 750

- "Changing the names used for the columns and rows" on page 750

- "Defining summary statistics" on page 751

- "Cross-tabulating ranges of values" on page 754

- "Creating static crosstabs" on page 757

- "Using property conditional expressions" on page 758

Specifying basic properties

Crosstabs are implemented as grid DataWindow objects, so you can specify the following grid properties for a crosstab:

- When grid lines are displayed

- How users can interact with the crosstab at runtime

❖ **To specify the crosstab's basic properties:**

1   In the Properties view, select the General tab.

2   Specify basic crosstab properties.

Table 27-2 lists basic crosstab properties.

Table 27-2: Basic properties for crosstabs

Option	Result
Display	*On* — Grid lines always display.
	Off — Grid lines never display (columns cannot be resized at runtime).
	Display Only — Grid lines display only when the crosstab displays online.
	Print Only — Grid lines display only when the contents of the crosstab are printed.
Column Moving	Columns can be moved at runtime.
Mouse Selection	Data can be selected at runtime (and, for example, copied to the clipboard).
Row Resize	Rows can be resized at runtime.

Modifying the data associated with the crosstab

When you initially define the crosstab, you associate the crosstab rows and columns with columns in a database table or other data source. You can change the associated data anytime in the Crosstab Definition dialog box.

❖ **To open the Crosstab Definition dialog box:**

1   Position the mouse below the footer band in the workspace and display the pop-up menu.

2   Select Crosstab from the pop-up menu.

The Crosstab Definition dialog box displays.

❖ **To modify the data associated with a crosstab:**

1   In the Crosstab Definition dialog box, fill in the boxes for Columns, Rows, and Values as described in "Associating data with a crosstab" on page 742.

2   Click OK.

Changing the names used for the columns and rows

Sometimes names of columns in the database might not be meaningful. You can change the names that are used to label rows and columns in crosstabs so that the data is easier to understand.

❖ **To change the names used in crosstabs:**
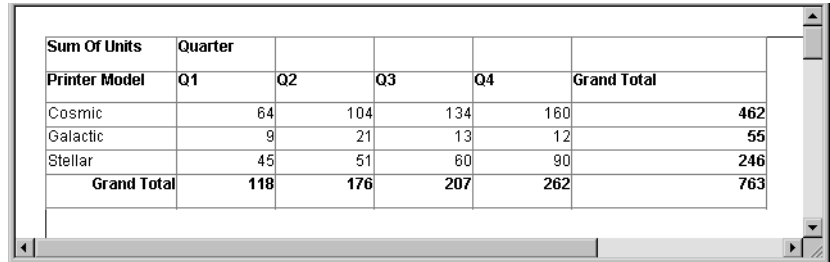
1   In the Crosstab Definition dialog box, double-click the name of the column in the Source Data box.

The New Name dialog box displays.

2   Specify the name you want used to label the corresponding column. You can have multiple-word labels by using underscores: underscores are replaced by spaces in the Design view and at runtime.

3   Click OK.

PowerBuilder changes the column name in the Source Data box and anywhere else the column is used.

Example

For example, if you want the product column to be labeled *Printer Model*, double-click product in the Crosstab Definition dialog box and specify `printer_model` in the New Name dialog box.

When the crosstab runs, you see this:

Sum Of Units	Quarter				
Printer Model	**Q1**	**Q2**	**Q3**	**Q4**	**Grand Total**
Cosmic	64	104	134	160	462
Galactic	9	21	13	12	55
Stellar	45	51	60	90	246
Grand Total	**118**	**176**	**207**	**262**	**763**

Defining summary statistics

When you generate a crosstab, the columns and rows are automatically totaled for you. You can include other statistical summaries in crosstabs as well. To do that, you place computed fields in the workspace.

❖ **To define a column summary:**

1   Enlarge the summary band to make room for the summaries.

2   Select Insert>Control>Computed Field from the menu bar.

3   Click the cell in the summary band where you want the summary to display.

   The Modify Expression dialog box displays.

4   Define the computed field.

   For example, if you want the average value for a column, specify `avg(units for all)`, where units is the column providing the values in the crosstab.

This is a crosstab that has been enhanced to show averages and maximum values for each column. This is the Design view:

This is the crosstab at runtime:

❖ **To define a row summary:**

1 Select Insert>Control>Computed Field from the menu bar.

2 Click the empty cell to the right of the last column in the detail band.

The Modify Expression dialog box displays.

3 Define the computed field. You should use one of the crosstab functions, described next.

Using crosstab functions

There are nine special functions you can use only in crosstabs: CrosstabAvg, CrosstabAvgDec, CrosstabCount, CrosstabMax, CrosstabMaxDec, CrosstabMin, CrosstabMinDec, CrosstabSum, and CrosstabSumDec.

These functions are listed in the Functions box when you define a computed field in a crosstab:
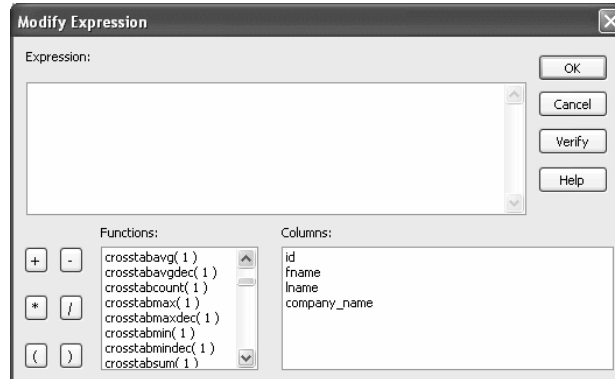
Each of these functions returns the corresponding statistic about a row in the crosstab (average, count, maximum value, minimum value, or sum). You place computed fields using these functions in the detail band in the Design view. Use the functions with the Dec suffix when you want to return a decimal datatype.

By default, PowerBuilder places CrosstabSum and CrosstabSumDec in the detail band, which returns the total for the corresponding row.

How to specify the functions

Each of these functions takes one numeric argument, which refers to the expression defined for Values in the Crosstab Definition dialog box. The first expression for Values is numbered 1, the second is numbered 2, and so on.

Generally, crosstabs have only one expression for Values, so the argument for the crosstab functions is 1. So, for example, if you defined sum(units for crosstab) as your Values expression, PowerBuilder places CrosstabSum(1) in the detail band.

If you want to cross-tabulate both total unit sales and a projection of future sales, assuming a 20 percent increase in sales (that is, sales that are 1.2 times the actual sales), you define two expressions for Values:

```
sum(units for crosstab)
sum(units * 1.2 for crosstab)
```

Here `CrosstabSum(1)` returns the total of `sum(units for crosstab)` for the corresponding row. `CrosstabSum(2)` returns the total for `sum(units * 1.2 for crosstab)`.

For more information

For complete information about defining computed fields, see Chapter 19, "Enhancing DataWindow Objects."

For more about the crosstab functions, see the *DataWindow Reference*.

Cross-tabulating ranges of values

You can build a crosstab where each row tabulates a *range* of values, instead of one discrete value, and you can make each column in the crosstab correspond to a range of values.

For example, in cross-tabulating departmental salary information, you might want one row in the crosstab to count all employees making between \$30,000 and \$40,000, the next row to count all employees making between \$40,000 and \$50,000, and so on.

❖ **To cross-tabulate ranges of values:**

1 Determine the expression that results in the raw values being converted into one of a small set of fixed values.

Each of those values will form a row or column in the crosstab.

2 Specify the expression in the Columns or Rows box in the Crosstab Definition dialog box.

You choose the box depending on whether you want the columns or rows to correspond to the range of values.

3 In the Values column, apply the appropriate aggregate function to the expression.

Example

This is best illustrated with an example.

You want to know how many employees in each department earn between \$30,000 and \$40,000, how many earn between \$40,000 and \$50,000, how many earn between \$50,000 and \$60,000, and so on. To do this, you want a crosstab where each row corresponds to a \$10,000 range of salary.

The first step is to determine the expression that, given a salary, returns the next smaller salary that is a multiple of $10,000. For example, given a salary of $34,000, the expression would return $30,000, and given a salary of $47,000, the expression would return $40,000. You can use the Int function to accomplish this, as follows:

```
int(salary/10000) * 10000
```

That expression divides the salary by 10,000 and takes the integer portion, then multiplies the result by 10,000. So for $34,000, the expression returns $30,000, as follows:

```
34000/10000 = 3.4
int(3.4) = 3
3 * 10000 = 30000
```

With this information you can build the crosstab. The following uses the Employee table in the EAS Demo DB:

1   Build a crosstab and retrieve the dept_id and salary columns.

2   In the Crosstab Definition dialog box, drag the dept_id column to the Columns box.

3   Drag the salary column to the Rows box *and* to the Values box and edit the expressions.

In the Rows box, use:

```
int(salary/10000) * 10000
```

In the Values box, use:

```
count(int(salary/10000) * 10000 for crosstab)
```

For more on providing expressions in a crosstab, see "Using expressions" on page 744.

4   Click OK.

This is the result in the Design view:

Number of employees by department and salary $30,000 includes up to $39,999	Total number of employees making the salary	
Header [1] ↑		
	Department Id	
Salary	display(@	
Header [2] ↑		
row_column	val	crosstabsum(1)
Detail ↑		
Total number of employees in the department	sum(val for	
Summary ↑		
Footer ↑		

This is the crosstab at runtime:

Number of employees by department and salary $30,000 includes up to $39,999						Total number of employees making the salary
			Department Id			
Salary	100	200	300	400	500	
$20,000				2	5	7
$30,000	3	8	2	5	2	20
$40,000	6	5	2	5	1	19
$50,000	4	3	3	2	1	13
$60,000	4	1		2		7
$70,000	2	1	1			4
$80,000	2	1				3
$90,000	1					1
$130,000			1			1
Total number of employees in the department	22	19	9	16	9	

You can see, for example, that 2 people in department 400 and 5 in department 500 earn between $20,000 and $30,000.

Displaying blank values as zero

In the preceding crosstab, several of the cells in the grid are blank. There are no employees in some salary ranges, so the value of those cells is null. To make the crosstab easier to read, you can add a display format to fields that can have null values so that they display a zero.

❖ **To display blank values in a crosstab as zero:**

1 Select the column you want to modify and click the Format tab in the Properties view.

2 Replace [General] in the Format box with `###0;###0;0;0`.

The fourth section in the mask causes a null value to be represented as zero.

Creating static crosstabs

By default, crosstabs are dynamic: when you run them, PowerBuilder retrieves the data and dynamically builds the columns and rows based on the retrieved data. For example, if you define a crosstab that computes sales of printers and a new printer type is entered in the database after you define the crosstab, you want the new printer to be in the crosstab. That is, you want PowerBuilder to build the rows and columns dynamically based on current data, not the data that existed when the crosstab was defined.

Occasionally, however, you might want a crosstab to be static. That is, you want its columns to be established when you define the crosstab. You do not want additional columns to display in the crosstab at runtime; no matter what the data looks like, you do not want the number of columns to change. You want only the updated statistics for the predefined columns. The following procedure shows how to do that.

❖ **To create a static crosstab:**

1 In the wizard page or in the Crosstab Definition dialog box, clear the Rebuild Columns At Runtime check box.

2 Define the data for the crosstab as usual, and click OK.

What happens

With the check box cleared, instead of immediately building the crosstab's structure, PowerBuilder first retrieves the data from the database. Using the retrieved data, PowerBuilder then builds the crosstab structure and displays the workspace. It places all the values for the column specified in the Columns box in the workspace. These values become part of the crosstab's definition.

For example, in the following screenshot, the four values for Quarter (Q1, Q2, Q3, and Q4) are displayed in the Design view:

Sum Of Units	Quarter					
Header [1] ↑						
Product	Q1	Q2	Q3	Q4	Grand Total	
Header [2] ↑						
product	units	units_1	units_2	units_3	**crosstabsum(1)**	
Detail ↑						
"Grand Total"	sum(units	sum(units	sum(units	sum(units	sum(grand_sum_units for	
Summary ↑						
Footer ↑						

At runtime, no matter what values are in the database for the column, the crosstab shows only the values that were specified when the crosstab was defined. In the printer example, the crosstab always has the four columns it had when it was first defined.

Making changes
You can modify the properties of any of the columns in a static crosstab. You can modify the properties of each column individually, since each column is displayed in the workspace as part of the crosstab's definition. For example, in the printer crosstab you can directly modify the way values are presented in each individual quarter, since each quarter is represented in the Design view. (The values are shown as units, units_1, units_2, and units_3.)

Using property conditional expressions

As with other DataWindow objects, you can specify property conditional expressions to modify properties at runtime. You can use them with either dynamic or static crosstabs. With dynamic crosstabs, you specify an expression once for a column or value, and PowerBuilder assigns the appropriate properties when it builds the individual columns at runtime. With static crosstabs, you have to specify an expression for each individual column or value, because the columns are already specified at definition time.

Example

In the following crosstab, an expression has been specified for Units:
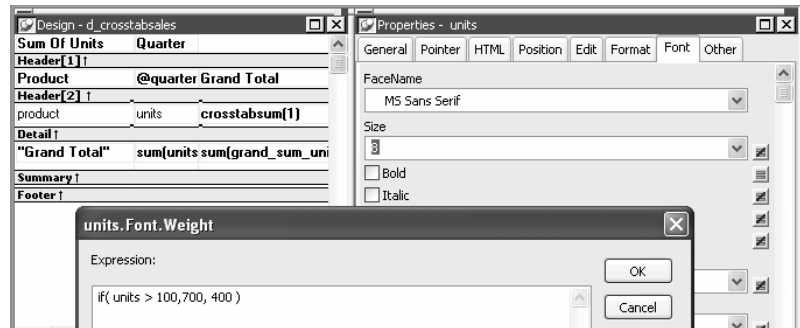
The expression is for the Font.Weight property of the units column:

```
if (units > 100, 700, 400)
```

The expression specifies to use bold font (weight = 700) if the number of units is greater than 100. Otherwise, use normal font (weight = 400).

This is the crosstab at runtime:

Values larger than 100 are shown in bold.

For more information about property conditional expressions, see Chapter 24, "Highlighting Information in DataWindow Objects."

Working with TreeViews

About this chapter

This chapter describes how to build and use DataWindow objects in PowerBuilder using the TreeView presentation style.

Contents

TreeView presentation style

The TreeView presentation style provides an easy way to create DataWindow objects that display hierarchical data in a TreeView, where the rows are divided into groups that can be expanded and collapsed.

The TreeView DataWindow displays a hierarchy of nodes, similar to the way:

• The left pane of Windows Explorer displays folders and files

• The PowerBuilder System Tree displays workspaces and their contents

In the TreeView DataWindow, each parent node contains other nodes called child nodes. You can display parent nodes—nodes that contain child nodes—in expanded or collapsed form.

With the TreeView DataWindow presentation style, you can group data in a hierarchy that allows users to browse the data and expand nodes to view details. Each TreeView level or node has an icon that users can click to expand or collapse the node.

No Web DataWindow support
Currently you cannot use the TreeView DataWindow presentation style with the Web DataWindow.

You use the TreeView DataWindow wizard to create a TreeView DataWindow object. For information, see "Creating a new TreeView DataWindow" on page 763.

Example

This sample TreeView DataWindow uses the department and employee tables in the EAS Demo DB database and has two TreeView levels. The first level is the department name. The second level is the city where each employee resides. The detail data for each employee is grouped in TreeView leaf nodes under these two levels.

Similarities to the Group presentation style

Creating and using a TreeView DataWindow is similar to creating and using a Group DataWindow. However, with the TreeView DataWindow, you can click the state icon to expand and collapse nodes.

The state icon in a TreeView DataWindow is a plus sign (+) when the node is collapsed and a minus sign (-) when the node is expanded. When a node is expanded, connecting lines display by default to show more detail and indicate how the parent data connects with the child data. When a node is collapsed, only the parent data displays; the detail data does not.

Creating a new TreeView DataWindow

You use the TreeView wizard and the DataWindow painter to create a TreeView DataWindow.

TreeView creation process

A TreeView DataWindow has multiple levels, each of which is a node in the TreeView. You use the TreeView wizard to create a TreeView DataWindow, but the wizard produces a DataWindow that includes only the top level of the TreeView.

Creating a complete TreeView DataWindow involves three steps:

1   Using the TreeView DataWindow wizard to create the top level (level 1) of the TreeView DataWindow.

2   Using the DataWindow painter to add additional levels to the TreeView DataWindow.

3   Setting TreeView DataWindow properties to customize the TreeView style.

For information about adding and deleting TreeView levels, see "Adding and deleting TreeView levels" on page 768. For information about setting properties in the DataWindow painter, see "Setting properties for the TreeView DataWindow" on page 772.

You can use TreeView DataWindow methods to expand and collapse TreeView nodes, and you can write code for TreeView DataWindow events that are fired when a node is expanded or collapsed. For detailed information about using TreeView DataWindow properties, methods, and events, see the *DataWindow Reference* or the online Help.

Creating a TreeView DataWindow

❖   **To create a TreeView DataWindow:**

1   Select File>New from the menu bar and select the DataWindow tab.

2   If there is more than one target in the workspace, select the target where you want to create the DataWindow from the drop-down list at the bottom of the dialog box.

3   Choose the TreeView presentation style for the DataWindow and click
OK.
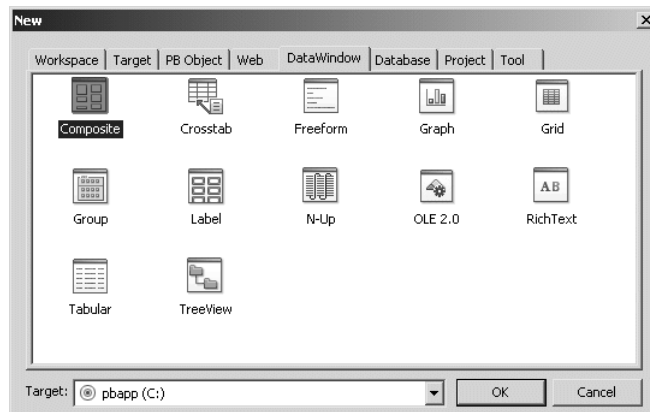
4   Select the data source you want to use: Quick Select, SQL Select, Query,
External, or Stored Procedure.

You are prompted to specify the data.

5   Define the tables and columns you want to use.

You are prompted to specify the TreeView grouping columns.

Multiple columns and multiple TreeView levels
You can specify more than one column, but all columns apply to TreeView
level one. At this point, you can define only one TreeView level. You
define additional levels later.

In the following example, TreeView grouping will be by department, as specified by the dept_id column:
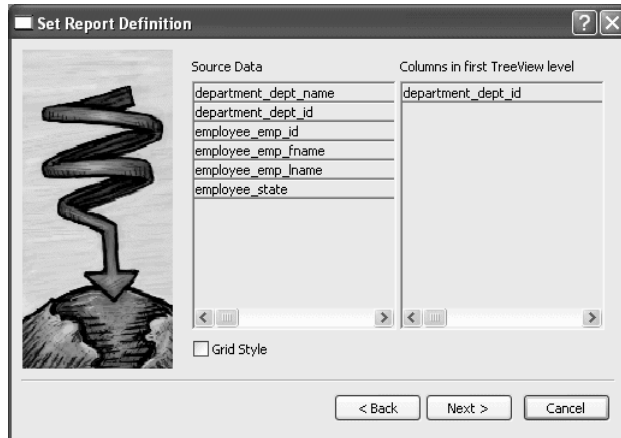
If you want to use an expression, you can define it when you have completed the wizard. See "Using an expression for a column name" on page 767.

The sample DataWindow shown in "Example" on page 762 uses the department and employee tables in the EAS Demo DB database.

6   Specify the column or columns that will be at the top level (level 1) of the TreeView DataWindow.

The sample DataWindow uses the department name as the top level. If you want to display both the department ID and department name, you specify that both columns are at the top level.

7   If you want the TreeView DataWindow to display grid lines, select the Grid Style check box.

When you select the Grid Style check box, the TreeView DataWindow displays grid lines for rows and columns. You can drag the grid lines to resize rows and columns.

8   Click Next.

9   Modify the default color and border settings if needed, and then click Next.

10   Review the TreeView DataWindow characteristics.

11   Click Finish.

The DataWindow painter Design view displays. For information about the Design view, see "TreeView DataWindow Design view" on page 771. For information about adding additional levels, see "Adding and deleting TreeView levels" on page 768.

What PowerBuilder does

As a result of your specifications, PowerBuilder generates a TreeView DataWindow object and creates:

- A TreeView header band with controls that include the heading text of the detail band columns

- The first TreeView level band with the TreeView level columns you chose in the wizard

- The detail (leaf node) band that includes all the column controls except for first-level columns you selected in the wizard

- A level 1 trailer band.

- A summary band, and a footer band.
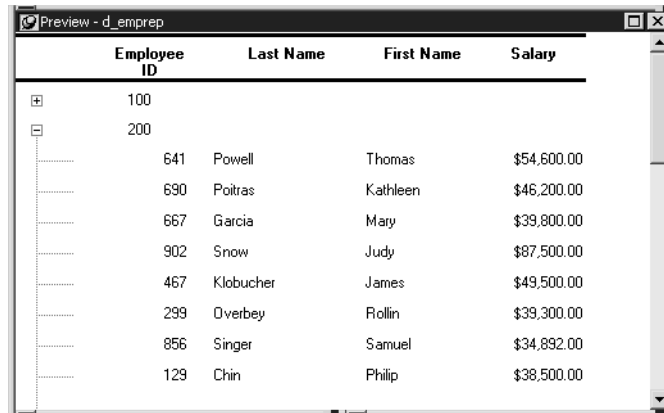
Here is the sample TreeView DataWindow object in the Design view:
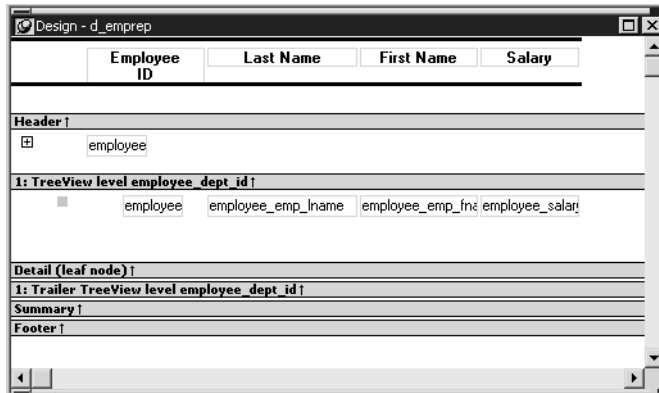
If you selected the Grid Style check box, vertical and horizontal grid lines display:

Here is the sample TreeView DataWindow object in the Preview view:

Using an expression
for a column name
If you want to use an expression for one or more column names in a TreeView, you can enter it as the TreeView definition on the General page in the Properties view after you finish using the TreeView wizard.

❖ **To use an expression for a TreeView column name:**

1 Open the Properties view and click the TreeView level band in the Design view.

2 Click the ellipsis button next to the TreeView Level Definition box on the General page in the Properties view to open the Specify Group Columns dialog box.

3 In the Columns box, double-click the column you want to use in an expression.

The Modify Expression dialog box opens. You can specify more than one grouping item expression for a group. A break occurs whenever the value concatenated from each column/expression changes.

What you can do
All of the techniques available in a tabular DataWindow object , such as moving controls and specifying display formats, are available for modifying and enhancing TreeView DataWindow objects. See "Adding and deleting TreeView levels" next to read more about the bands in a TreeView DataWindow object and see how to add features especially suited for TreeView DataWindow objects, such as additional TreeView levels or summary statistics.

DataWindow object is not updatable by default
When you generate a DataWindow object using the TreeView presentation
style, PowerBuilder makes it not updatable by default. If you want to be able
to update the database through the TreeView DataWindow object, you must
modify its update characteristics. For more information, see Chapter 21,
"Controlling Updates in DataWindow Objects."

Adding and deleting TreeView levels

You add and delete TreeView levels using the Rows menu in the DataWindow
painter.

❖ **To create an additional level in a TreeView DataWindow:**

1   Open the TreeView DataWindow if it is not already open.

2   Select Rows>Create TreeView Level from the menu bar.

     The Specify Group Columns dialog box displays.

3   Specify the columns you want to set as the next TreeView level by
     dragging them from the Source Data pane to the Columns pane.

     In the sample DataWindow shown in "Example" on page 762, the second
     level has a single column, the employee_city column.
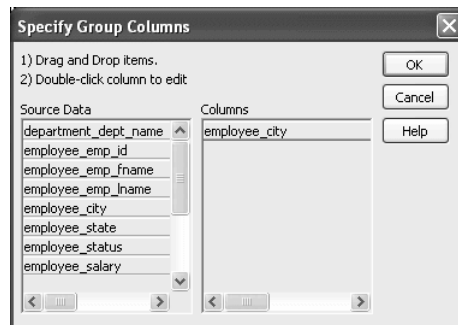
4   Click OK.

     The new TreeView level and a Trailer band for that level are created in the
     TreeView Design view. For information on how to set properties for a
     TreeView level, see "Setting TreeView level properties" on page 774.

❖ **To delete a level in a TreeView DataWindow:**

1   Select Rows>Delete TreeView Level from the menu bar.
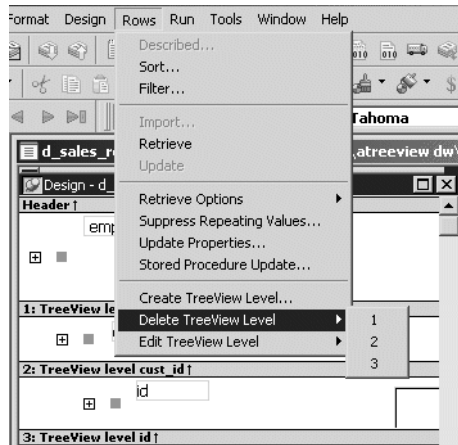
2   Select the number of the level to delete from the list of levels that displays.

The level in the TreeView DataWindow is deleted immediately.

If you delete a level by mistake
If you unintentionally delete a level, close the TreeView DataWindow without saving changes, then reopen it and continue working.

Selecting a tree node and navigating the tree

You can select a tree node in the TreeView DataWindow in the following ways:

•   Use the SelectTreeNode method to select a tree node.

•   Set the Select Node By Mouse property to true and then click a tree node to select it with the mouse.

After you select a tree node in the TreeView DataWindow, you can navigate the tree using the up, down, left, and right keys.

Use this key	To do this
Up	Select a tree node prior to the currently selected node.
Down	Select a tree node next to the currently selected node.

Use this key	To do this
Left	Collapse the currently selected node. If the current tree node is a leaf node or the node has been collapsed, the DataWindow just scrolls to the left, which is its normal behavior.
Right	Expand the currently selected node. If the current tree node is a leaf node or the node has been expanded, the DataWindow just scrolls to the right, which is its normal behavior.

For detailed information about TreeView DataWindow properties, methods and events, see the *DataWindow Reference* or the online Help.

Sorting rows in a TreeView Report

❖ **To sort the rows within levels in a TreeView DataWindow:**

1   Select Rows>Sort from the menu bar.

2   Drag the columns that you want to sort the rows on from the Source Data box to the Columns box.

The order of the columns determines the precedence of the sort. The sort order is ascending by default. To sort in descending order, clear the Ascending check box.

For example, the sample DataWindow shown in "Example" on page 762 has department name as the first level and the employee's city of residence as the second level.
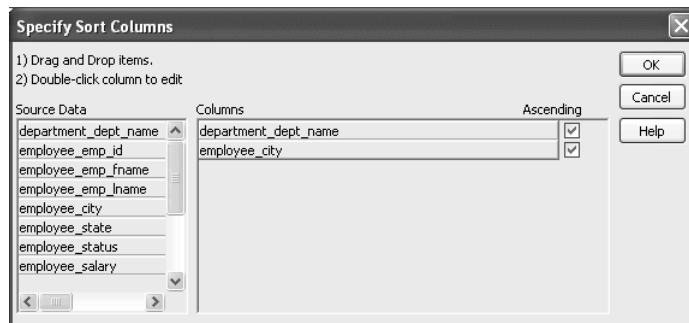
Other actions you can
take

To reorder the columns, drag them up or down in the list. To delete a column
from the sort columns list, drag the column outside the dialog box. To specify
an expression to sort on, double-click a column name in the Columns box and
modify the expression in the Modify Expression dialog box.

TreeView DataWindow Design view

The Design view for the TreeView DataWindow differs from the traditional
Design view for most DataWindow presentation styles.
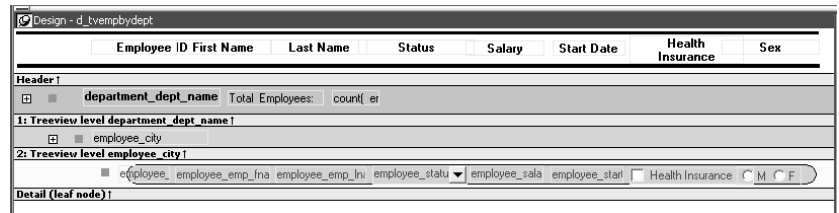
The Design view has a header band, a TreeView level band for each added
level, a detail band, a Trailer band for each level, a summary band, and a footer
band.

By default, the controls in the header band are the heading text of the detail
band columns, and the controls in the detail (leaf node) band are all the column
controls except for the first-level columns (in the 1:Treeview level band) that
you selected when you used the TreeView wizard. Columns that you specify as
additional levels remain in the detail band.

The minimum height of each TreeView level band is the height of the tree node
icon.

Icons in the Design
view

There are three icons in the Design view that represent the locations of nodes,
icons, and connecting lines in the tree to help you design the DataWindow.
Columns must always display to the right of the state and tree node icons:

• A square icon with a plus sign (+) in each TreeView level band represents
the position of the state icon, the icon that indicates whether a node is
expanded or collapsed. On the XP platform, the plus (+) and minus (-)
icons have the Windows XP style.

- A shaded square icon in the detail band and in each TreeView level band represents the position of the image you specify as a tree node icon.
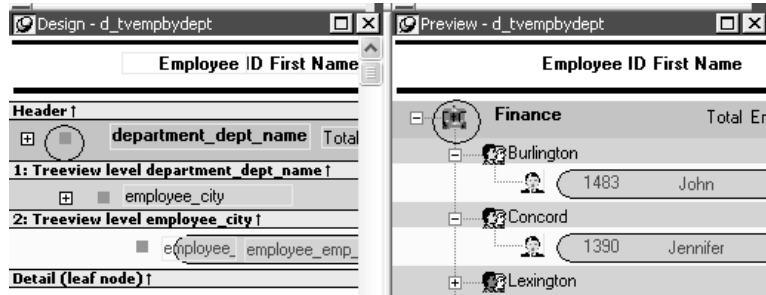
- When there is no tree node icon specified, a shaded square icon in the detail band and in each TreeView level band represents where the connecting line ends.
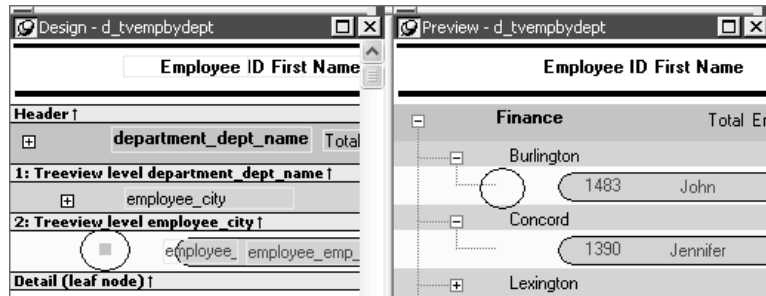
The position of all the icons changes when you change the indent value.

For more information about specifying icons and the indent value, see "Setting properties for the TreeView DataWindow."

Setting properties for the TreeView DataWindow

You can set three types of properties for the TreeView DataWindow:

- General properties
- TreeView level properties
- Detail band properties

Specifying images for tree node icons

In the sample DataWindow shown in "Creating a new TreeView DataWindow" on page 763, different tree node icons display for collapsed and expanded levels. The icons are also different for each level. You specify images for these icons as TreeView level band properties.

The sample DataWindow also displays a tree node icon next to every row in the detail band. You specify an image for this icon as a detail band property.

Tree node icons do not display by default. After specifying images for icons, select the Use Tree Node Icon general property.

Setting general TreeView properties

You set most TreeView DataWindow properties on the General page in the DataWindow Properties view.
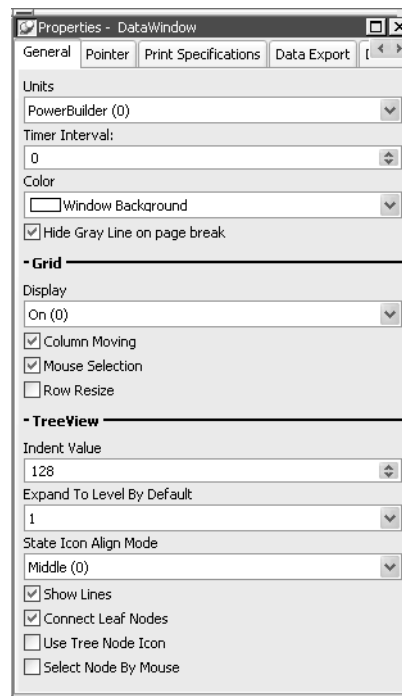
The properties that are specific to a TreeView DataWindow are the TreeView properties and the Grid properties in the General page in the Properties view. The Grid properties display in the Properties view only if you select the Grid Style check box when you define the TreeView DataWindow:

Property	Description
Display On (0)	Grid lines always display.
Display Off (1)	Grid lines never display, so users cannot resize columns at runtime.
Display Only (2)	Grid lines display only when the DataWindow object displays online.
Display Print Only (3)	Grid lines display only when the contents of the DataWindow object are printed.
Column Moving	Columns can be moved at runtime.
Mouse Selection	Data can be selected at runtime and, for example, copied to the clipboard.
Row Resize	Rows can be resized at runtime.
Indent Value	The indent value of the child node from its parent in the units specified for the DataWindow. The indent value defines the position of the state icon. The X position of the state icon is the X position of its parent plus the indent value.
Expand To Level By Default	Expand to TreeView level 1, 2, or 3.
State Icon Align Mode	Align the state icon in the middle (0), at the top (1), or at the bottom (2).
Show Lines	Whether lines display that connect parent nodes and child nodes. If you want to display lines that connect the rows in the detail band to their parent, select Connect Leaf Nodes.
Connect Leaf Nodes	Whether lines display that connect the leaf nodes in the detail band rows.
Use Tree Node Icon	Whether an icon for the tree node displays. This applies to icons in the level and detail bands. For how to specify icon images, see "Setting TreeView level properties" and "Setting detail band properties" next.
Select Node By Mouse	Whether a Tree node is selected by clicking the Tree node with the mouse.

Setting TreeView level properties

In the Properties view, you can specify expanded and collapsed icons for each TreeView level. You access the Properties view by clicking the bar identifying the band for that level in the Design view in the DataWindow painter. You can also access the Properties view from the Rows menu, or by clicking any of the icons in the Design view that represent the locations of nodes, icons, and connecting lines. (See "Icons in the Design view" on page 771.)

❖ **To modify properties for a level in a TreeView DataWindow:**

    1   Select Rows>Edit TreeView Level from the menu bar and then select the number of the level from the list of levels, or click the bar identifying the band for that level or any of the icons in that band.

    2   Use the DataWindow TreeView Level properties view that displays to edit the properties for the level you selected.
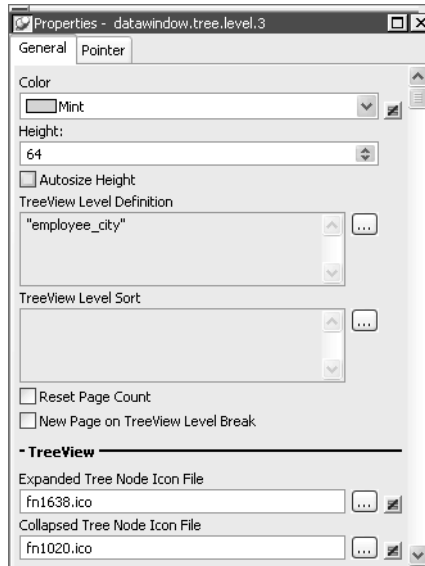
The properties that are specific to a TreeView level band are at the bottom of the Properties view:

Property	Description
Expanded Tree Node Icon File	The file name of the tree node icon in a TreeView level band when it is in the expanded state.
Collapsed Tree Node Icon File	The file name of the tree node icon in a TreeView level band when it is in the collapsed state.

You set the tree node icon file name separately for each TreeView level band. You can use a quoted expression for the tree node icon file.

Setting detail band properties

You can specify an icon for the rows in the detail band by clicking the detail band in the DataWindow painter to display the Properties view.
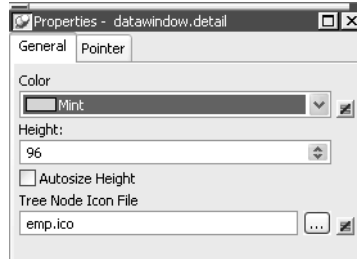
The only property that is specific to the detail band is located at the bottom of the Properties view:

Property	Description
Tree Node Icon File	The file name of the tree node icon in the detail band. You can use a quoted expression.

For more information

For reference information about TreeView DataWindow properties, methods and events, see the *DataWindow Reference* or the online Help.

TreeView DataWindow samples

This section provides you with two samples that demonstrate use of the TreeView DataWindow.

What the samples do

The two TreeView DataWindow samples are:

• Data Explorer

The Data Explorer uses a TreeView DataWindow to display sales-related data in a Windows Explorer-like interface and allows users to update the data.

• Data Linker

The Data Linker uses a TreeView DataWindow on the left for data navigation, linked to four DataWindows on the right for updating the data. The Data Linker demonstrates populating a TreeView DataWindow with data and linking each TreeView level to a separate DataWindow.

Tables and database	Both samples use the employee, sales_order, sales_order_items, customer, and product tables in the EAS Demo DB database.

| TreeView DataWindows | The TreeView DataWindows are *d_sales_report* and *d_sales_report2*. Each TreeView DataWindow has three TreeView levels: |

- The first level (level 1) is the sales representative's name.

  You create the first level using the TreeView DataWindow wizard.

- The second level (level 2) is the name of the customer's company.

  You create the second level using the Rows>Create TreeView Level menu item in the DataWindow painter.

- The third level (level 3) is the sales order ID.

  You also create the third level using the Rows>Create TreeView Level menu item in the DataWindow painter.

Data Explorer sample

What the Data Explorer does	Clicking on each TreeView level displays details in a DataWindow on the right. For example, if you click a name in the TreeView DataWindow on the left, detailed customer data displays in the DataWindow on the right.
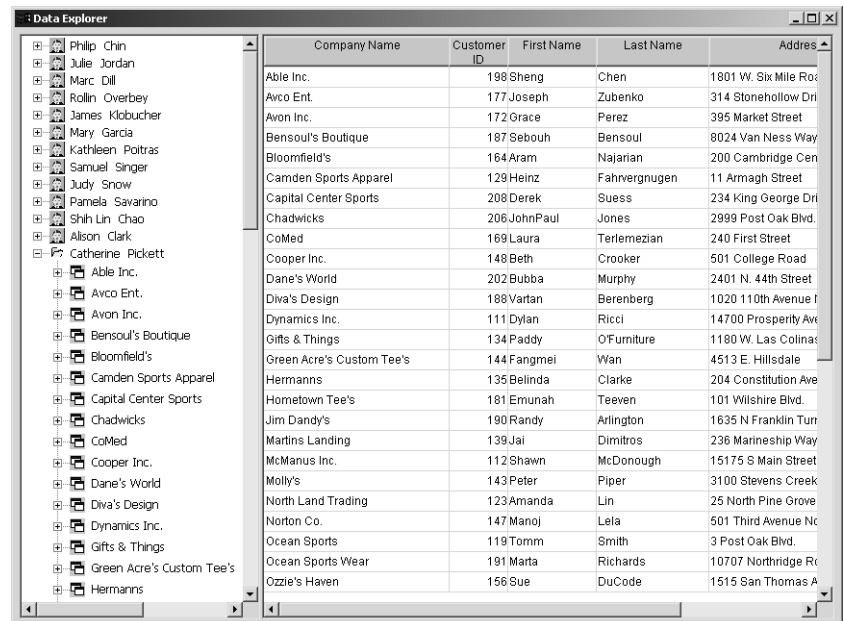
Using the Data
Explorer

You can click on any TreeView level in the Data Explorer. If you click a
company name in the TreeView DataWindow on the left (for example, Able
Inc., under Catherine Pickett), order information displays on the right.
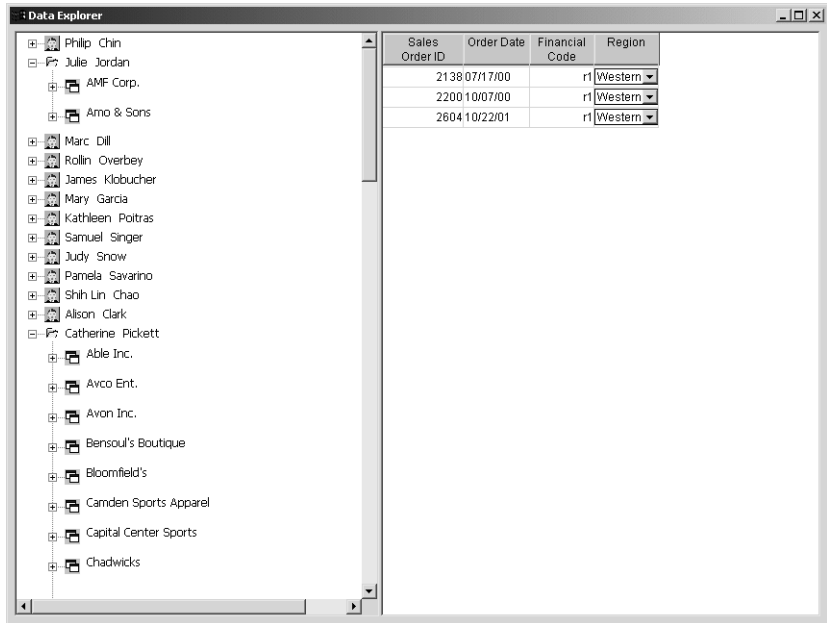
If you click an order ID in the TreeView DataWindow on the left (for example,
order ID 2400, under Bilhome Industries, under Alison Clark), the customer
order information displays on the right.
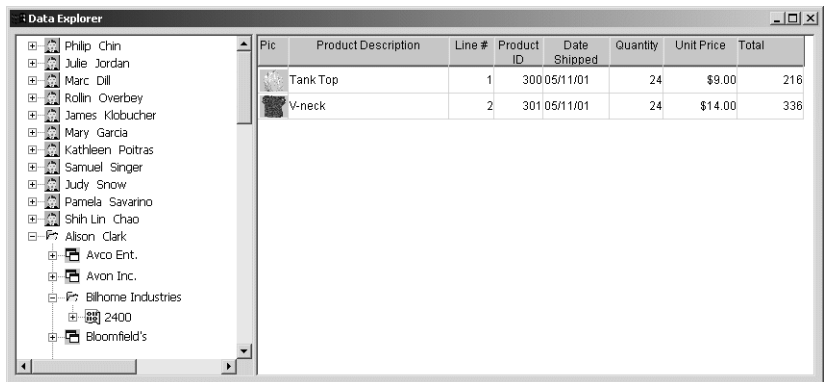
Data Explorer
TreeView
DataWindow

Here is the TreeView DataWindow used in the Data Explorer.
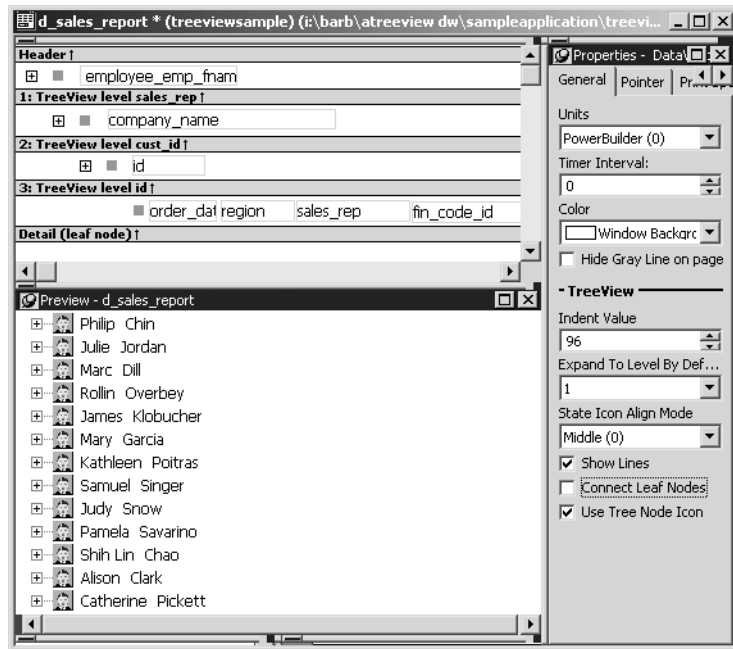
One TreeView DataWindow
The Data Explorer uses one TreeView DataWindow, but DataWindows that are not TreeView DataWindows also support the Data Explorer's functionality.

Data Explorer code

The code in the Clicked event uses GetBandAtPointer to determine which DataWindow to display. Clicking on some editable items in the detail DataWindow opens a window in which you can manipulate the data.

The PopMenu menu object has two menu items that call the CollapseAll and ExpandAll methods to collapse or expand all the nodes in the TreeView.

Data Linker sample

What the Data Linker does

When you first run the Data Linker, no data displays on the right side of the window.
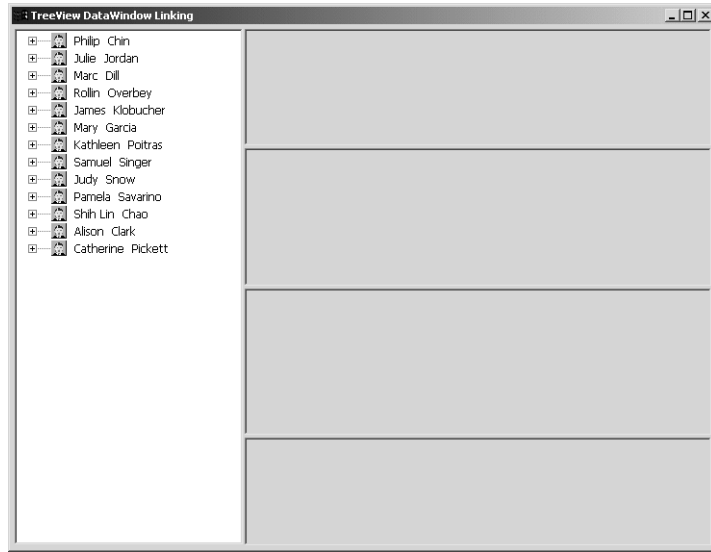
Using the Data Linker

To use the Data Linker, you first expand an employee name and a company's data in the TreeView DataWindow.

Expanding the TreeView displays the company names, the orders for the company you select, and in the detail band, the icon and name for each item in the order.

You can click on each of the TreeView levels in order, and then click in the detail band to display the details in the four DataWindows on the right.

For example, if you click first on Catherine Pickett, then on Avon Inc., then on 2073, and last on Baseball Cap, the data in each of the related DataWindows displays on the right. You can also update the data in each of the DataWindows.
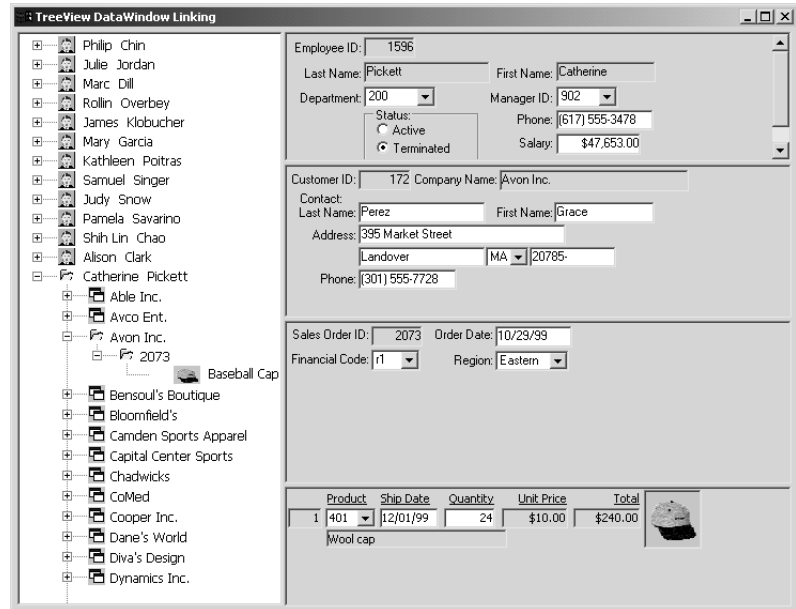
Data Linker TreeView
DataWindow

Here is the TreeView DataWindow used in the Data Linker sample.
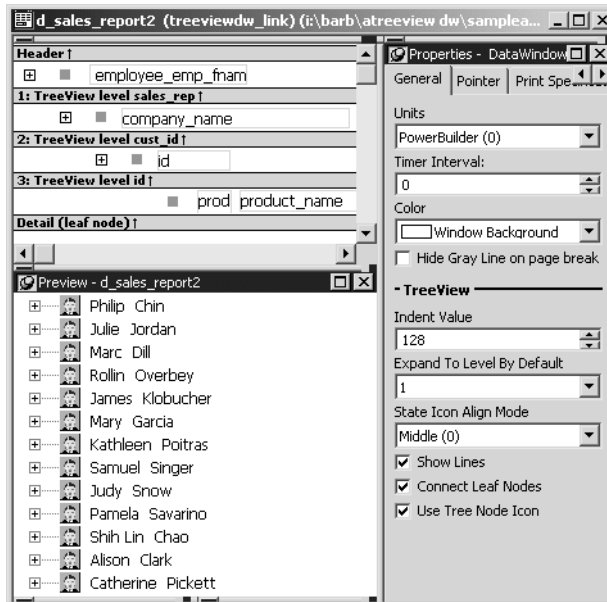
One TreeView DataWindow
The Data Linker uses one TreeView DataWindow, but other DataWindows that
are not TreeView DataWindows also support the Data Linker's functionality.

Data Linker code

The code in the Clicked event uses GetBandAtPointer to determine which
DataWindow to display.

Exporting and Importing XML Data

About this chapter

The row data in a DataWindow can be exported and imported in the Extensible Markup Language (XML). This chapter describes how to create and use templates that control the export and import of data in XML format.

Contents

About XML

Like Hypertext Markup Language (HTML), Extensible Markup Language (XML) is a subset of Standardized General Markup Language (SGML) and has been designed specifically for use on the Web. XML is defined in the W3C Recommendation published by the World Wide Web Consortium. The latest version of this document is available at http://www.w3.org/TR/REC-xml.

XML is more complete and disciplined than HTML, and it is also a framework for creating markup languages—it allows you to define your own application-oriented markup tags.

XML provides a set of rules for structuring data. Like HTML, XML uses tags and attributes, but the tags are used to delimit pieces of data, allowing the application that receives the data to interpret the meaning of each tag. These properties make XML particularly suitable for data interchange across applications, platforms, enterprises, and the Web. The data can be structured in a hierarchy that includes nesting.

An XML document is made up of declarations, elements, comments, character references, and processing instructions, indicated in the document by explicit markup.

The simple XML document that follows contains an XML declaration followed by the start tag of the root element, `<d_dept_list>`, nested row and column elements, and finally the end tag of the root element. The root element is the starting point for the XML processor.

```
<?xml version="1.0">
<d_dept_list>
   <d_dept_list_row>
      <dept_id>100</dept_id>
      <dept_name>R &amp;D</dept_name>
      <dept_head_id>501</dept_head_id>
   </d_dept_list_row>
   ...
</d_dept_list>
```

This section contains a brief overview of XML rules and syntax. For a good introduction to XML, see XML in 10 points at http://www.w3.org/XML/1999/XML-in-10-points. For more detailed information, see the W3C XML page at http://www.w3.org/XML/, the O'Reilly page XML from the inside out at http://www.xml.com, the XML Cover Pages at http://xml.coverpages.org/xml.html, or one of the many books about XML.

Valid and well-formed XML documents

An XML document must be valid, well-formed, or both.

Valid documents

To define a set of tags for use in a particular application, XML uses a separate document named a document type definition (DTD). A DTD states what tags are allowed in an XML document and defines rules for how those tags can be used in relation to each other. It defines the elements that are allowed in the language, the attributes each element can have, and the type of information each element can hold. Documents can be verified against a DTD to ensure that they follow all the rules of the language. A document that satisfies a DTD is said to be valid.

If a document uses a DTD, the DTD must immediately follow the declaration.

XML Schema provides an alternative mechanism for describing and validating XML data. It provides a richer set of datatypes than a DTD, as well as support for namespaces, including the ability to use prefixes in instance documents and accept unknown elements and attributes from known or unknown namespaces. For more information, see the W3C XML Schema page at http://www.w3.org/XML/Schema.

Well-formed documents

The second way to specify XML syntax is to assume that a document is using its language properly. XML provides a set of generic syntax rules that must be satisfied, and as long as a document satisfies these rules, it is said to be well-formed. All valid documents must be well-formed.

Processing well-formed documents is faster than processing valid documents because the parser does not have to verify against the DTD or XML schema. When valid documents are transmitted, the DTD or XML schema must also be transmitted if the receiver does not already possess it. Well-formed documents can be sent without other information.

XML documents should conform to a DTD or XML schema if they are going to be used by more than one application. If they are not valid, there is no way to guarantee that various applications will be able to understand each other.

XML syntax

There are a few more restrictions on XML than on HTML; they make parsing of XML simpler.

Tags cannot be omitted

Unlike HTML, XML does not allow you to omit tags. This guarantees that parsers know where elements end.

The following example is acceptable HTML, but not XML:

```
<table>
  <tr>
    <td>Dog</td>
    <td>Cat
    <td>Mouse
</table>
```

To change this into well-formed XML, you need to add all the missing end tags:

```
<table>
  <tr>
    <td>Dog</td>
    <td>Cat</td>
    <td>Mouse</td>
  </tr>
</table>
```

Representing empty elements

Empty elements cannot be represented in XML in the same way they are in HTML. An empty element is one that is not used to mark up data, so in HTML, there is no end tag. There are two ways to handle empty elements:

- Place a dummy tag immediately after the start tag. For example:

```
<img href="picture.jpg"></img>
```

- Use a slash character at the end of the initial tag:

```
<img href="picture.jpg"/>
```

This tells a parser that the element consists only of one tag.

XML is case sensitive

XML is case sensitive, which allows it to be used with non-Latin alphabets. You must ensure that letter case matches in start and end tags: `<MyTag>` and `</Mytag>` belong to two different elements.

White space

White space within tags in XML is unchanged by parsers.

All elements must be nested.

All XML elements must be properly nested. All child elements must be closed before their parent elements close.

XML parsing

There are two major types of application programming interfaces (APIs) that can be used to parse XML:

- Tree-based APIs map the XML document to a tree structure. The major tree-based API is the Document Object Model (DOM) maintained by W3C. A DOM parser is particularly useful if you are working with a deeply-nested document that must be traversed multiple times.

  For more information about the DOM parser, see the W3C Document Object Model page at http://www.w3.org/DOM.

PowerBuilder provides the PowerBuilder Document Object Model (PBDOM) extension to enable you to manipulate complex XML documents. For more information about PBDOM, see *Application Techniques* and the *PowerBuilder Extension Reference*.

- Event-based APIs use callbacks to report events, such as the start and end of elements, to the calling application, and the application handles those events. These APIs provide faster, lower-level access to the XML and are most efficient when extracting data from an XML document in a single traversal.

   For more information about the best-known event-driven parser, SAX (Simple API for XML), see the SAX page at http://sax.sourceforge.net/.

Xerces parser   PowerBuilder includes software developed by the Apache Software Foundation (http://www.apache.org/). The XML services for DataWindow objects are built on the Apache Xerces-C++ parser, which conforms to both DOM and SAX specifications and is portable across Windows and UNIX platforms. For more information about SAX, see the Xerces C++ Parser page at http://xml.apache.org/xerces-c/index.html.

XML support in the DataWindow painter

PowerBuilder supports both the export and import of XML in DataStore and DataWindow objects using XML template objects. You construct XML templates for export and import graphically in the Export/Import Template view for XML. Each template you create is encapsulated in the DataWindow object. A template enables you to specify the XML logical structure of how the row data iterates inside the root element of the XML document.

The possible uses of this feature include the following:

- You can code events in data entry or data reporting applications to export selected data values, or the entire contents of a DataWindow object, to a structured XML document. The structure of the XML document can be customized for use by other internal or external applications, processes, or systems.

- You can add a method to a custom class user object that uses DataStore objects for server-side database processing or middle-tier management of a client-side DataWindow object. The method would export data to XML, which could then be processed by a different component or subsystem, such as an Enterprise JavaBeans component or a Web service.

• A similar method could be added to a Web application that uses a Web DataWindow. The method might be invoked by a user action, such as selecting checkout in a shopping cart application.

XML services
In addition to the support for XML in the DataWindow painter, PowerBuilder also provides the PowerBuilder Document Object Model (PBDOM). For more information, see the chapter on XML services in *Application Techniques*.

Export templates

An XML export template lets you customize the XML that is generated.

You can specify optional XML and document type declarations that precede the root element in the exported XML, as well as the logical structure and nesting level of iterative DataWindow row data inside the root element. The children of the root element can contain elements, character references, and processing instructions as well as the row data, using explicit markup. For more information, see "Header and Detail sections" on page 792.

If the exported XML is used by different applications or processes, you can define a separate export template for each use.

Import templates

You need to create an import template if you want to import data that does not match the DataWindow column definition or is associated with a schema, or if you want to import attribute values.

Only the mapping of column names to element and attribute names is used for import. All other information in the template is ignored.

Validating XML

XML export and import do not validate the data after export or before import. You can use the XMLParseFile and XMLParseString functions to validate an XML file or string against a DTD or XML schema before proceeding with additional processing.

If no DTD or schema is included or referenced, XMLParseFile and XMLParseString check whether the content is well-formed XML.

The Export/Import Template view for XML

You define and edit templates for export and import in the Export/Import Template view for XML in the DataWindow painter. The view uses a tree view to represent the template.

When you create a new DataWindow object, PowerBuilder displays a default template in the Export/Import Template view. You can edit only one template at a time in the view, but you can create multiple templates and save them with the DataWindow object. Each template is uniquely associated with the DataWindow object open in the painter.

The default template has one element for each column in the DataWindow object.
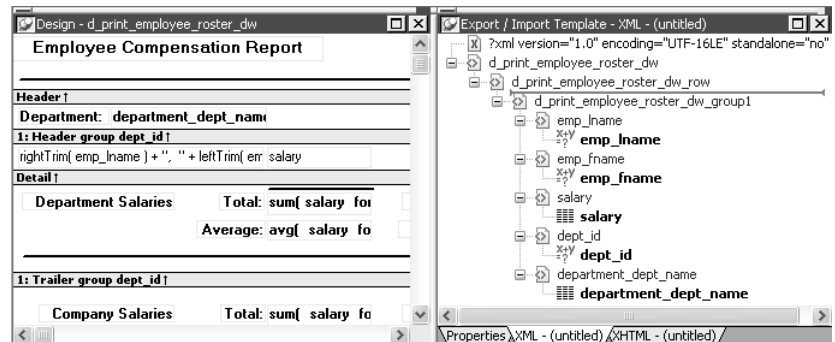
Creating, opening, and saving templates

From the pop-up menu for the Export/Import Template view (with nothing selected), you can create new templates with or without default contents, open an existing template, save the current template, or delete the current template. You can only open and edit templates that are associated with the current DataWindow object.
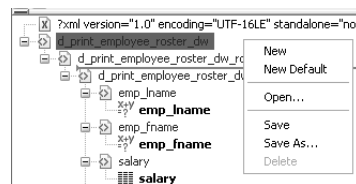
Representing tree view items

Each item in the template displays as a single tree view item with an image and font color that denotes its type. Elements are represented by a yellow icon that resembles a luggage tag. The end tags of elements and the markup delimiters used in an XML document do not display.

Table 29-1 shows the icons used in the Export/Import Template view.

Table 29-1: Icons used in the Export/Import Template view

Icon	Description
	XML declaration or document type declaration
	Root or child element
	Group header element
	Report column reference
	Static text control reference
	Computed field or DataWindow expression reference
	Literal text
	Comment
	Processing instruction
	CDATA section
	Nested report

Creating templates

To create a template, select the New menu item or the New Default menu item from the pop-up menu in the Export/Import Template view.

Creating new base templates

The New menu item creates a template that is empty except for the XML declaration, the root element, and the first element of the row data section, referred to as the Detail Start element. The name of the root element is the same as the name of the DataWindow object, and the default name for the Detail Start element is the name of the root element with _row appended.

For example, if the DataWindow object is named d_name, the default template has this structure:

```
<?xml version="1.0"?>
<d_name>
    <d_name_row>
    </d_name_row>
</d_name>
```

Creating new default templates

The New Default menu item creates a template with the same contents as the New menu item, as well as a flat structure of child elements of the Detail Start element. A child element is created for each DataWindow column name, in the order in which the columns appear in the SELECT statement, with the exception of blob and computed columns. The default tag for the element is the column's name.

If the names of the column and the control are the same, the content of the child element displays with a control reference icon. If there is no control name that matches the column name, the content of the child element displays using the DataWindow expression icon. For example, consider a DataWindow object in which the dept_id column is used as a retrieval argument and does not display:
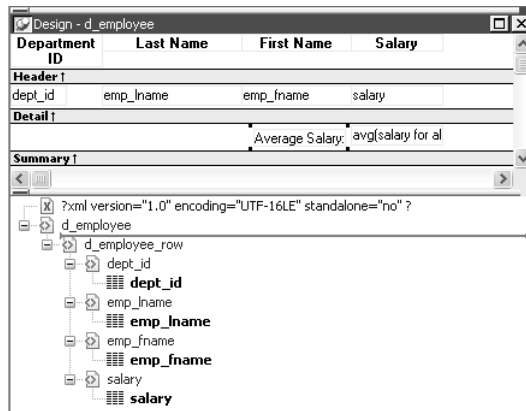
The SQL syntax is:

```
SELECT "employee"."dept_id",
        "employee"."emp_lname",
        "employee"."emp_fname",
        "employee"."salary"
    FROM "employee"
    WHERE employee.dept_id = :deptnum
ORDER BY "employee"."emp_lname" ASC
```

In the default template, dept_id uses the DataWindow expression icon. All the other columns used the column control reference icon.

Saving templates

To save a new template, select Save from the pop-up menu in the Export/Import Template view, and give the template a name and optionally a comment that identifies its use.
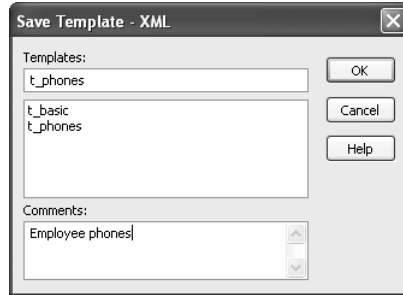
The template is stored inside the DataWindow object in the PBL. After saving a template with a DataWindow object, you can see its definition in the Source editor for the DataWindow object. For example, this is part of the source for a DataWindow that has two templates. The templates have required elements only:

```
export.xml(usetemplate="t_address"
   template=(comment="Employee Phone Book"
      name="t_phone" xml="<d_emplist><d_emplist_row
      __pbband=~"detail~"/></d_emplist>")
   template=(comment="Employee Address Book"
      name="t_address" xml="<d_emplist><d_emplist_row
      __pbband=~"detail~"/></d_emplist>"))
```

Header and Detail sections

An XML template has a Header section and a Detail section, separated graphically by a line across the tree view.

The items in the Header section are generated only once when the DataWindow is exported to XML, unless the DataWindow is a group DataWindow. For group DataWindow objects, you can choose to generate the contents of the header section iteratively for each group. For more information, see "Generating group headers" on page 805.

The Detail section contains the row data, and is generated iteratively for each row in the DataWindow object.
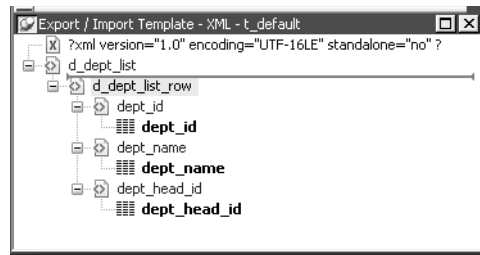
The Detail Start element

A line across the Export/Import Template view separates the Header section from the Detail section. The first element after this line, d_dept_list_row in the previous screenshot, is called the Detail Start element.

There can be only one Detail Start element, and it must be inside the document's root element. By default, the first child of the root element is the Detail Start element. It usually wraps a whole row, separating columns across rows. When the DataWindow is exported to XML, this element and all children and/or siblings after it are generated iteratively for each row. Any elements in the root element above the separator line are generated only once, unless the DataWindow is a group DataWindow and the Iterate Group Headers check box has been selected.

The Detail Start element can be a nested (or multiply-nested) child of an element from the Header section, permitting a nested detail. This might be useful for DataStores being packaged for submission to external processes, such as B2B, that require company and/or document information, date, or other master data preceding the detail.

Moving the separator

You can change the location of the separator line by selecting the element that you want as the Detail Start element and selecting Starts Detail from its pop-up menu. The separator line is redrawn above the new Detail Start element. When you export the data, the Detail Start element and the children and siblings after it are generated iteratively for each row.

If no Detail Start element is specified (that is, if the Starts Detail option has been deselected), the template has only a Header section. When you export the data, only one iteration of row data is generated.

Header section

The Header section can contain the items listed in Table 29-2. Only the root element is required:

Table 29-2: Items permitted in the Header section of an XML document

Item	Details
XML declaration	This must be the first item in the tree view if it exists. See "XML declaration" on page 796.
Document type declaration	If there is an XML declaration, the document type declaration must appear after the XML declaration and any optional processing instructions and comments, and before the root element. Otherwise, this must be the first item in the tree view. See "Document type declaration" on page 797.
Comments	See "Comments" on page 802.
Processing instructions	See "Processing instructions" on page 803.
Root element (start tag)	See "Root element" on page 798.
Group header elements	See "Generating group headers" on page 805.
Child elements	Child elements in the Header section cannot be iterative except in the case of group DataWindows.

Detail section in root element
The root element displays in the Header section, but the entire content of the Detail section is contained in the root element.

Detail section

The Detail section, which holds the row data, can contain the items listed in Table 29-3.

Table 29-3: Items permitted in the Detail section of an XML document

Item	Details
Detail Start element	See "The Detail Start element" on page 793.
Child or sibling elements to the Detail Start element	To add a sibling to the Detail Start element, add a child to its parent (the root element by default).
Control references	These references are in text format and can include references to column, text, computed field, and report controls. See "Controls" on page 799. Nested report controls can only be referenced as child elements. See "Composite and nested reports" on page 800.

Item	Details
DataWindow expressions	See "DataWindow expressions" on page 799.
Literal text	Literal text does not correspond to a control in the DataWindow object.
Comments	See "Comments" on page 802.
Processing instructions	See "Processing instructions" on page 803.
CDATA sections	See "CDATA sections" on page 802.
Attributes	You can assign attributes to all element types. See "Attributes" on page 800.

Editing XML templates

Using templates for data import

If you use a template created for data export, DataWindow expressions, text, comments, and processing instructions are ignored when data is imported. If you are creating a template specifically for import, do not add any of these items. You need only map column names to element and attribute names.

Every item in the Export/Import Template view has a pop-up menu from which you can perform actions appropriate to that item, such as editing or deleting the item, adding or editing attributes, adding child elements or other items, and inserting elements, processing instructions, CDATA sections, and so forth, before the current item.
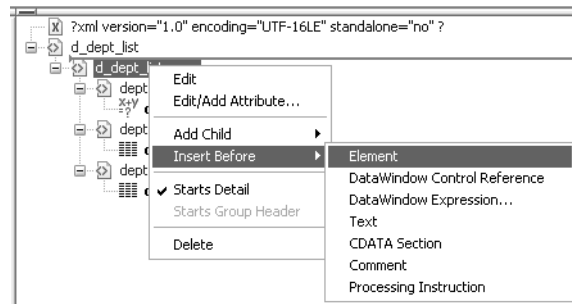
If an element has no attributes, you can edit its tag in the Export/Import Template view by selecting it and left-clicking the tag or pressing F2. Literal text nodes can be edited in the same way. You can delete items (and their children) by pressing the Delete key.

The examples in this section show the delimiters used in the XML document. When you edit the template in dialog boxes opened from the Export/Import Template view for XML, you do not need to type these delimiters in text boxes.

The rest of this section describes some of the items in the template. For more information, see the XML specification at http://www.w3.org/TR/REC-xml.

XML declaration

The XML declaration specifies the version of XML being used. You may need to change this value for a future version of XML. It can also contain an encoding declaration and a standalone document declaration. From the pop-up menu, you can edit the declaration, and, if the document is well-formed, delete it. If you have deleted the XML declaration, you can insert one from the Insert Before item on the pop-up menu for the next item in the template.

Encoding declaration

The encoding declaration specifies the character-set encoding used in the document, such as UTF-16 or ISO-10646-UCS-4.

If there is no encoding declaration, the value defaults to UTF-16LE encoding in ASCII environments. In DBCS environments, the default is the default system encoding on the computer where the XML document is generated. This ensures that the document displays correctly as a plain text file. However, since the DBCS data is serialized to Unicode, XML documents that use UTF-16LE, UTF-16 Big Endian, or UTF-16 Little Endian can all be parsed or generated correctly on DBCS systems.

Several other encodings are available, including ASCII, UCS4 Big Endian, UCS4 Little Endian, EBCDIC code pages IBM037 and IBM1140, ISO Latin-1, and Latin 1 Windows (code page 1252). You can select these values from a drop-down list box in the XML Declaration dialog box.

Standalone document declaration

The standalone document declaration specifies whether the document contains no external markup that needs to be processed and can therefore stand alone (Yes), or that there are, or might be, external markup declarations in the document (No). The value in the default template is No, and if there is no standalone document declaration, the value is assumed to be No.

Example

This is an XML declaration that specifies XML version 1.0, UTF-16LE encoding, and that the document can stand alone:

```
<?xml version="1.0" encoding="UTF-16LE"
standalone="yes"?>
```

Document type declaration

The document type declaration contains or points to markup declarations that provide a grammar for a class of documents. This grammar is known as a document type definition, or DTD. The document type declaration defines constraints on the sequence and nesting of tags, attribute values, names and formats of external references, and so forth. You can edit the document type declaration to change its name, but the name must always be the same as the name of the root element. Changing the name in either the document type declaration or the root element automatically changes the name in the other.

Adding DTDs

You can add an identifier pointing to an external DTD subset, and you can add an internal DTD subset. If you supply both external and internal subsets, entity and attribute-list declarations in the internal subset take precedence over those in the external subset.

Public identifiers

An external identifier can include a public identifier that an XML processor can use to generate an alternative URI. If an alternative URI cannot be generated, the URI provided in the system identifier is used. External identifiers without a public identifier are preceded by the keyword SYSTEM. External identifiers with a public identifier are preceded by the keyword PUBLIC.

Exporting metadata

If you specify a system or public identifier and/or an internal subset in the Document Type Declaration dialog box, a DTD cannot be generated when the data is exported to XML. A MetaDataType of XMLDTD! is ignored. For more information about the properties that control the export of metadata, see "Exporting metadata" on page 809.

Examples

These are examples of valid document type declarations.

An external system identifier:

```
<!DOCTYPE d_dept_listing SYSTEM "d_dept_listing.dtd">
```

An external system identifier with a public identifier:

```
<!DOCTYPE d_test PUBLIC "-//MyOrg//DTD Test//EN"
"http://www.mysite.org/mypath/mytest.dtd">
```

An external system identifier with an internal DTD. The internal DTD is enclosed in square brackets:

```
<!DOCTYPE d_orders
    SYSTEM "http://www.acme.com/dtds/basic.dtd"[
    <!ELEMENT Order (Date, CustID, OrderID, Items*)>
    <!ELEMENT Date (#PCDATA)>
    <!ELEMENT CustID (#PCDATA)>
    <!ELEMENT OrderID (#PCDATA)>
    <!ELEMENT Items (ItemID, Quantity)>
    <!ELEMENT ItemID (#PCDATA)>
    <!ELEMENT Quantity (#PCDATA)>
]>
```

Root element

You can change the name of the root element, add attributes and children, and insert comments, instructions, and, if they do not already exist, XML and/or document type declarations before it.

Changing the name of the root element changes the name of its start and end tags. You can change the name using the Edit menu item, or in the Element Attributes dialog box. Changing the name of the document type declaration, if it exists, also changes the name of the root element, and vice versa. The root element name is always the same as the document type declaration name.

You can add the following kinds of children to the root element:

- Elements

- Text

- Control references

- DataWindow expressions (including column references)

- CDATA sections

- Comments

- Processing instructions

Controls

Adding a DataWindow control reference opens a dialog box containing a list of the columns, computed fields, report controls, and text controls in the document.

Control references can also be added to empty attribute values or element contents using drag-and-drop from the Control List view. Column references can also be added using drag-and-drop from the Column Specifications view.

Drag-and-drop cannot replace
You cannot drag-and-drop an item on top of another item to replace it. For example, if you want to replace one control reference with another control reference, or with a DataWindow expression, you first need to delete the control reference you want to replace.

DataWindow expressions

Adding a DataWindow expression opens the Modify Expression dialog box. This enables you to create references to columns from the data source of the DataWindow object. It also enables the calling of global functions. One use of this feature is to return a fragment of XML to embed, providing another level of dynamic XML generation.

Using Date and DateTime with strings

If you use a control reference or a DataWindow expression that does not include a string to represent Date and DateTime columns in a template, the XML output conforms to ISO 8601 date and time formats. For example, consider a date that displays as `12/27/2004` in the DataWindow object, using the display format `mm/dd/yyyy`. If the export template does not use an expression that includes a string, the date is exported to XML as `2004-12-27`.

However, if the export template uses an expression that combines a column with a Date or DateTime datatype with a string, the entire expression is exported as a string and the regional settings in the Windows registry are used to format the date and time.

Using the previous example, if the short date format in the registry is `MM/dd/yy`, and the DataWindow expression is: `"Start Date is " + start_date`, the XML output is `Start Date is 12/27/04`.

Attributes

Controls or expressions can also be referenced for element attribute values. Select Edit/Add Attribute from the pop-up menu for elements to edit an existing attribute or add a new one.

For each attribute specified, you can select a control reference from the drop-down list or enter a literal text value. A literal text value takes precedence over a control reference. You can also use the expression button to the right of the Text box to enter an expression.
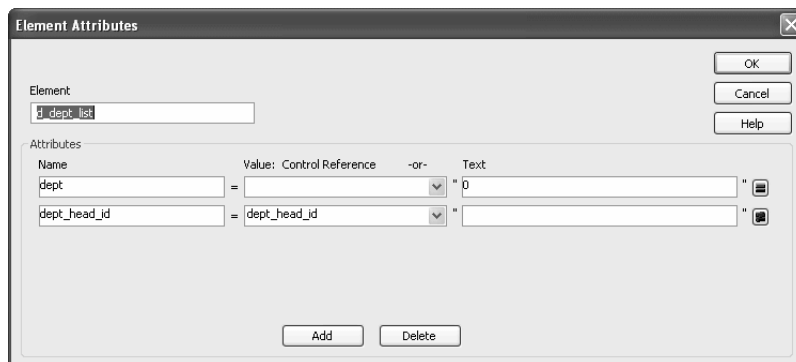
The expression button and entry operates similarly to DataWindow object properties in the Properties view. The button shows a green equals sign if an expression has been entered, and a red not-equals sign if not. A control reference or text value specified in addition to the expression is treated as a default value. In the template, this combination is stored with the control reference or text value, followed by a tab, preceding the expression. For example:

```
attribute_name=~"text_val~~tdw_expression~"
```

Composite and nested reports

Report controls can be referenced in the Detail section of export templates as children of an element.

Nested reports supported for XML export only
Import does not support nested reports. If you attempt to import data in any format, including XML, CSV, DBF, and TXT, that contains a nested report, the nested report is not imported and the import may fail with errors.

Composite reports

For composite reports that use the Composite presentation style, the default template has elements that reference each of its nested reports.

If a composite DataWindow contains two reports that have columns with identical names, you must use the procedure that follows if you want to generate an XML document with a DTD or schema. If you do not follow the procedure, you will receive a parsing error such as "Element '*identical_column_name*' has already been declared."

1. Create a template in the first report and select this template in the Use Template list on the Data Export property page.

2. Create a template in the second report.

3. If any element name is used in the template in the first report, change it to another name in the template in the second report.

4. Select the template for the second report in the Use Template list.

5. Generate the XML document.

These steps are necessary because you cannot use a given element name more than once in a valid DTD or schema.

Nested reports

For report controls added to the detail band of a base report that is related to the inserted report with retrieval arguments or criteria, the report control is available to the export template in two ways:

- Select an element in the template or add a new element, then select Add Child>DataWindow Control Reference. Any report controls inserted in the detail band are available for selection in the dialog box that displays.

- Drag a report control from the Control List view and drop it on an existing empty element.

When you export XML using a template that has a reference to a report control, the export template assigned to the nested report with the Use Template property is used, if it exists, to expand the XML for the nested report. If no template is specified for the nested report, the default template is used.

The relationship between the nested report and the base report, for example a Master/Detail relationship, is reflected in the exported XML.

CDATA sections

You can export the name of a column in a CDATA section using the syntax
`<![CDATA[`***columnname***`]]>`. You can export the value of a column using the
syntax `<![CDATA[~t `***columnname***`]]>`. The ~t is used to introduce a
DataWindow expression, in the same way that it is used in the Modify method.
You can also use an expression such as `~t `***columnname*columnname*** to export
a computed value to the XML.

You can import a value into a column using the syntax `<![CDATA[`
columnname`]]>`. Note that this syntax in a template has different results for
import and export: it imports the column value but exports the column name.

You *cannot* import an XML file that has a ~t expression in a CDATA section.

Everything else inside a CDATA section is ignored by the parser. If text
contains characters such as less than or greater than signs (< or >) or
ampersands (&) that are significant to the parser, it should be defined as a
CDATA section. A CDATA section starts with `<![CDATA[` and ends with `]]>`.
CDATA sections cannot be nested, and there can be no white space characters
inside the `]]>` delimiter—for example, you cannot put a space between the two
square brackets.

Example

```
<![CDATA[
    do not parse me
]]>
```

This syntax in an export template exports the value of the column emp_salary:

```
<![CDATA[~t emp_salary]]>
```

This syntax in an import template imports the value of the column emp_salary:

```
<![CDATA[emp_salary]]>
```

Comments

Comments can appear anywhere in a document outside other markup. They
can also appear within the document type declaration in specific locations
defined by the XML specification.

Comments begin with `<!--` and end with `-->`. You cannot use the string `--`
(a double hyphen) in a comment, and parameter entity references are not
recognized in comments.

Example

```
<!-- this is a comment -->
```

Processing instructions

Processing instructions (PIs) enable you to provide information to the application that uses the processed XML. Processing instructions are enclosed in `<?` and `?>` delimiters and must have a name, called the target, followed by optional data that is processed by the application that uses the XML. Each application that uses the XML must process the targets that it recognizes and ignore any other targets.

The XML declaration at the beginning of an XML document is an example of a processing instruction. You cannot use the string `xml` as the name of any other processing instruction target.

Example

In this example, `usething` is the name of the target, and `thing=this.thing` is the data to be processed by the receiving application:

```
<?usething thing=this.thing?>
```

Exporting to XML

You can export the data in a DataWindow or DataStore object to XML using any of the techniques used for exporting to other formats such as PSR or HTML:

• Using the SaveAs method:

```
ds_1.SaveAs("C:\TEMP\Temp.xml", Xml!, true)
```

• Using PowerScript dot notation or the Describe method:

```
ls_xmlstring = dw_1.Object.DataWindow.Data.XML
ls_xmlstring = dw_1.Describe(DataWindow.Data.XML)
```

• Using the Save Rows As menu item in the DataWindow painter.

With the Preview view open, select File>Save Rows As, select XML from the Files of Type drop-down list, provide a file name, and click Save. You can use this in the development environment to preview the XML that will be generated at runtime.

When you export data, PowerBuilder uses an export template to specify the content of the generated XML.

Default export format
If you have not created or assigned an export template, PowerBuilder uses a default export format. This is the same format used when you create a new default export template. See "Creating templates" on page 790.

OLE DataWindow objects cannot be exported using a template. You must use the default format.

Setting properties on the Data Export page

The Data Export page in the Properties view lets you set properties for exporting data to XML.

In addition to the properties that you can set on this page, PowerBuilder provides two properties that you can use to let the user of an application select an export template at runtime. See "Selecting templates at runtime" on page 813.

The Use Template property

The names of all templates that you create and save for the current DataWindow object display in the Use Template drop-down list.
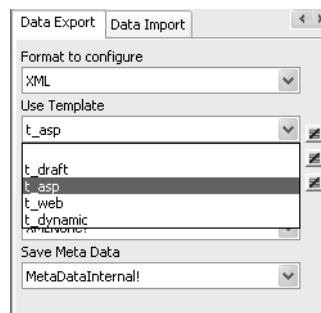
The template you select from the list is used to conform the XML generated by any of the methods of saving as XML to the specifications defined in the named template. Selecting a template from the list box sets the DataWindow object's Export.XML.UseTemplate property. You can also modify the value of the UseTemplate property dynamically in a script. For example, an XML publishing engine would change templates dynamically to create different presentations of the same data.

When you open a DataWindow, the Export/Import Template view displays the template specified in the DataWindow's Use Template property. (If the view is not visible in the current layout, select View>Export/Import Template>XML from the menu bar.) If the property has not been set, the first saved template displays or, if there are no saved templates, the default structured template displays as a basis for editing.

Template used when saving

When the DataWindow is saved as XML, PowerBuilder uses the template specified in the Use Template property. If the property has not been set, PowerBuilder uses the default template.

When you are working on a template, you might want to see the result of your changes. The template specified in the Use Template property might not be the template currently displayed in the Export/Import Template view, so you should check the value of the Use template property to be sure you get the results you expect.

❖ **To save to XML using the current template:**

1   Right-click in the Export/Import template view and select Save or Save As from the pop-up menu to save the current template.

2   On the Data Export page in the properties view, select the current template from the Use Template drop-down list.

3   Select File>Save Rows As, select XML from the Files of Type drop-down list, enter a file name, and click Save.

Generating group headers

To generate the contents of the header section iteratively for each group in a group DataWindow, check the Iterate Header for Groups check box, or set the Export.XML.HeadGroups DataWindow property. This property is on by default.

For example, consider a group DataWindow object that includes the columns sales_order_id and sales_order_order_date. The following screenshot shows the template for this DataWindow object:
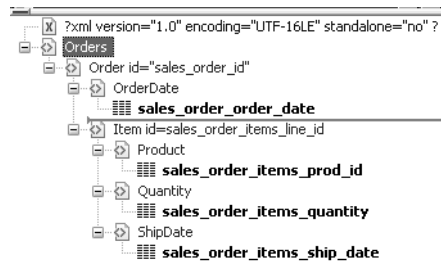
The root element in the Header section of the template, Orders, has a child element, Order. Order has an id attribute whose value is a control reference to the column sales_order_id. Order also has a child element, OrderDate, that contains a column reference to the sales_order_order_date column. These elements make up the header section that will be iterated for each group.

The Detail Start element, Item, has an id attribute whose value is a control reference to the column sales_order_items_line_id. It also has three child elements that contain column references to the line items for product ID, quantity, and ship date.

When the DataWindow is exported with the Iterate Header for Groups property on, the order ID and date iterate for each group. The following XML output shows the first three iterations of the group header:

```
<?xml version="1.0" encoding="UTF-16LE"
standalone="no"?>
<Orders>
    <Order id="2001">
        <OrderDate>2002-03-14</OrderDate>
            <Item id="1">
                <Product>300</Product>
                <Quantity>12</Quantity>
                <ShipDate>2005-09-15</ShipDate>
            </Item>
            <Item id="2">
                <Product>301</Product>
                <Quantity>12</Quantity>
                <ShipDate>2005-09-14</ShipDate>
            </Item>
            <Item id="3">
                <Product>302</Product>
                <Quantity>12</Quantity>
```

```
            <ShipDate>2005-09-14</ShipDate>
         </Item>
    </Order>
    <Order id="2002">
        <OrderDate>2002-03-18</OrderDate>
            <Item id="2">
                <Product>401</Product>
                <Qty>24</Qty>
                <ShipDate>2002-09-18</ShipDate>
            </Item>
            <Item id="1">
                <Product>400</Product>
                <Qty>24</Qty>
                <ShipDate>2002-09-18</ShipDate>
            </Item>
    </Order>
    <Order id="2003">
        <OrderDate>2002-03-21</OrderDate>
            <Item id="3">
                <Product>400</Product>
                <Qty>12</Qty>
                <ShipDate>2002-09-23</ShipDate>
            </Item>
            ...
```

For DataWindow objects with more than one group, when you generate a new default template, each group after the first is identified with a special icon and a check on the pop-up menu next to the Starts Group Header item.
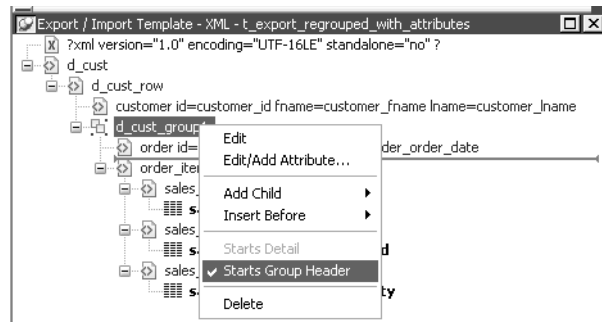
When the Iterate Header for Groups check box is selected, each XML fragment in the header section between a Group Header element and the next Group Header element or Detail Start element is iterated.

In the template shown in the previous illustration, sales are grouped by customer ID, then by order ID. The customer group header has attributes for the customer's ID and first and last names. The order group header has attributes for the order ID and date. The following illustration shows the DataWindow in the Design view:

The following XML output shows the first iteration of the customer group header and the first and second iterations of the order group header:

```xml
<?xml version="1.0" encoding="UTF-16LE" standalone="no"?>
<d_customer>
    <customer id="101" fname="Michaels" lname="Devlin">
        <order id="2001" date="1996-03-14">
            <order_item>
                <sales_order_items_line_id>1</sales_order_items_line_id>
                <sales_order_items_prod_id>300</sales_order_items_prod_id>
                <sales_order_items_quantity>12</sales_order_items_quantity>
            </order_item>
            <order_item>
                <sales_order_items_line_id>2</sales_order_items_line_id>
                <sales_order_items_prod_id>301</sales_order_items_prod_id>
                <sales_order_items_quantity>12</sales_order_items_quantity>
            </order_item>
            <order_item>
                <sales_order_items_line_id>3</sales_order_items_line_id>
                <sales_order_items_prod_id>302</sales_order_items_prod_id>
                <sales_order_items_quantity>12</sales_order_items_quantity>
            </order_item>
        </order>
        <order id="2005" date="1996-03-24">
            <order_item>
                <sales_order_items_line_id>1</sales_order_items_line_id>
                <sales_order_items_prod_id>700</sales_order_items_prod_id>
                <sales_order_items_quantity>12</sales_order_items_quantity>
            </order_item>
        </order>
```

Formatting the exported XML

By default, the XML is exported without formatting. If you want to view or verify the exported XML in a text editor, check the Include Whitespace check box or set the Export.XML.IncludeWhitespace property in a script. Turning this property on causes the export process to insert tabs, carriage returns, and linefeed characters into the XML so that it is easier to read. Most of the examples in this chapter were exported with this property turned on.

Do not import formatted XML
You should not try to import XML formatted with white space characters, because the white space between data element tags is considered to be part of the element.

Exporting metadata

You can specify that metadata in the form of a DTD or schema should be exported when you save the DataWindow object. You can choose to save the metadata with the XML or in a separate file.

If you export metadata as a schema, you can associate it with a namespace. See "Associating a namespace with an exported schema" on page 811.

To specify how metadata should be saved, select a value from the Meta Data Type drop-down list or set the Export.XML.MetaDataType property. The possible values are:

- XMLNone!—No metadata is generated

- XMLSchema!—An XML schema is generated

- XMLDTD!—A DTD is generated

The metadata is saved into the exported XML itself or into an associated file, depending on the setting in the Save Meta Data drop-down list or the Export.XML.SaveMetaData property. The possible values are:

- MetaDataInternal!—The metadata is saved into the generated XML document or string. To save metadata using the .Data.XML expression syntax, you must use this value.

- MetaDataExternal!—The metadata is saved as an external file with the same name as the XML document but with the extension .*xsd* (for a schema) or .*dtd* (for a DTD). A reference to the name of the metadata file is included in the output XML document.

Example: internal
metadata

For example, if you select XMLDTD! and MetaDataInternal!, the header and
first row of the exported XML would look like this for a simple grid
DataWindow for the contact table in the EAS Demo DB. The Include
Whitespace property has also been selected and the file name is
dtdinternal.xml:

```
<?xml version="1.0" encoding="UTF-16LE"
standalone="yes"?>
<!DOCTYPE dtdinternal [<!ELEMENT dtdinternal
(dtdinternal_row*)>
<!ELEMENT dtdinternal_row (id, last_name, first_name,
title, street, city, state, zip, phone, fax)>
<!ELEMENT id (#PCDATA)>
<!ELEMENT last_name (#PCDATA)>
<!ELEMENT first_name (#PCDATA)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT street (#PCDATA)>
<!ELEMENT city (#PCDATA)>
<!ELEMENT state (#PCDATA)>
<!ELEMENT zip (#PCDATA)>
<!ELEMENT phone (#PCDATA)>
<!ELEMENT fax (#PCDATA)>
]>
<dtdinternal>
   <dtdinternal_row>
      <id>1</id>
      <last_name>Hildebrand</last_name>
      <first_name>Jane</first_name>
      <title>ma</title>
      <street>1280 Washington St.</street>
      <city>Emeryville</city>
      <state>MI</state>
      <zip>94608</zip>
      <phone>5105551309</phone>
      <fax>5105554209</fax>
   </dtdinternal_row>
```

Example: external
metadata

If you select MetaDataExternal! instead, the generated XML in
dtdexternal.xml looks like this:

```
<?xml version="1.0" encoding="UTF-16LE"?>
<!DOCTYPE dtdexternal SYSTEM "dtdexternal.dtd">
<dtdexternal>
   <dtdexternal_row>
      <id>1</id>
      <last_name>Hildebrand</last_name>
      <first_name>Jane</first_name>
```

```
        <title>ma</title>
        <street>1280 Washington St.</street>
        <city>Emeryville</city>
        <state>MI</state>
        <zip>94608</zip>
        <phone>5105551309</phone>
        <fax>5105554209</fax>
    </dtdexternal_row>
```

The DTD is in *dtdexternal.dtd*:

```
<?xml version="1.0" encoding="UTF-16LE"?><!ELEMENT
dtdexternal (dtdexternal_row*)>
<!ELEMENT dtdexternal_row (id, last_name, first_name,
title, street, city, state, zip, phone, fax)>
<!ELEMENT id (#PCDATA)>
<!ELEMENT last_name (#PCDATA)>
<!ELEMENT first_name (#PCDATA)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT street (#PCDATA)>
<!ELEMENT city (#PCDATA)>
<!ELEMENT state (#PCDATA)>
<!ELEMENT zip (#PCDATA)>
<!ELEMENT phone (#PCDATA)>
<!ELEMENT fax (#PCDATA)>
```

MetaDataExternal! not supported for dot notation
The metadata cannot be saved in an external file if you use dot notation to
generate the XML.

Associating a
namespace with an
exported schema

If you export metadata in the form of a schema, you can associate a namespace
with the schema. To do so, right-click the root element in the Export/Import
template view and select Schema Options from the pop-up menu. In the dialog
box, specify the namespace prefix and URI.

When the Meta Data Type property is XMLSchema! and the Save Meta Data
property is MetaDataInternal!, so that the XML schema is generated inline, you
can specify a name for the root element. If the root element name is specified,
it appears in the generated XML.

In the following example, the root element name is Contacts, the namespace
prefix is po, and the URI is *http://www.example.com/PO1*.

The example shows the header and the first row of the generated XML:

```
<?xml version="1.0" encoding="UTF-16LE"
standalone="no"?>
<Contacts>
  <xs:schema xmlns:po="http://www.example.com/PO1"
      xmlns:xs="http://www.w3.org/2001/XMLSchema"
      targetNamespace="http://www.example.com/PO1"
      elementFormDefault="qualified"
      attributeFormDefault="unqualified">
    <xs:element name="d_contact_list">
      <xs:complexType>
        <xs:sequence>
          <xs:element ref="d_contact_list_row"
              maxOccurs="unbounded" minOccurs="0"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="d_contact_list_row">
      <xs:complexType>
        <xs:sequence>
          <xs:element ref="id"/>
          <xs:element ref="last_name"/>
          <xs:element ref="first_name"/>
          <xs:element ref="city"/>
          <xs:element ref="state"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="id" type="xs:int"/>
    <xs:element name="last_name" type="xs:string"/>
    <xs:element name="first_name" type="xs:string"/>
    <xs:element name="city" type="xs:string"/>
    <xs:element name="state" type="xs:string"/>
  </xs:schema>
  <po:d_contact_list xmlns:po=
        "http://www.example.com/PO1" xmlns:xsi=
        "http://www.w3.org/2001/XMLSchema-instance">
    <po:d_contact_list_row>
      <po:id>1</po:id>
      <po:last_name>Hildebrand</po:last_name>
      <po:first_name>Jane</po:first_name>
      <po:city>Emeryville</po:city>
      <po:state>MI</po:state>
    </po:d_contact_list_row>
```

By default, the generated XML is not associated with a namespace.

Selecting templates at runtime

Two DataWindow properties, Export.XML.TemplateCount and Export.XML.Template[ ].Name, enable you to provide a list of templates from which the user of the application can select at runtime.

The TemplateCount property gets the number of templates associated with a DataWindow object. You can use this number as the upper limit in a FOR loop that populates a drop-down list with the template names. The FOR loop uses the Template[ ].Name property.

```
string ls_template_count, ls_template_name
long i

ls_template_count=dw_1.Describe
  ("DataWindow.Export.XML.TemplateCount")

for i=1 to Long(ls_template_count)
  ls_template_name=
    dw_1.Object.DataWindow.Export.XML.Template[i].Name
  ddlb_1.AddItem(ls_template_name)
next
```

Before generating the XML, set the export template using the text in the drop-down list box.

```
dw_1.Object.DataWindow.Export.XML.UseTemplate=
  ddlb_1.text
```

Importing XML

You can select XML as a file type in the dialog box that displays when you select Rows>Import in the DataWindow painter. (The Preview view must be open to enable the Rows>Import menu item.)

You can also import data from an XML document or string using the ImportFile, ImportString, or ImportClipboard methods. These methods have an optional first parameter that enables you to specify the type of data to be imported.

Data can be imported with or without a template. To import data without a template, the data must correspond to the DataWindow column definition. The text content of the XML elements must match the column order, column type, and validation requirements of the DataWindow columns.

Composite, OLE, and Graph DataWindow objects
Composite, OLE, and Graph DataWindow objects cannot be imported using a
template. You must use the default format. Graph objects must also be
imported using the default format.

Importing with a template

If the XML document or string from which you want to import data does not
correspond to the DataWindow column definition, or if you want to import
attribute values, you must use a template.

If a schema is associated with the XML to be imported, you must create a
template that reflects the schema.

For complex, nested XML with row data in an iterative structure, you may need
to design a structure that uses several linked DataWindow definitions to import
the data. Each DataWindow must define the structure of a block of iterative
data with respect to the root element. Importing the data into the DataWindow
objects would require multiple import passes using different import templates.

For data that does not conform to an iterative row data structure or has
additional complexities, you can use the PBDOM parser to handle the data on
a node-by-node basis. For more information, see *Application Techniques* and
the *PowerBuilder Extension Reference*.

Defining import
templates

The XML import template can be defined in the Export/Import Template view
for XML. If you are defining a template for use only as an import template, do
not include DataWindow expressions, text, comments, and processing
instructions. These items are ignored when data is imported.

Only mappings from DataWindow columns to XML elements and attributes
that follow the Starts Detail marker in the template are used for import.
Element and attribute contents in the header section are also ignored. If the
Starts Detail marker does not exist, all element and attribute to column
mappings within the template are used for import. For more information about
the Starts Detail marker, see "The Detail Start element" on page 793.

Matching template
structure to XML

An XML import template must map the XML element and attribute names in
the XML document to DataWindow column names, and it must reflect the
nesting of elements and attributes in the XML.

The order of elements and attributes with column reference content in the template does not have to match the order of columns within the DataWindow, because import values are located by name match and nesting depth within the XML. However, the order of elements and attributes in the template must match the order in which elements and attributes occur in the XML. Each element or attribute that has column reference content in the template must occur in each row in the XML document or string. The required elements and attributes in the XML can be empty.

If an element or attribute does not occur in the XML document, the DataWindow import column remains empty.

The data for the DataWindow is held in the columns of the data table. Some data columns, such as those used for computed fields, may not have an associated control. To import data into a column that has no control reference, add a child DataWindow expression that contains the column name.

Remove tab characters
When you select a column name in the DataWindow expression dialog box, tab characters are added before and after the name. You should remove these characters before saving the expression.

Importing data with group headers

For XML import using a template, element and attribute contents in the header section are ignored. However, if the Starts Detail marker does not exist, all element and attribute to column mappings within the template are used for import. This has the following implications for DataWindow objects with group headers:

- If data is imported to a Group DataWindow using a template that has a Starts Detail marker, the group header data is not imported because import starts importing from the Starts Detail location.

- If the Group DataWindow has one group and the import template has no Starts Detail marker, all the data is imported successfully.

Nested groups cannot be imported
If the Group DataWindow has nested groups, the data cannot be imported successfully even if the Starts Detail marker in the import template is turned off.

Restrictions

Report columns cannot be referenced twice for import. A second column reference to a DataWindow column within an XML import template is ignored.

An XML element or attribute name whose content references a DataWindow column for import must be unique within the level of nesting. It cannot occur twice in the template at the same nesting level.

Setting the import template

The names of all templates for the current DataWindow object display in the Use Template drop-down list on the Data Import page in the Properties view.
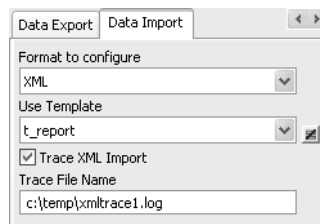
Using export templates for import
If you have already defined an export template for a DataWindow object, you can use it as an import template, but only the mapping of column names to element attribute names is used for import. All other information in the template is ignored.

The template you select in the list box is used to conform the XML imported to the specifications defined in the named template. Selecting a template from the list sets the DataWindow object's Import.XML.UseTemplate property. You can also modify the value of the Import.XML.UseTemplate property dynamically in a script.

The Data Import page also contains a check box that enables you to create a trace log of the import. See "Tracing import" on page 821.

Example

This example uses a DataWindow object that includes the columns emp_id, emp_fname, emp_lname, and dept_id. The template used in this example includes only these columns. Any other columns in the DataWindow remain empty when you import using this template.

To illustrate how template import works, create a new template that has one element in the header section, called before_detail_marker. This element contains a column reference to the emp_id column.

The Detail Start element, employee, has an attribute, dept_id, whose value is a control reference to the column dept_id. It also has three children:

• The emp_id element contains a column reference to the emp_id column.

• The emp_fname element contains static text.

• The name element has two children, emp_fname and emp_lname, that contain column references to those columns.

The template exports and imports the dept_id DataWindow column using the attribute of the employee element. It exports and imports the emp_id, emp_fname, and emp_lname columns using the column references in the elements. The following shows the beginning of the XML exported using this template:

```
<?xml version="1.0" encoding="UTF-16LE"
standalone="no"?>

<employee_list>
  <before_detail_marker>102</before_detail_marker>
  <employee dept_id="100">
     <emp_id>102</emp_id>
     <emp_fname>static text content</emp_fname>
     <name>
       <emp_fname>Fran</emp_fname>
       <emp_lname>Whitney</emp_lname>
     </name>
  </employee>
  <employee dept_id="100">
     <emp_id>105</emp_id>
     <emp_fname>static text content</emp_fname>
     <name>
       <emp_fname>Matthew</emp_fname>
       <emp_lname>Cobb</emp_lname>
     </name>
```

```
                  </employee>
                  ...
```

The exported XML can be reimported into the DataWindow columns dept_id, emp_id, emp_fname, and emp_lname. Before importing, you must set the import template on the Data Import page in the Properties view or in a script using the DataWindow object's Import.XML.UseTemplate property.

The following items are exported, but ignored on import:

- The before_detail_marker element is ignored because it is in the header section.

- The first occurrence of the element tag name emp_fname is ignored because it does not contain a mapping to a DataWindow column name.

If you change the nesting of the emp_fname and emp_lname elements inside the name element, the import fails because the order of the elements and the nesting in the XML and the template must match.

Default data import

When there is no import template assigned to a DataWindow object with the Use Template property, PowerBuilder attempts to import the data using the default mechanism described in this section.

Elements that contain text

The text between the start and end tags for each element can be imported if the XML document data corresponds to the DataWindow column definition. For example, this is the case if the XML was exported from PowerBuilder using the default XML export template.

The text content of the XML elements must match the column order, column type, and validation requirements of the DataWindow columns. (The same restriction applies when you import data from a text file with the ImportFile method).

All element text contents are imported in order of occurrence. Any possible nesting is disregarded. The import process ignores tag names of the elements, attributes, and any other content of the XML document.

Empty elements

Empty elements (elements that have no content between the start and end tags) are imported as empty values into the DataWindow column. If the element text contains only white space, carriage returns, and new line or tab characters, the element is treated as an empty element.

Any attributes of empty elements are ignored.
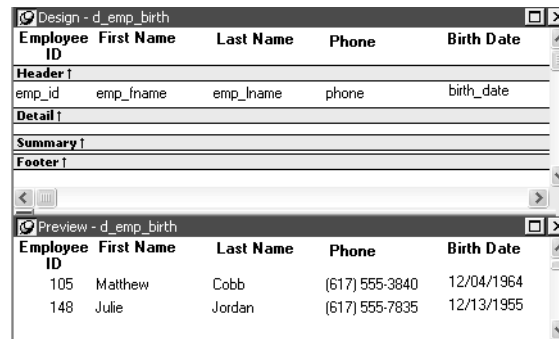
Elements with non-text content	If the element has no text content, but does contain comments, processing instructions, or any other content, it is not regarded as an empty element and is skipped for import.

Example with no empty elements

The three XML documents that follow all show the same result when you select Rows>Import in the DataWindow painter if ImportFile is called with or without default arguments for start and end column, start and end row, and DataWindow start column.

The DataWindow object has five columns: emp_id, emp_fname, emp_lname, phone, and birth_date.

Example 1

This example contains two rows, each with five elements that match the column order, type, and validation requirements for the DataWindow object.

```xml
<?xml version="1.0"?>
<d_emp_birth_listing>
   <d_emp_birth_row>
      <element_1>105</element_1>
      <element_2>Matthew</element_2>
      <element_3>Cobb</element_3>
      <element_4>6175553840</element_4>
      <element_5>04/12/1960</element_5>
   </d_emp_birth_row>
   <d_emp_birth_row>
      <element_1>148</element_1>
      <element_2>Julie</element_2>
      <element_3>Jordan</element_3>
      <element_4>6175557835</element_4>
      <element_5>11/12/1951</element_5>
   </d_emp_birth_row>
</d_emp_birth_listing>
```

Example 2   In this example, the elements are not contained in rows, but they still match the
                     DataWindow object.

```
<?xml version="1.0"?>
<root_element>
    <element_1>105</element_1>
    <element_2>Matthew</element_2>
    <element_3>Cobb</element_3>
    <element_4>6175553840</element_4>
    <element_5>04/12/1960</element_5>
    <element_6>148</element_6>
    <element_7>Julie</element_7>
    <element_8>Jordan</element_8>
    <element_9>6175557835</element_9>
    <element_10>11/12/1951</element_10>
</root_element>
```

Example 3   The comments and processing instructions in this example are not imported.
                     The nesting of the <first> and <last> elements within the <Name> element is
                     ignored.

```
<?xml version="1.0"?>
<root_element>
<!-- some comment -->
<row_element><?process me="no"?>105<name Title="Mr">
<first>Matthew</first>
<last>Cobb</last>
</name>
<!-- another comment -->
<phone>6175553840</phone>
<birthdate>04/12/1960</birthdate>
</row_element>
<row_element>148<name Title="Ms">
<first>Julie</first>
<last>Jordan</last>
</name>
<phone>6175557835</phone>
<birthdate>11/12/1951</birthdate>
</row_element>
</root_element>
```

Result   All three XML documents produce this result:

emp_id	emp_fname	emp_lname	phone	birth_date
105	Matthew	Cobb	6175553840	04/12/1960
148	Julie	Jordan	6175557835	11/12/1951

Example with empty elements

Example 4

This example uses the same DataWindow object, but there are two empty elements in the XML document. The first has no content, and the second has an attribute but no content. Both are imported as empty elements.

```
<?xml version="1.0"?>
<root_element>
<!-- some comment -->
<row_element>
<?process me="no"?>105<name Title="Mr">
<first>Matthew</first>
<!-- another comment -->
<last>Cobb</last>
</name>
<empty></empty>
<birthdate>04/12/1960</birthdate>
</row_element>
<row_element>148<name Title="Ms">
<empty attribute1 = "blue"></empty>
<last>Jordan</last>
</name>
<phone>6175557835</phone>
<birthdate>11/12/1951</birthdate>
</row_element>
</root_element>
```

Result

The XML document produces this result:

emp_id	emp_fname	emp_lname	phone	birth_date
105	Matthew	Cobb		04/12/1960
148		Jordan	6175557835	11/12/1951

Tracing import

When you import data from XML with or without a template, you can create a trace log to verify that the import process worked correctly. The trace log shows whether a template was used and if so which template, and it shows which elements and rows were imported.

To create a trace log, select the Trace XML Import check box on the Data Import page in the Properties view and specify the name and location of the log file in the Trace File Name box. If you do not specify a name for the trace file, PowerBuilder generates a trace file with the name *pbxmtrc.log* in the current directory.

You can also use the Import.XML.Trace and Import.XML.TraceFile DataWindow object properties.

If you use ImportClipboard or ImportString to import the data, you must specify XML! as the *importtype* argument. For example:

```
importstring(XML!, ls_xmlstring)
```

If you omit the *importtype* argument, the trace file is not created. You do not need to specify the *importtype* argument if you use ImportFile.

Example: default import

The following trace log shows a default import of the department table in the EAS Demo database:

```
/*------------------------------------------------*/
/*                 09/10/2005 18:26               */
/*------------------------------------------------*/
CREATING SAX PARSER.
NO XML IMPORT TEMPLATE SET - STARTING XML DEFAULT
IMPORT.
DATAWINDOW ROWSIZE USED FOR IMPORT: 3

ELEMENT: dept_id: 100
ELEMENT: dept_name: R & D
ELEMENT: dept_head_id: 501
--- ROW
ELEMENT: dept_id: 200
ELEMENT: dept_name: Sales
ELEMENT: dept_head_id: 902
--- ROW
ELEMENT: dept_id: 300
ELEMENT: dept_name: Finance
ELEMENT: dept_head_id: 1293
--- ROW
ELEMENT: dept_id: 400
ELEMENT: dept_name: Marketing
ELEMENT: dept_head_id: 1576
--- ROW
ELEMENT: dept_id: 500
ELEMENT: dept_name: Shipping
ELEMENT: dept_head_id: 703
--- ROW
```

Example: template
import

The following trace log shows a template import of the department table. The
template used is named t_1. Notice that the DataWindow column dept_id is
referenced twice, as both an attribute and a column. The second occurrence is
ignored for the template import, as described in "Restrictions" on page 816.
The Detail Start element has an implicit attribute named __pbband which is
also ignored.

```
/*------------------------------------------------*/
/*                09/10/2005 18:25                 */
/*------------------------------------------------*/
CREATING SAX PARSER.
USING XML IMPORT TEMPLATE: t_1

XML NAMES MAPPING TO DATAWINDOW IMPORT COLUMNS:
ATTRIBUTE: /d_dept/d_dept_row NAME: '__pbband'
>>> RESERVED TEMPLATE NAME - ITEM WILL BE IGNORED
ATTRIBUTE: /d_dept/d_dept_row/dept_id_xml_name NAME: 'dept_id'
DATAWINDOW COLUMN: 1, NAME: 'dept_id'
ELEMENT: /d_dept/d_dept_row/dept_id_xml_name
>>> DUPLICATE DATAWINDOW COLUMN REFERENCE: 1, NAME: 'dept_id' - ITEM WILL
BE IGNORED
ELEMENT: /d_dept/d_dept_row/dept_head_id
DATAWINDOW COLUMN: 3, NAME: 'dept_head_id'
ELEMENT: /d_dept/d_dept_row/dept_name
DATAWINDOW COLUMN: 2, NAME: 'dept_name'

ATTRIBUTE: /d_dept/d_dept_row/dept_id_xml_name NAME: 'dept_id': 100
ELEMENT: /d_dept/d_dept_row/dept_head_id: 501
ELEMENT: /d_dept/d_dept_row/dept_name: R & D
--- ROW
ATTRIBUTE: /d_dept/d_dept_row/dept_id_xml_name NAME: 'dept_id': 200
ELEMENT: /d_dept/d_dept_row/dept_head_id: 902
ELEMENT: /d_dept/d_dept_row/dept_name: Sales
--- ROW
ATTRIBUTE: /d_dept/d_dept_row/dept_id_xml_name NAME: 'dept_id': 300
ELEMENT: /d_dept/d_dept_row/dept_head_id: 1293
ELEMENT: /d_dept/d_dept_row/dept_name: Finance
--- ROW
ATTRIBUTE: /d_dept/d_dept_row/dept_id_xml_name NAME: 'dept_id': 400
ELEMENT: /d_dept/d_dept_row/dept_head_id: 1576
ELEMENT: /d_dept/d_dept_row/dept_name: Marketing
--- ROW
ATTRIBUTE: /d_dept/d_dept_row/dept_id_xml_name NAME: 'dept_id': 500
ELEMENT: /d_dept/d_dept_row/dept_head_id: 703
ELEMENT: /d_dept/d_dept_row/dept_name: Shipping
--- ROW
```

Working with Rich Text

About this chapter

This chapter explains how to create DataWindow objects using the RichText presentation style and how to use the RichTextEdit control.

Contents

Topic	Page
About rich text	825
Using the RichText presentation style	826
Using the RichTextEdit control	837
Formatting keys and toolbars	839

About rich text

Rich text format (RTF) is a standard for specifying formatting instructions and document content in a single ASCII document. An editor that supports rich text format interprets the formatting instructions and displays formatted content. If you look at rich text in a plain ASCII editor, you see complex instructions that are not very readable. The actual text of the document is obscured by the formatting instructions:

```
{\par}\pard\ql{\f2\fs18\cf0\up0\dn0 A RichText
piece of text}
```

The same sample displayed without the commands looks like this:

```
A RichText piece of text
```

Elements of rich text

Rich text in PowerBuilder can have:

- Margins and tab settings for each paragraph

- Character formatting such as italic, bold, underline, or superscripts for each character

- Named input fields associated with database columns or other data

- Bitmaps

- A header and footer for the document

The user can use toolbars, editing keys, and a pop-up menu to specify formatting. A print preview lets users view a reduced image of the document to see how it fits on the page.

Rich text support in
PowerBuilder

In PowerBuilder you can use rich text as a DataWindow presentation style. You can also add a RichTextEdit control to a window or visual user object.

What is not supported

PowerBuilder supports version 1.6 of the RTF standard, except for the following features:

> Formatted tables
> Drawing objects

Using the RichText presentation style

The RichText presentation style allows you to combine input fields that represent database columns with formatted text. This presentation style is useful for display-only reports, especially mail-merge documents. However, if you want to use the RichText DataWindow object for data entry, you can specify validation rules and display formats for the input fields.

In the Design view, you see the text along with placeholders called input fields:

```
{FNAME} {LNAME}
{COMPANY_NAME}
{ADDRESS}
{CITY}, {STATE} {ZIP}

Dear {FNAME}:
. . .
```

In the Preview view, the text is the same, but PowerBuilder replaces the input fields with values from the database:

```
Beth Reiser
AMF Corp.
1033 Whippany Road
New York, NY 10154

Dear Beth:
. . .
```

Document template

The formatted text acts like a document template. There is only one copy of the text. As the user scrolls from row to row, the data for the current row is inserted in the input fields and the user sees the document with the current data. If the user edits the text, the changes show up in every row of data.

Input fields

In the RichText presentation style, an input field is associated with a column or computed field. It gets its value from the retrieved data or from the computed field's expression.

If an input field is not a computed field and its name does not match a column, there is no way to specify data for the input field.

There can be more than one copy of an input field in the rich text. In the sample above, there are two instances of the field FNAME. Each instance of the field displays the same data.

Unavailable settings

Not all the settings available in other DataWindow styles are available. You cannot apply code tables and edit styles, such as a DropDownDataWindow or EditMask, to input fields. You cannot use slide left and slide up settings to reposition input fields automatically. However, you can set the LineRemove property at runtime to achieve a similar effect.

Creating the DataWindow object

❖ **To create a RichText DataWindow object:**

1   In the New dialog box, select RichText from the DataWindow tab and click OK.

2   Select data for the DataWindow object as you do for any DataWindow object.

If you want data to be retrieved into the Preview view automatically, select the Retrieve on Preview check box. For more information, see "Building a DataWindow object" on page 474.

3   Specify settings for the DataWindow object on the Specify RichText Settings screen, click Next, and then click Finish.
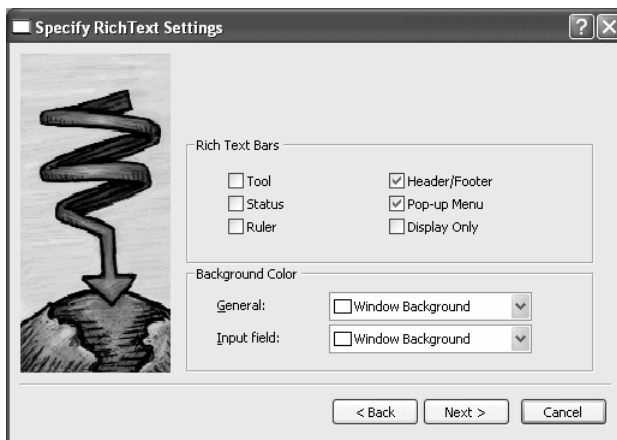
Available settings

Table 30-1 describes the types of settings you can make for the RichText DataWindow object in the wizard.

Table 30-1: Wizard settings for RichText DataWindow objects

You can specify	With these settings
Tools available to the user	Rich text bars: Tool, Status, Ruler, and PopUp Menu
Whether there will be a header and footer for the printed DataWindow object	Header/Footer
Whether users are prevented from editing input fields and text	Display Only
Colors for the whole background and the background of input fields	Background Color: General and Input Field

Editing the content

After you click Finish in the wizard, you see input fields with their labels in the detail band in the Design view:
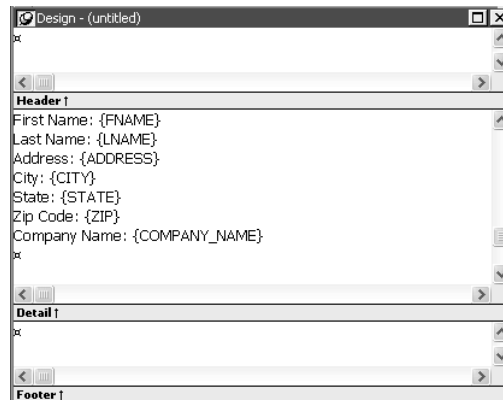
You can:

- Begin editing text in the detail, header, or footer bands, building a report around the input fields. You can delete, move, copy, and paste text and input fields as needed.

- Include a rich text file you have already prepared. If you include a rich text file created in PowerBuilder that contains input fields, those names should match the columns selected in the DataWindow object.

  For information about creating rich text files, see *Application Techniques*.

- Add computed fields that will appear as input fields in the report and whose values come from the computed field expression.

This sample shows how you might rearrange the input fields in a sales letter:
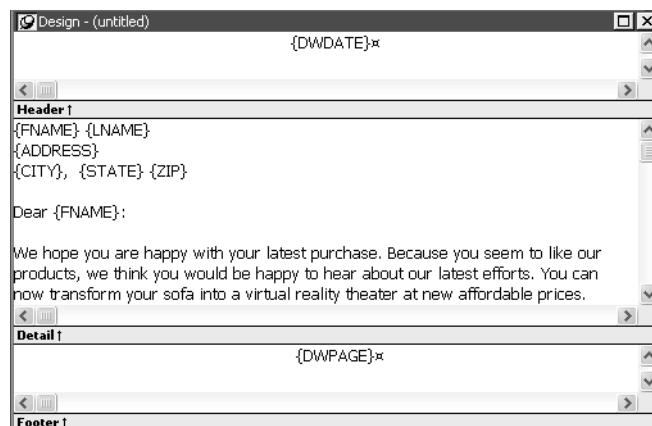
Editing text

You can add text by typing directly in the Design view. You do not have to create text objects as you do for other DataWindow object styles. The Report painter's StyleBar lets you apply formatting to selected text. The RichText toolbars are *not* available in the painter.

Preview mode and editing text
You cannot edit text in the Preview view, but you can edit it when you preview the DataWindow object by selecting File>Run/Preview from the menu bar. It may seem convenient to edit text in Preview mode because the toolbars are available. However, *any changes you make to the text when previewing are temporary*. They are discarded as soon as you return to the Design view.

Inserting a file

If you have a rich text file, you can include it in the DataWindow object. In the Design view, you can insert text from a file into the detail, header, or footer band.

❖ **To insert a file:**

1   Click in the text in any band to set the insertion point for the file.

2   Right-click in the Design view and select Insert File from the pop-up menu.

3   In the file selection dialog box, select the file you want to insert.

Only the body of the file is used. If the file has a header or footer, it is ignored.

Headers and footers

You decide whether your RichText DataWindow object has a header and footer by checking Header/Footer in the wizard or Rich Text Object dialog box (described in "Formatting for RichText objects within the DataWindow object" next). The decision to include a header and footer must be made at design time; it cannot be changed at runtime.

To display a page number or a date in the header or footer, you can insert the predefined computed fields *Page n of n* or *Today()*. You do not need to write scripts to set the values of these fields for each page, as you do for the RichTextEdit control.

Formatting for RichText objects within the DataWindow object

Each type of object in a RichText DataWindow object has its own dialog box. When you select Properties from the pop-up menu, the dialog box you get depends on what is selected.

Properties and Control List views

The Properties and Control List views are not available for RichText DataWindow objects. The painter uses the same property sheets as are available to users when they run the DataWindow object, and controls in RichText DataWindow objects cannot be manipulated in the same way as in other DataWindow objects.

Most of the objects in a RichText DataWindow object correspond to familiar objects like bitmaps, columns, and computed fields. You can also specify formatting for a temporary *selected text object*. In a RichText DataWindow object, the objects are:

• The whole document

• Selected text and paragraphs

• Input fields (associated with columns or computed fields)

• Pictures

This section describes how to select each type of object and access its dialog box. The user can access the property sheets too if you enable the Popup Menu option on the Rich Text Object's General dialog box.

The whole RichText object

Settings for the whole RichText DataWindow object include the values you specified in the wizard, as well as:

• Whether pictures are displayed or represented by empty frames

• Whether newly entered text will wrap within the display

• Whether various nonprinting characters, such as tabs, returns, and spaces, are visible

• Standard DataWindow object settings such as units of measurement and the pointer

• Print specifications

Use the following procedure to change settings:

❖ **To set values for the RichText DataWindow object:**

1 Make sure nothing is selected in the Design view by clicking to set the insertion point.

2 Right-click in the Design view and select Properties from the pop-up menu.

3 Click Help to get more information about a specific setting.

Selected text and paragraphs

You can specify detailed font formatting for selected text. The selected text can be one character or many paragraphs.

If an input field is part of the selection, the font settings apply to it, too. A picture that is part of the selection ignores settings for the selected text object.

❖ **To specify formatting for selected text:**

1 Select the text you want to format.

2 Right-click in the Design view and select Properties from the pop-up menu.

The Selected Text Object dialog box displays. You can set:

• **Paragraph alignment**   The alignment setting on the Selected Text page applies to all paragraphs in the selection.

• **Font formatting**   Settings on the Font page apply to text in the selection, including input fields.

Paragraphs
There are also settings for selected paragraphs. You can display the Paragraph dialog box by pressing Ctrl+Shift+S. The user can double-click the ruler bar or press the key combination to display the same dialog box.

Default font
The user can change the default font by double-clicking on the toolbar or pressing Ctrl+Shift+D. You cannot change the default font in the painter.

Input fields

An input field can be either a column or a computed field. Before you retrieve data, its value is shown as two question marks (??).

The text can include many copies of a named input field. The same data will appear in each instance of the input field.

Column input fields

The columns you select for the DataWindow object become input fields in the rich text. Because the input field's name matches the column name, PowerBuilder displays the column's data in the input field.

If an input field exists in the text, you can copy and paste it to create another copy. If you need to recreate a column input field that you deleted, use this procedure.

❖ **To insert a column input field in the text:**

1 Select Insert>Control>Column from the menu bar.

2 Click in the text where you want the column input field to appear.

   PowerBuilder displays a list of the columns selected for the DataWindow object.

3 Select a column for the input field.

Properties for input fields

You select an input field by clicking inside it. A computed input field is selected when the whole field is highlighted.

❖ **To set properties for an input field:**

1 Click in the input field in Design view.

2 Display the pop-up menu and select Properties.

3 On the Font page, specify text formatting.

4 On the Format page, specify a display format.

5 On the Validation page, specify a validation rule for the column.

If there are multiple copies of an input field, the validation and format settings apply to all the copies. Background color on the Font page applies to all input fields. Other settings on the Font page apply to individual instances.

The user cannot change the format or validation rule. At runtime, these pages are not available in the dialog box.

Computed field input fields   When you display the dialog box for a computed field, the settings are a little different. You can specify the input field name and its expression on the Compute page and there is no validation.

Data Value in preview   For both columns and computed fields, you see a value in the Data Value box when you preview the DataWindow object. The user sees a value in the Data Value box when the current row has a value. For columns, users can change the value.

Computed fields

Computed fields have an expression that specifies the value of the computed field. In rich text, they are represented as input fields, too. You specify a name and an expression. The data value comes from evaluating the expression and cannot be edited.

❖ **To define a computed field:**

1 Select Insert>Control>Computed Field.

Predefined computed fields
You can also select one of the predefined computed fields at the bottom of the menu. PowerBuilder provides several predefined computed fields, but in a RichText DataWindow object, only the page number (*Page n of n*) and today's date (*Today()*) are available.

---

2 Click in the text where you want the computed field to appear.

If you do not select a predefined computed field, PowerBuilder displays the dialog box for the computed field:
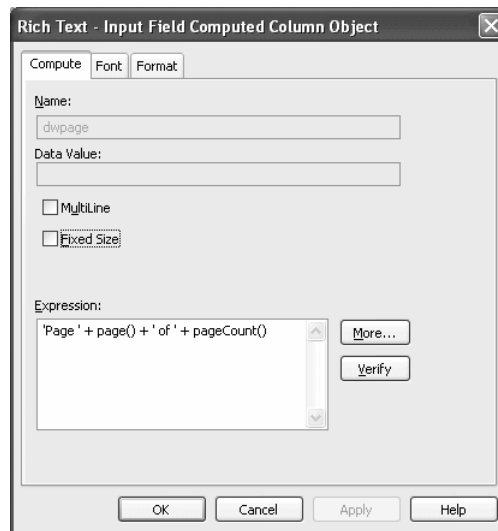
3 On the Compute page, name the computed field and specify its expression.

4 (Optional) On the Font page, specify text formatting.

5 (Optional) On the Format page, specify a display format.

If there are multiple copies of a computed field input field, the expression and format settings apply to all the copies. Font settings apply to individual instances. For more about computed field expressions and display formats, see Chapter 19, "Enhancing DataWindow Objects."

Pictures

Inserting a picture

You can include bitmaps (*BMP*, *GIF*, *JPG*, *RLE*, or *WMF* files) in a RichText DataWindow.

❖ **To insert a picture in the rich text:**

1   Select Insert>Control>Picture from the menu bar.

2   Click in the text where you want the picture to appear.

     PowerBuilder displays the Select Picture dialog box.

3   Select the file containing the picture.

Specifying picture size

A picture is selected when you can see a dashed outline in Design or Preview view. When the picture is part of a text selection, it displays with inverted colors.

You can change the size of a picture as a percentage of the original picture size. The allowable range for a size percent change is between 10 and 250 percent.

❖ **To specify size settings for the picture:**

1   Click on the picture in the Design or Preview view so you see its dashed-outline frame.

2   Right-click in the Design or Preview view and select Properties from the pop-up menu.

     The Rich Text - Picture Object dialog box displays.

3   Change the percent of the original picture size in the Width and Height text boxes.

     The picture expands or contracts according to the size percentage you selected.

Previewing and printing

To see what the RichText DataWindow object looks like with data, you can preview it in the Preview view or in preview mode.

❖ **To preview the DataWindow object in preview mode:**

1   Select File>Run/Preview from the menu bar, or click the Run/Preview button on the PowerBar.

2   Select Rows>Retrieve from the menu bar.

Retrieve on Preview
If the RichText definition specifies Retrieve on Preview, data is retrieved automatically when you open the Preview view or preview the DataWindow object in preview mode.

Changes in preview

Data   While previewing the DataWindow object in preview mode, or when focus is in the Preview view, you can use the scroll buttons in the Preview toolbar to move from row to row, and you can change data in the input fields. If you choose the Save Changes button on the toolbar, you will update the data in the database.

Text   Any changes you make to the rich text in the Preview view *will not be reflected* in the Design view. Any changes that you want to keep must be made in the Design view, not in preview.

If the Display Only setting is checked, you cannot change text or data in the Preview view.

Print Preview

Print Preview displays a reduced view of one row of data as it would appear when printed.

❖ **To see the DataWindow object in Print Preview:**

1   Click in the Preview view to make it the current view.

2   Select File>Print Preview.

In Print Preview, you can test different margin settings and scroll through the pages of the document.

You *cannot* scroll to view other rows of data.

Any changes you make to settings in Print Preview are discarded when you return to the Design view.

Setting margins
To specify permanent margin settings for the RichText DataWindow object,
use the Print Specifications page of the Rich Text Object dialog box.

Using the RichTextEdit control

You can add a RichTextEdit control to a window to enhance your application
with word processing capabilities.

Users can enter text in a RichTextEdit control, format it, save it to a file, and
print it. You can also enable a pop-up menu from which users can control the
appearance of the control and import documents.

❖ **To add a RichTextEdit control to a window:**

• In the Window painter, select Insert>Control>RichTextEdit and click the
window.

Controlling the
appearance of a
RichTextEdit control

You modify the appearance of a RichTextEdit control by setting its properties.
Some of the properties you can set are:

• The toolbars that appear in the control

• The visibility of nonprinting characters and graphics

❖ **To control the appearance of a RichTextEdit control:**

1 Select the control, then select the Document tab in the Properties view.

2 Choose the appropriate properties to display toolbars.

3 Choose the appropriate properties if you want to display nonprinting
characters such as tabs, spaces, and returns.

For information about other options on the Document properties page,
select Help from the property page's pop-up menu.

Making a RichTextEdit
control read-only

There are times when you might want to import a file into the RichTextEdit
control and not give the user the opportunity to alter it. You can make a control
read-only by setting the Enabled and Popup Menu properties.

❖ **To make a RichTextEdit control read-only:**

1 Select the control, then select the General tab in the Properties view.

2 Make sure the Enabled check box is cleared.

3 Select the Document tab.

4 Make sure the PopMenu check box is cleared.

Enabling the pop-up menu

If you enable the pop-up menu property, users can customize the appearance of the RichTextEdit control.

From the pop-up menu, users can:

• Perform editing tasks (cut, copy, paste, and clear)

• Insert a file into the RichTextEdit control

• Display and modify the Rich Text Object dialog box
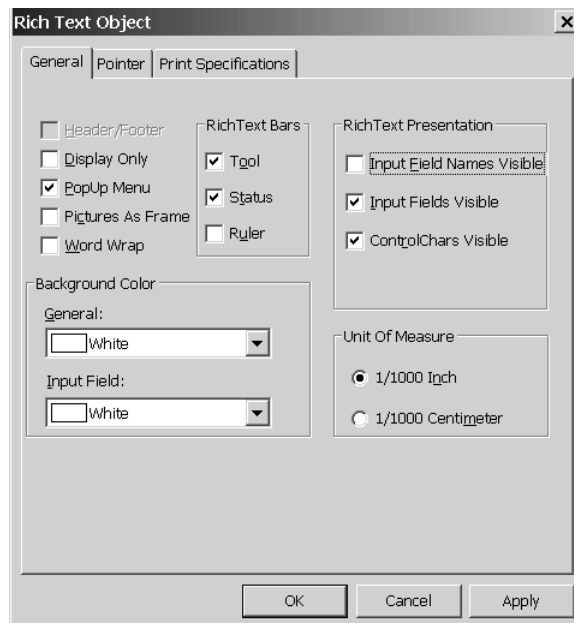
The General property page on the user's Rich Text Object dialog box presents many of the same options as the Document property page in the development environment.

For more information about the RichTextEdit control, see the chapter on implementing rich text in *Application Techniques*.

Formatting keys and toolbars

When the toolbar is visible, you can use its buttons to format text. The changes you make in preview are temporary.

The keystrokes listed in the following tables also assign formatting to selected text.

Keyboard shortcuts do not work in the painter
These keystrokes work only when you are running the DataWindow object or the window containing the RichTextEdit control. In PowerBuilder, only keyboard shortcuts defined for menu items in the painter can be used.

Table 30-2: Keyboard shortcuts for RichText DataWindow objects

Category	Action	Key
Using the clipboard	Cut	Ctrl+X
	Paste	Ctrl+V, Shift+Insert
	Copy	Ctrl+C
	Undo	Ctrl+Z
Assigning font attributes	Bold	Ctrl+B
	Italic	Ctrl+I
	Underline	Ctrl+U
	Subscript	Ctrl+=
	Superscript	Ctrl+Shift+=
	Strikeout	Ctrl+U
	Change font	Ctrl+Shift+U
Setting line spacing	Single space	Ctrl+1
	Double space	Ctrl+2
	One and a half space	Ctrl+5
Aligning text	Justify	Ctrl+J
	Center	Ctrl+E
	Left	Ctrl+L
	Right	Ctrl+R
	Set paragraph formatting	Ctrl+Shift+S
Editing	Insert a new paragraph	Enter
	Insert an empty line	Ctrl+N
	Delete character to right of insertion point	Delete
	Delete character to left of insertion point	Backspace

Category	Action	Key
Input fields	Select the input field at the insertion point	Enter
	Activate the input field at the insertion point	Space
	When input field is active, accept data and exit field	Enter
	When input field is active, exit field without changing data	Esc
	Move to next input field	Ctrl+Tab
	Move to previous input field	Shift+Ctrl+Tab
Miscellaneous	Select All	Ctrl+A
	Print	Ctrl+P
	Undo	Ctrl+Z
	Toggle display of nonprinting characters	Ctrl+*
	Toggle preview mode	Ctrl+F2

Navigating and selecting text

Table 30-3: Keyboard shortcuts for navigating and selecting text

Move or select	Navigating key	Selection key
A character to the right or left	Right arrow or Left arrow	Shift+Right arrow or Shift+Left arrow
A word to the right or left	Ctrl+Right arrow or Ctrl+Left arrow	Ctrl+Shift+Right arrow or Ctrl+Shift+Left arrow
A line up or down	Up arrow or Down arrow	Shift+Up arrow or Shift+Down arrow
To start of line	Home	Shift+Home
To end of line	End	Shift+End
To start of document	Ctrl+Home	Ctrl+Shift+Home
To end of document	Ctrl+End	Ctrl+Shift+End
To next input field	Ctrl+Tab	
To previous input field	Shift+Ctrl+Tab	

Using OLE in a DataWindow Object

About using OLE in DataWindow objects

A DataWindow object can include a control that is a container for an OLE object. The container stores information about the application that created the object and it can launch the application to display or modify the OLE object.

The container can fill the whole DataWindow object, when you create a new DataWindow object using the OLE presentation style, or it can exist alongside other controls in a DataWindow object, when you add an OLE object to an existing DataWindow object. You can also read OLE data from a blob column in a database and display the objects in the DataWindow object.

You can use OLE objects in DataWindow objects in the following ways:

- **OLE object in a DataWindow object**   The OLE object is displayed in its container control with the DataWindow data and other controls, such as bitmaps or text. You can associate it with data in a particular row, the rows on a page, or with all rows. You choose which columns in the DataWindow object are transferred to the OLE object. You can add an OLE container control to a DataWindow object that uses any presentation style that supports multiple DataWindow objects. (This does not include the graph and RichText presentation styles.)

- **OLE presentation style**   The OLE presentation style is similar to an OLE object in a DataWindow object. The difference is that the OLE container is *the only* control in the DataWindow object. The underlying data is *not* presented in column controls and *there are no other* controls, such as bitmaps or text. The OLE object is *always* associated with all the rows in the DataWindow object.

- **OLE database blob column**   OLE objects that are stored in the database in a blob column are displayed in each row of the DataWindow object.

You can also add ActiveX controls (also called OLE custom controls or OCXs) to DataWindow objects. ActiveX controls range from simple visual displays, such as meters and clocks, to more complex controls that perform spell checking or image processing.

The behavior of OLE objects in DataWindow objects is similar to the behavior of OLE controls in windows.

For more information about linked and embedded objects and automation, see the chapter on using OLE in an application in *Application Techniques*.

Activating OLE objects

When you are working in the DataWindow painter, you can start the server application for an OLE object by selecting Open from the pop-up menu. Once the server application has started, you can use the tools provided by the server to edit the initial presentation of the object.

If the OLE object is associated with *all rows* retrieved and is in the foreground or background layer, not the band layer, users can activate the object. If the object is associated with a single row or page or is in the band layer, users can see the object but cannot activate it. DataWindows created using the OLE presentation style are *always* associated with all rows.

Unlike OLE objects, ActiveX controls are always active. They do not contain objects that need to be opened or activated.

Editing OLE objects

When an OLE object is activated, you can edit the presentation of the data. Changes made to DataWindow data affect the OLE object. Changes made to the OLE object *do not* affect the data the DataWindow object retrieved.

Each OLE object stored in the database in a blob column can be activated and changed. When the DataWindow object updates the database, the changes are saved.

What's next

Whether you are inserting an OLE object into a DataWindow object or using the OLE presentation style, you use the same procedures to define, preview, and specify data for the OLE object. Because of their similarities, the next section discusses both OLE objects in DataWindow objects and the OLE presentation style. The last section discusses OLE database blob columns.

OLE objects and the OLE presentation style

Whether you insert an OLE object into a DataWindow object or create a new DataWindow object using the OLE presentation style, you are working with an OLE container object within the DataWindow object.

Similarities

They have these characteristics in common:

- **Icon or contents**   The DataWindow object can display the OLE object as an icon, or it can display an image of the contents when display of contents is supported by the server.

- **Data from the DataWindow object**   You specify which DataWindow columns you want to transfer to the OLE object. The data that is sent to the OLE server replaces the OLE object template specified in the painter.

Differences

The OLE object in a DataWindow object and the OLE presentation style have these main differences:

- **Associating the object with rows**   When the OLE object is added to a DataWindow object, you can associate it with individual rows, groups of rows, or all rows. In the presentation style, the OLE object is *always* associated with all rows.

- **Properties view**   The Properties view for an OLE object has different pages and some different properties from the OLE DataWindow object. For example, the Properties view for an OLE object in a DataWindow object does not contain detailed print specification settings because these are set in the DataWindow object's own Properties view. However, it does have settings related to the position of the OLE object within the DataWindow object.

Not all servers are appropriate
The features of the OLE server application determine whether it can provide useful information in a DataWindow object.

If the server does not support display of contents, it is not useful for objects associated with rows. The user sees only the icon. Some servers support the display of contents, but the view is scaled too small to be readable even when the object is activated.

In this section

This section includes procedures for:

- Adding an OLE object to a DataWindow object

- Using the OLE presentation style

- Defining the OLE object

- Previewing the DataWindow object

- Specifying DataWindow data for the OLE object

Adding an OLE object to a DataWindow object

To add an OLE object to a DataWindow object, you begin by specifying where you want the OLE object and opening the Insert Object dialog box so you can define the OLE object.

Adding an ActiveX control
Adding an ActiveX control to a DataWindow object is similar to adding an OLE object. Both exist within the DataWindow object with other controls, such as columns, computed fields, and text controls. Use the following procedure whether you want to add an OLE object or an ActiveX control to an existing DataWindow object.

❖ **To place an OLE object in a DataWindow object:**

1   Open the DataWindow object that will contain the OLE object.

2   Select Insert>Control>OLE Object from the menu bar, or from the toolbar, click the Object drop-down arrow and select the OLE button (*not* OLE Database Blob).

3   Click where you want to place the OLE object.

PowerBuilder displays the Insert Object dialog box.

To use the Insert Object dialog box, see "Defining the OLE object" on page 846.

Using the OLE presentation style

Use the OLE presentation style to create a DataWindow object that consists of a single OLE object. The following procedure creates the new DataWindow object and opens the Insert Object dialog box.

❖ **To create a new DataWindow object using the OLE presentation style:**

1 In the New dialog box, select OLE 2.0 from the DataWindow tab and click OK.

2 Select data for the DataWindow object as you do for any DataWindow object.

For more information about selecting data, see Chapter 18, "Defining DataWindow Objects."

3 Specify how the OLE object will use the DataWindow object's data on the Specify OLE Data page:
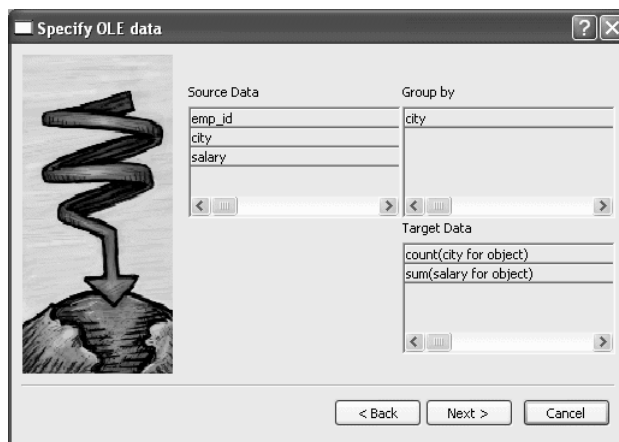
You can drag the columns you want the OLE object to use to the Target Data box. You can also control the grouping of data and edit the expression for a column. If necessary, you can change these specifications later.

For more information, see "Specifying data for the OLE object" on page 848.

4 Click Next, and then click Finish.

PowerBuilder displays the Insert Object dialog box in which you define the OLE object.

To use the Insert Object dialog box, see "Defining the OLE object" next.

Defining the OLE object

You define the OLE object in the Insert Object dialog box. It has three tab pages:

If you want to	Select this tab page
Embed an OLE server object in the DataWindow object	Create New
Link or embed the contents of an existing file as an OLE object so that it can be activated using the application that created it	Create From File
Insert an ActiveX control in the DataWindow object	Insert Control

This section contains procedures for each of these selections.

Create New

Use the following procedure if you want to embed a new OLE server object.

❖ **To embed a new OLE server object using the Create New tab:**

1   Select the Create New tab.

2   In the Object Type box, highlight the OLE server you want to use.

   You can click Browse to get information about the server from the registry.

3   Optionally display the OLE object as an icon by doing one of the following:

   •   Check Display as Icon to display the server application's default icon in the control.

   •   Check Display as Icon and then select Change Icon to supply a nondefault icon and icon label.

4   Click OK.

   The OLE object is inserted in your DataWindow object and the OLE server is activated. Depending on the OLE server and whether or not you have already specified how the OLE object will use the DataWindow object's data, the object may be empty or may show an initial presentation of the OLE object. Close the server application and, if you are inserting an OLE object in a DataWindow object, specify the object's properties (see "Specifying properties for OLE objects" on page 848).

Create From File

Use the following procedure if you want to link or embed the contents of an existing file as an OLE object so that it can be activated using the application that created it. Most of the steps in this procedure are the same as those for embedding a new OLE server object.

A server application must be available

You (and the user) must have an application that can act as a server for the type of object you link or embed. For example, if you insert a BMP file, it displays because an application that can handle bitmaps is installed with Windows. If you insert a GIF or JPEG file, it displays only if you have a third-party graphics application installed.

❖ **To link or embed an existing object using the Create From File tab:**

1   Select the Create From File tab.

2   Specify the file name in the File Name box. If you do not know the name of the file, click the Browse button and select a file in the dialog box.

3   To create a link to the file, rather than embed a copy of the object in the control, select the Link check box.

4   Click OK.

The OLE object is inserted in your DataWindow object and the OLE server is activated. Depending on the OLE server and whether or not you have already specified how the OLE object will use the DataWindow object's data, the object might be empty or might show an initial presentation of the OLE object. Close the server application and, if you are inserting an OLE object in a DataWindow object, specify the object's properties (see "Specifying properties for OLE objects" on page 848).

Insert Control

Use the following procedure if you want to insert an ActiveX control (OLE custom control) in the DataWindow object.

❖ **To insert an ActiveX control using the Insert Control tab:**

1   Select the Insert Control tab.

2   In the Control Type box, highlight the ActiveX control you want to use, or, if the ActiveX control you want has not been registered, click Register New.

If you select an existing ActiveX control, you can click Browse to get more information about it. ActiveX controls are self documenting. PowerBuilder gets the property, event, and function information from the ActiveX control itself from the registry.

If you click Register New, you are prompted for the file that contains the registration information for the ActiveX control.

3   Click OK.

4   If you did not specify how the OLE object will use the DataWindow object's data when you created the DataWindow object, do so on the Data property page.

If you have inserted an ActiveX control that does not display data, such as the Clock control, you do not need to transfer data to it.

For more information, see "Specifying data for the OLE object" on page 848.

Specifying properties for OLE objects   For OLE objects, you need to specify how the OLE object will use the DataWindow object's data. If you used the OLE presentation style, you did this when you created the DataWindow object.

If you are inserting an OLE object in an existing DataWindow object, you can also associate the object with the current row. If you are using the OLE presentation style, the OLE object is always associated with all rows.

❖   **To specify properties for an OLE object:**

1   Select the Data property page in the Properties view.

2   Specify how the OLE object will use the DataWindow object's data.

For more information, see "Specifying data for the OLE object" next.

3   (Optional) To associate the object with the current row, select the Position property page and change the value in the Layer box to Band.

4   Click OK when you have finished.

Specifying data for the OLE object

You set data specifications for an OLE object in a DataWindow object on the Data property page in the Properties view. You can also use the Data property page to modify the data specifications you made in the wizard for a DataWindow object using the OLE presentation style.

What the data is for   When an OLE object is part of a DataWindow object, you can specify that some or all of the data the DataWindow object retrieves be transferred to the OLE object too. You can specify expressions instead of the actual columns so that the data is grouped, aggregated, or processed in some way before being transferred.

The way the OLE object uses the data depends on the server. For example, data transferred to Microsoft Excel is displayed as a spreadsheet. Data transferred to Microsoft Graph populates its datasheet, which becomes the data being graphed.

Some ActiveX controls do not display data, so you would not transfer any data to them. For an ActiveX control such as Visual Speller, you would use automation to process text when the user requests it.

Group By and Target Data boxes

Two boxes on the Data property page list data columns or expressions:

- **Group By**   Specifies how PowerBuilder groups the data it transfers to the OLE object. Aggregation functions in the target data expressions use the groupings specified here.

- **Target Data**   Specifies the data that you want to transfer to the OLE object.

Populating the Group By and Target Data boxes

If you are using the OLE presentation style, you populated the Group By and Target Data boxes when you created the DataWindow object. If you placed an OLE object in an existing DataWindow object, the boxes are empty. You use the browse buttons next to the Group By and Target Data boxes to open dialog boxes where you can select the data you want to use or modify your selections.

Modifying source data
You cannot modify the source data for the DataWindow object on the Data property page. Select Design>Data Source from the menu bar if you need to modify the data source.

❖ **To select or modify how data will be grouped in the OLE object:**

1   Click the Browse button next to the Group By box.

2   In the Modify Group By dialog box, drag one or more columns from the Source Data box to the Group By box.

     You can rearrange columns and specify an expression instead of the column name if you need to. For more information, see the next procedure.

❖ **To select or modify which data columns display in the OLE object:**

1   Click the Browse button next to the Target Data box.

2   In the Modify Target Data dialog box, drag one or more columns from the Source Data box to the Target Data box.

     The same source column can appear in both the Group By and Target Data box.

3   If necessary, change the order of columns by dragging them up or down within the Target Data box.

The order of the columns and expressions is important to the OLE server. You need to know how the server will use the data to choose the order.

4   Double-click an item in the Target Data box to specify an expression instead of a column.

In the Modify Expression dialog box, you can edit the expression or use the Functions or Columns boxes and the operator buttons to select elements of the expression. For example, you may want to specify an aggregation function for a column. Use the range `for object` if you use an aggregation function; for example, `sum (salary for object)`.

For more information about using operators, expressions, and functions, see the *DataWindow Reference*.

Example of a completed Data property page   This example of the Data property page specifies two columns to transfer to Microsoft Graph: city and salary. Graph expects the first column to be the categories and the second column to be the data values. The second column is an aggregate so that the graph will show the sum of all salaries in each city:
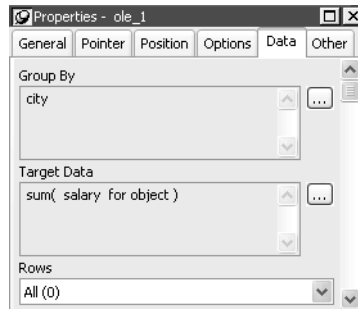
Specifying a value for Rows

The last setting on the Data property page specifies how the OLE object is associated with rows in the DataWindow object. The selection (all rows, current row, or page) usually corresponds with the band where you placed the OLE object, as explained in this table. If you used the OLE presentation style to create the DataWindow object, this setting does not display on the property page: the OLE object is always associated with all the rows in the DataWindow object.

Table 31-1: Associating an OLE object with rows in the DataWindow

Range of rows	When to use it
All	When the OLE object is in the summary, header, or footer band, or the foreground or background layer.
	Rows must be All and Layer must be Foreground or Background if you want the user to be able to activate the object.
	Target data for all rows is transferred to the object.
Current Row	When the OLE object is in the detail band.
	There is an instance of the OLE object for every row. Target data for a single row is transferred to each object.
	Because ActiveX controls must be in the foreground or background layer, they cannot be associated with individual rows in the detail band.
Page	When the OLE object is in the group header or trailer, foreground, or background.
	Target data for the rows on the current page is transferred to the OLE object.

Range of rows and activating the object

When the range of rows is Current Row or Page, the user *cannot* activate the OLE object. The user can see contents of the object in the form of an image presented by the server but cannot activate it.

If you want the user to activate the object, Rows must be set to All and Layer on the Position property page must be Foreground or Background.

Additional settings in the Properties view

The Options property page in the OLE object's Properties view has some additional settings. They are similar to the settings you can make for the OLE control in a window. These settings display on the General property page for OLE DataWindow objects. Table 31-2 describes the settings you can make.

Table 31-2: Settings on the OLE object's Options property page

Property	Effect
Client Name	A name for the OLE object that some server applications use in the title bar of their window.
	Corresponds to the ClientName DataWindow property.

Property	Effect
Activation	How the OLE object is activated. Choices are: • **Double click** When the user double-clicks on the object, the server application is activated. • **Manual** The object can only be activated programmatically. The object can always be activated programmatically, regardless of the Activation setting.
Contents	Whether the object in the OLE container is linked or embedded. The default is Any, which allows either method.
Display Type	What the OLE container displays. You can choose: • **Manual** Display a representation of the object, reduced to fit within the container. • **Icon** Display the icon associated with the data. This is usually an icon provided by the server application.
Link Update	When the object in the OLE container is linked, the method for updating link information. Choices are: • **Automatic** If the link is broken and PowerBuilder cannot find the linked file, it displays a dialog box in which the user can specify the file. • **Manual** If the link is broken, the object cannot be activated. You can let the user re-establish the link in a script using the UpdateLinksDialog function.

Previewing the DataWindow object

Previewing the DataWindow object lets you see how the OLE object displays the data from the DataWindow object. You can preview in the Preview view or in preview mode

❖ **To preview the DataWindow object with the OLE object in preview mode:**

1   Select File>Run/Preview from the menu bar, or click the Run/Preview button on the PowerBar.

2   Select Rows>Retrieve from the menu bar.

The DataWindow object retrieves rows from the database and replaces the initial presentation of the OLE object with an image of the data that the OLE server provides.

3   If you associated the OLE object with all rows, activate the OLE object by double-clicking on it.

Although you can edit the presentation or the data in the server, your changes do not affect the DataWindow object's data.

You cannot always activate the OLE object
If the OLE object is associated with individual rows in the detail band or with the page, you (and the user) cannot activate it; you can only view it.

4   Close the preview window.

Closing the preview window or the Preview view deactivates the OLE object.

Activating and editing the OLE object

In the Design view

PowerBuilder stores an initial presentation of the OLE object that it displays before data is retrieved and in newly inserted rows. When you activate the OLE object in the Design view, you are editing the initial presentation of the OLE object. Any changes you make and save affect only this initial presentation. After rows are retrieved and data transferred to the OLE object, an object built using the data replaces the initial presentation.

In preview or at execution time

PowerBuilder displays the initial presentation of the OLE object while it is retrieving rows and then replaces it with the retrieved data.

After you activate the OLE object in preview or at execution time, you can edit the presentation of the data. However, you cannot save these changes. The object is recreated whenever data from retrieved rows is transferred to the OLE object.

For more information, see "Activating OLE objects" on page 842.

Saving as a PSR
You can save the object with its data by saving the DataWindow object as a Powersoft report (PSR). Select File>Save As File or File>Save Rows As from the menu bar.

> ❖ **To activate the OLE object in the container in the Design view:**

- Select Open from the container's pop-up menu.

  Selecting Open from an ActiveX control's pop-up menu has no effect. ActiveX controls are always active.

Changing the object in the control

In the DataWindow painter, you can change or remove the OLE object in the OLE container object.

> ❖ **To delete the OLE object in the container:**

- Select Delete from the container's pop-up menu.

  The container object is now empty and cannot be activated.

> ❖ **To change the OLE object in the container:**

1 Select Insert from the container's pop-up menu.

   PowerBuilder displays the Insert Object dialog box.

2 Choose one of the tabs and specify the type of object you want to insert, as you did when you defined the object.

3 Click OK.

Using OLE columns in a DataWindow object

You can create OLE columns in a DataWindow object. An OLE column allows you to:

- Store blob (binary large-object) data, such as Microsoft Excel worksheets or Microsoft Word documents, in the database

- Retrieve blob data from a database into a DataWindow object

- Use an OLE server application, such as Microsoft Excel or Microsoft Word, to modify the data

- Store the modified data back in the database

You can modify the document in the server, then update the data in the DataWindow object. When the database is updated, the OLE column, which contains the modified document, is stored in the database.

Database support for OLE columns
If your database supports a blob datatype, then you can implement OLE columns in a DataWindow object. The name of the datatype that supports blob data varies. For information on which datatypes your DBMS supports, see your DBMS documentation.

Creating an OLE column

This section describes how to create an OLE column in a DataWindow object. The steps are illustrated using a table that you can create in the Database painter. It must contain at least two columns, id and object:

• The id column is an integer and serves as the table's key.

• The object column is a blob datatype and contains OLE objects associated with several OLE servers.

❖ **To create the database table:**

1   In the Database painter, create a table to hold the blob (binary large-object) data.

The table must have at least two columns: a key column and a column with the blob datatype. The actual datatype you choose depends on your DBMS. For example, in Adaptive Server Anywhere, choose long binary as the datatype for the blob column. For information about datatypes, see your DBMS documentation.

2   Define the blob columns as allowing NULLs (this allows you to store a row that does not contain a blob).

Adding a blob column
to the DataWindow
object

The following procedure describes how to add a blob column to a DataWindow object.

❖ **To add a blob column to a new DataWindow object:**

1 Create a new DataWindow object.

2 Specify the table containing the blob as the data source for the DataWindow object.

Be sure to include the key column in the data source. You cannot include the blob column in the data source; if you try, a message tells you that its datatype requires the use of an embedded SQL statement. You add the blob column later in the DataWindow painter workspace. (If you use Quick Select, the blob column is not listed in the dialog box.)

3 Select Insert>Control>OLE Database Blob and click where you want the blob column in the Design view.

The Database Binary/Text Large Object dialog box displays:
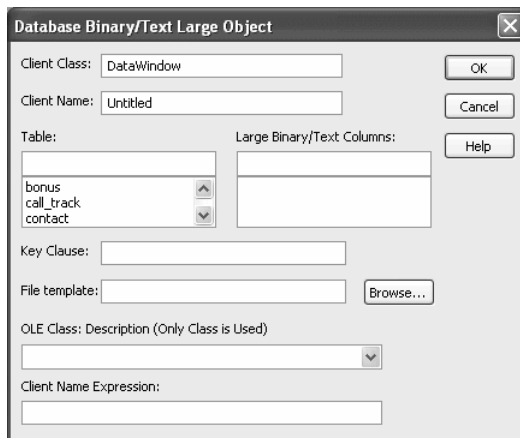
Setting properties for
the blob column

The following procedure describes the properties you need to set for the blob column.

❖ **To set properties for a blob column:**

1 (Optional) Enter the client class in the Client Class box. The default is DataWindow.

This value is used in some OLE server applications to build the title that displays at the top of the server window.

2   (Optional) Enter the client name in the Client Name box. The default is Untitled.

This value is used in some OLE server applications to build the title that displays in the title bar of the server window.

3   In the Table box, select the database table that contains the blob database column you want to place in the DataWindow object.

The names of the columns in the selected table display in the Large Binary/Text Columns list.

4   In the Large Binary/Text Columns box, select the column that contains the blob datatype from the list.

5   If necessary, change the default key clause in the Key Clause box.

PowerBuilder uses the key clause to build the WHERE clause of the SELECT statement used to retrieve and update the blob column in the database. It can be any valid WHERE clause.

Use colon variables to specify DataWindow columns. For example, if you enter this key clause:

```
id = :id
```

the WHERE clause will be:

```
WHERE id = :id
```

6   Identify the OLE server application by doing one of the following:

•   If you always want to open the same file in the OLE server application, enter the name of the file in the File Template box.

For example, to specify a particular Microsoft Word document, enter the name of the *DOC* file. If the file is not on the current path, enter the fully qualified name.

Use the Browse button to find the file
If you do not know the name of the file you want to use, click the Browse button to display a list of available files. Select the file you want from the resulting window.

- If you do not want to open the same file each time, select an OLE server application from the OLE Class: Description drop-down list.

When the server does not match the OLE blob data
If you specify a server that does not match the OLE blob object or if your database contains objects belonging to different servers, the OLE mechanism can usually handle the situation. It looks for the server specified in the object and starts it instead of the server you specified.

7 Enter text or an expression that evaluates to a string in the Client Name Expression box.

The server might use this expression in the title of the window in the OLE server application. The expression you specify can identify the current row in the DataWindow object.

Use an expression to make sure the name is unique
To make sure the name is unique, you should use an expression. For example, you might enter the following expression to identify a document (where id is the integer key column):

```
"Document " + String(id)
```

8 Click OK.

PowerBuilder closes the dialog box. The blob column is represented by a box labeled Blob in the Design view.

9 Save the DataWindow object.

The following screenshot shows what a completed Definition page for a Blob object in a table called ole looks like in the Properties view:
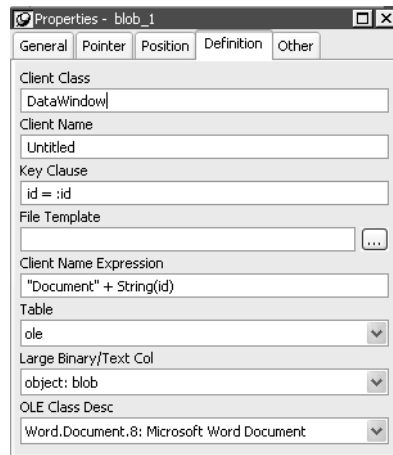
Making the blob column visible

If the blob column is invisible in the DataWindow object until you activate the OLE server, you can make it easy to find the blob column by adding a border to the object.

Previewing an OLE column

Before using the DataWindow object in an application, you should preview it in the Preview view or in preview mode to see how it works.

❖ **To preview an OLE column in preview mode:**

1 Select File>Run/Preview from the menu bar and select the DataWindow object.

2 Click the Insert Row button.

PowerBuilder adds a blank row.

3 In the blank row, enter a value in the key column.

4 Double-click the column that contains the blob datatype.

The OLE server application starts and displays the file you specified in the File Template box, or an empty workspace if you specified only the OLE server name.

5   Review the file in the OLE server application and make changes if you
     want.

     When you use an OLE column to access an OLE server application, the
     server application adds an item to its File menu that allows you to update
     the data in the server application and in the client (the DataWindow
     object). The text of the menu item depends on the OLE server application.
     In most applications, it is Update.

6   Select the menu item in the OLE server that updates the OLE client with
     the modifications.

     In the example, you would select Update from the File menu in Microsoft
     Word. The OLE server application sends the updated information to the
     DataWindow object.

7   Close the file in the server application (typically by selecting Close from
     the File menu).

8   To save the blob data in the database, click the Save Changes button in the
     PainterBar.

     The new row, including the key value and the blob, is stored in the
     database..

Later, after you retrieve the rows from the database, you can view and edit the
blob by double-clicking it, which invokes the OLE server application and
opens the stored document. If you make changes and then update the database,
all the modified OLE columns are stored in the database.

P A R T   7   # Running Your Application

This part describes the ways in which your application can be run. The first chapter describes how to run your application from within PowerBuilder: in debug mode, where you can set breakpoints and examine the state of your application as it executes, and in regular mode, where the application runs until you stop it or an error occurs. The second chapter describes how to collect trace information so that you can analyze performance and evaluate your application's structure. The third chapter describes how to build your application for distribution to users.

Debugging and Running Applications

About this chapter

This chapter describes how to debug and run an application in PowerBuilder. The chapter also lists the errors that can occur at runtime.

Contents

Overview of debugging and running applications

After you build all or part of an application and compile and save its objects, you can run the application. The PowerBuilder development environment provides two ways to run an application: in debug mode and in regular mode.

Debug mode

In debug mode, you can insert breakpoints (stops) in scripts and functions, single-step through code, and display the contents of variables to locate logic errors that will result in errors at runtime.

Regular mode

In regular mode, the application responds to user interaction and runs until the user stops it or until a runtime error occurs. This is the mode you and your users will use to run the completed application.

You can also collect trace information while you run your application in regular mode, then use the trace data to profile your application. For more information, see Chapter 33, "Tracing and Profiling Applications."

This chapter describes:

- Running applications in debug mode

- Running applications in regular mode

Debugging an application

Sometimes an application does not behave the way you think it will. Perhaps a variable is not being assigned the value you expect, or a script does not perform as desired. In these situations, you can examine your application by running it in debug mode.

When you run the application in debug mode, PowerBuilder stops execution before it executes a line containing a breakpoint (stop). You can then step through the application and examine its state.

❖ **To debug an application:**

1 Open the debugger.

2 Set breakpoints at places in the application where you have a problem.

3 Run the application in debug mode.

4 When execution is suspended at a breakpoint, look at the values of variables, examine the properties of objects in memory and the call stack, or change the values of variables.

5 Step through the code line by line.

6 As needed, add or modify breakpoints as you run the application.

7 When you uncover a problem, fix your code and run it in the debugger again.

Debugging distributed applications
You can debug a PowerBuilder component deployed to EAServer. For more information, see the chapter on building EAServer components in *Application Techniques*.

Starting the debugger

❖ **To open the debugger:**

• Do one of the following:

• In the System Tree, highlight a target and select Debug from the pop-up menu

• Click the Debug or Select and Debug button on the PowerBar

• Select Run>Debug or Run>Select and Debug from the menu bar

The Debug button opens the debugger for the last debugged or run target. The name of the current target is displayed in the Debug button tool tip. The Select and Debug button opens a dialog box that lets you select the target to be debugged.

Views in the debugger

The debugger contains several views. Each view shows a different kind of information about the current state of your application or the debugging session. Table 32-1 summarizes what each view shows and what you can do from that view.

Table 32-1: Views in the debugger

View	What it shows	What you can do
Breakpoints	A list of breakpoints with indicators showing whether the breakpoints are currently active or inactive	Set, enable, disable, and clear breakpoints, set a condition for a breakpoint, and show source for a breakpoint in the Source view.
Call Stack	The sequence of function calls leading up to the function that was executing at the time of the breakpoint, shown as the script and line number from which the function was called	Examine the context of the application at any line in the call stack.
Instances	Instances of remote objects and their current status	Change the context of the debugging session to a different instance. This view has content only if you are debugging a remote component.
Objects in Memory	An expandable list of objects currently in memory	View the names and memory locations of instances of each memory object and property values of each instance. This view is not used if you are debugging a remote component.
Source	The full text of a script	Go to a specific line in a script, find a string, open another script, including ancestor and descendent scripts, manage breakpoints, and use TipWatch and QuickWatch.

View	What it shows	What you can do
Source Browser	An expandable hierarchy of objects in your application	Select any script in your application and display it in the Source view.
Source History	A list of the scripts that have been displayed in the Source view	Select any script in the Source History and display it in the Source view.
Variables	An expandable list of all the variables in scope	Select which kinds of variables are shown in the view, change the value of a variable, and set a breakpoint when a variable changes. You cannot change the value of a variable in a remote component.
Watch	A list of variables you have selected to watch as the application proceeds	Change the value of a variable, set a breakpoint when a variable changes, and add an arbitrary expression to the Watch view.

Changing Variable views

The default debugger layout contains a separate view for each variable type in a stacked pane. You can combine two or more Variables views in a single pane. For example, you might want to combine local and global variables in a single view that you keep at the top of the stacked pane.

❖ **To display multiple variable types in a single view:**

1 Display the pop-up menu for a pane that contains a Variables view you want to change.

2 Click the names of the variable types you want to display.

   A checkmark displays next to selected variable types. The pop-up menu closes each time you select a variable type or clear a checkmark, so you need to reopen the menu to select an additional variable type.

   When you select or clear variable types, the tab for the pane changes to show the variable types displayed on that pane.

Setting breakpoints

A breakpoint is a point in your application code where you want to interrupt the normal execution of the application while you are debugging. If you suspect a problem is occurring in a particular script or function call, set a breakpoint at the beginning of the script or at the line where the function is called.

When you close the debugger, any breakpoints you set are written to your PowerBuilder initialization file and are available when you reopen the debugger.

Setting a simple breakpoint

This procedure describes setting a breakpoint in the Source view in the debugger. You can also set a breakpoint by selecting Add Breakpoint from the pop-up menu in the Script view when you are not running the debugger.

❖ **To set a breakpoint on a line in a script:**

1   Display the script in a Source view and place the cursor where you want to set the breakpoint.

For how to change the script shown in the Source view, see "Using the Source view" on page 878.

2   Double-click the line or select Add Breakpoint from the pop-up menu.

PowerBuilder sets a breakpoint and a red circle displays at the beginning of the line. If you select a line that does not contain executable code, PowerBuilder sets the breakpoint at the beginning of the next executable statement.

Setting special breakpoints

Breakpoints can be triggered when a statement has been executed a specified number of times (an occasional breakpoint), when a specified expression is true (a conditional breakpoint), or when the value of a variable changes.

You use the Edit Breakpoints dialog box to set and edit occasional and conditional breakpoints. You can also use it to set a breakpoint when the value of a variable changes. The Edit Breakpoints dialog box opens when you:

•   Click the Edit Stop button on the PainterBar

•   Select Breakpoints from the pop-up menu in the Source, Variables, Watch, or Breakpoints view

•   Select Edit>Breakpoints from the menu bar

•   Double-click a line in the Breakpoints view

Setting occasional and conditional breakpoints

If you want to check the progress of a loop without interrupting execution in every iteration, you can set an occasional breakpoint that is triggered only after a specified number of iterations. To specify that execution stops only when conditions you specify are met, set a conditional breakpoint. You can also set both occasional and conditional breakpoints at the same location.

- **If you specify an occurrence**   Each time PowerBuilder passes through the specified location, it increments a counter by one. When the counter reaches the number specified, it triggers the breakpoint and resets the counter to zero.

- **If you specify a condition**   Each time PowerBuilder passes through the specified location, it evaluates the expression. When the expression is true, it triggers the breakpoint.

- **If you specify both an occurrence and a condition**   Each time PowerBuilder passes through the specified location, it evaluates the expression. When the expression is true, it increments the counter. When the counter reaches the number specified, it triggers the breakpoint and resets the counter to zero.

    For example, if you specify an occurrence of 3 and the condition `notisNull(val)`, PowerBuilder checks whether val is NULL each time the statement is reached. The breakpoint is triggered on the third occurrence of a non-NULL val, then again on the sixth occurrence, and so forth.

❖ **To set an occasional or conditional breakpoint:**

1   On the Location tab in the Edit Breakpoints dialog box, specify the script and line number where you want the breakpoint.

    You can select an existing location or select New to enter a new location.

Set a simple breakpoint first
You must specify a line that contains executable code. Set a simple breakpoint on the line before opening the Edit Breakpoints dialog box to ensure the format and line number are correct.

---

2   Specify an integer occurrence, a condition, or both.

The condition must be a valid boolean PowerScript expression (if it is not, the breakpoint is always triggered). PowerBuilder displays the breakpoint expression in the Edit Breakpoints dialog box and in the Breakpoints view. When PowerBuilder reaches the location where the breakpoint is set, it evaluates the breakpoint expression and triggers the breakpoint only when the expression is true.

Setting a breakpoint when a variable changes

You can interrupt execution every time the value of a variable changes. The variable must be in scope when you set the breakpoint.

❖ **To set a breakpoint when a variable changes:**

- Do one of the following:

  - Select the variable in the Variables view or Watch view and select Break on Change from the pop-up menu.

  - Drag the variable from the Variables view or Watch view to the Breakpoints view.

  - Select New on the Variable tab in the Edit Breakpoints dialog box and specify the name of a variable in the Variable box.

The new breakpoint displays in the Breakpoints view and in the Edit Breakpoints dialog box if it is open. PowerBuilder watches the variable at runtime and interrupts execution when the value of the variable changes.

Disabling and clearing breakpoints

If you want to bypass a breakpoint for a specific debugging session, you can disable it and then enable it again later. If you no longer need a breakpoint, you can clear it.

❖ **To disable a breakpoint:**

- Do one of the following:

  - Click the red circle next to the breakpoint in the Breakpoints view or Edit Breakpoints dialog box

  - Select Disable Breakpoint from the pop-up menu in the Source view

  - Select Disable from the pop-up menu in the Breakpoints view

The red circle next to the breakpoint is replaced with a white circle.

You can enable a disabled breakpoint from the pop-up menus or by clicking the white circle.

> **Disabling all breakpoints**
> To disable all breakpoints, select Disable All from the pop-up menu in the Breakpoints view.

❖ **To clear a breakpoint:**

- Do one of the following:

  - Double-click the line containing the breakpoint in the Source view

  - Select Clear Breakpoint from the pop-up menu in the Source view

  - Select Clear from the pop-up menu in the Breakpoints view

  - Select the breakpoint in the Edit Breakpoints dialog box and select Clear

  The red circle next to the breakpoint disappears.

> **Clearing all breakpoints**
> To clear all breakpoints, select Clear All in the Edit Breakpoints dialog box or from the pop-up menu in the Breakpoints view.

Running in debug mode

After you have set some breakpoints, you can run the application in debug mode. The application executes normally until it reaches a statement containing a breakpoint. At this point it stops so that you can examine the application. After you do so, you can single-step through the application, continue execution until execution reaches another breakpoint, or stop the debugging run so that you can close the debugger and change the code.

❖ **To run an application in debug mode:**

1 If necessary, open the debugger by clicking the Debug or Select and Debug button.

  The Debug button opens the debugger for the target you last ran or debugged. Use the Select and Debug button if you want to debug a different target in the workspace.

2   Click the Start button in the PainterBar or select Debug>Start from the menu bar.

The application starts and runs until it reaches a breakpoint. PowerBuilder displays the debugger, with the line containing the breakpoint displayed in the Source view. The yellow arrow cursor indicates that this line contains the next statement to be executed. You can now examine the application using debugger views and tools.

For more information, see "Examining an application at a breakpoint" on page 872 and "Stepping through an application" on page 879.

❖   **To continue execution from a breakpoint:**

•   Click the Continue button in the PainterBar or select Debug>Continue from the menu bar

Execution begins at the statement indicated by the yellow arrow cursor and continues until the next breakpoint is hit or until the application terminates normally.

❖   **To terminate a debugging run at a breakpoint:**

•   Click the Stop Debugging button in the PainterBar or select Debug>Stop from the menu bar

PowerBuilder resets the state of the application and all the debugger views to their state at the beginning of the debugging run. You can now begin another run in debug mode, or close the debugger.

Cleaning up

When you terminate a debugging run or close the debugger without terminating the run, PowerBuilder executes the application's close event and destroys any objects, such as autoinstantiated local variables, that it would have destroyed if the application had continued to run and exited normally.

Examining an application at a breakpoint

When an application is suspended at a breakpoint, use QuickWatch, TipWatch, and the Variables, Watch, Call Stack, and Objects in Memory views to examine its state.

About icons used in debugging views
The Variables, Watch, and Objects in Memory views use many of the icons used in the PowerBuilder Browser as well as some additional icons: I represents an Instance; F, a field; A, an array; and E, an expression.

Examining variable values

The debugger provides three different ways to examine the values of variables: TipWatch, QuickWatch, and the Variables view.

TipWatch

TipWatch is a quick way to get the current value of a variable of a simple datatype. When execution stops at a breakpoint, you can place the edit cursor over a variable in the Source view to display a pop-up tip window that shows the current value of that variable. You can also select a simple expression to display its current value.

TipWatch has some limitations: if the variable you select is an object type, the tip window shows only an internal identifier. For array types, it shows { . . . } to indicate that more information is available. To show complete information for object type and array type variables, use QuickWatch instead.

TipWatch does not evaluate function, assignment, or variable value modification expressions. If TipWatch cannot parse the string you select, the pop-up window does not display.

Remote debugging
When you are debugging a remote component, Tip Watch does not evaluate expressions or indirect variables.

QuickWatch

QuickWatch provides the current value of simple variables and detailed information about object variables, including the values of all fields in the variable. QuickWatch can also evaluate function expressions, and you can use it to change the values of variables, evaluate expressions, and add variables and expressions to the Watch view.

Exercise caution when evaluating expressions
QuickWatch evaluates all kinds of expressions, including functions, in local debugging. If you select a function and activate QuickWatch, the function is executed. This may have unexpected results. For example, if you select `dw_1.print()` and activate QuickWatch, the DataWindow is printed.

❖ **To open the QuickWatch dialog box:**

- When execution stops at a breakpoint, move the edit cursor to a variable or select an expression in the Source view, and do one of the following:

  - Select QuickWatch from the Debug or pop-up menu

  - Press Shift+F9

❖ **To change the value of a variable from the QuickWatch dialog box:**

1   Select an item in the tree view and do one of the following:

   - Click Change Value

   - Double-click the tree view item

2   In the Modify Variable dialog box, type a new value for the variable in the New Value box, or select the Null check box to set the value of the variable to null, and click OK.

3   Close the QuickWatch dialog box and continue debugging the application with the variable set to the new value.

❖ **To evaluate an expression in the QuickWatch dialog box and add it to the Watch view:**

1   Change the variable or expression in the Expression box.

2   Click Reevaluate to display the new value in the tree view.

3   (Optional) Click Add Watch to add the expression to the Watch view.

Remote debugging
When you are debugging a remote component, expressions and indirect variables are not evaluated, and you cannot modify variable values.

Using Variables views

Each Variables view shows one or more types of variables in an expandable outline. Double-click the variable names or click on the plus and minus signs next to them to expand and contract the hierarchy. If you open a new Variables view, it shows all variable types.

Table 32-2: Variable views in the debugger

Variable type	What the Variables view shows
Local	Values of variables that are local to the current script or function
Global	Values of all global variables defined for the application and properties of all objects (such as windows) that are open
Instance	Properties of the current object instance (the object to which the current script belongs) and values of instance variables defined for the current object
Parent	Properties of the parent of the current instance
Shared	Objects, such as application, window, and menu objects, that have been opened and the shared variables associated with them

About Instance and Parent variables

In the following illustration, an application has stopped at a breakpoint in the script for the Clicked event for the Close menu item on a frame's File menu. The Instance Variables view shows the properties of the current instance of the Close menu item. The Parent Variables view shows the properties of its parent, an instance of the File menu. Navigating through the hierarchy in the Global Variables view shows the same objects.
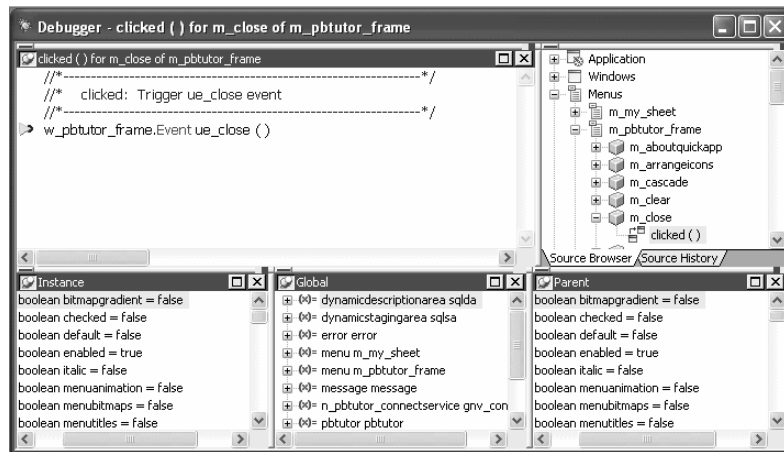
Watching variables and expressions

The Watch view lets you monitor the values of selected variables and expressions as the application runs.

If the variable or expression is in scope, the Watch view shows its value. Empty quotes indicate that the variable is in scope but has not been initialized. A pair of glasses in the Watch view indicates that the variable or expression is not in scope.
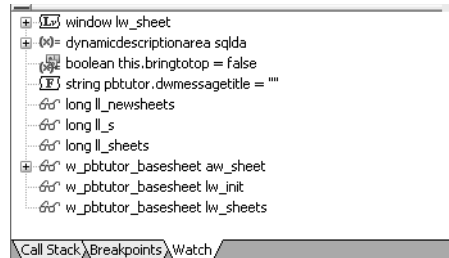
Setting variables and expressions in the Watch view

You can select variables you want to watch as the application runs by copying them from a Variables view. You can also set a watch on any PowerScript expression. When you close the debugger, any watch variables and expressions you set are saved.

Using QuickWatch
You can also add variables and expressions to the Watch view from the QuickWatch dialog box. See "QuickWatch" on page 872.

❖ **To add a variable to the Watch view:**

1   Select the variable in the Variables view.

2   Do one of the following:

   •   Drag the variable to the Watch view

   •   Click the Add Watch button on the PainterBar

   •   Select Debug>Add Watch from the menu bar

   PowerBuilder adds the variable to the watch list.

❖ **To add an expression to the Watch view:**

1 Select Insert from the pop-up menu.

2 Type any valid PowerScript expression in the New Expression dialog box and click OK.

PowerBuilder adds the expression to the watch list.

❖ **To edit a variable in the Watch view:**

1 Select the variable you want to edit.

2 Double-click the variable, or select Edit Variable from the pop-up menu.

3 Type the new value for the variable in the Modify Variable dialog box and click OK.

❖ **To edit an expression in the Watch view:**

1 Select the expression you want to edit.

2 Double-click the expression, or select Edit Expression from the pop-up menu.

3 Type the new expression in the Edit Expression dialog box and click OK.

❖ **To clear variables and expressions from the Watch view:**

1 Select the variable or expression you want to delete.

2 Do one of the following:

- Select Clear from the pop-up menu
- Click the Remove Watch button on the PainterBar
- Select Debug>Remove Watch from the menu bar

❖ **To clear all variables and expressions from the Watch view:**

- Select Clear All from the pop-up menu

Monitoring the call stack

The Call Stack view shows the sequence of function calls leading up to the script or function that was executing at the time of the breakpoint. Each line in the Call Stack view displays the name of the script and the line number from which the call was made. The yellow arrow shows the script and line where execution was suspended.

You can examine the context of the application at any line in the call stack.

❖ **To show a different context from the Call Stack view:**

1   Select a line in the Call Stack view.

2   Do one of the following:

   •   Double-click the line

   •   Select Set Context from the pop-up menu

   •   Drag the line into the Source view

   A green arrow indicates the script that you selected. The Source view
   shows the script and line number you selected, and the Variables and
   Watch views show the variables and expressions in scope in that context.
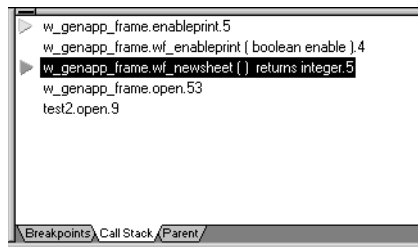
Examining objects in memory

The Objects in Memory view shows an expandable list of objects currently in
memory. Double-click the name of an object or click the plus sign next to it to
view the names and memory locations of instances of each object and property
values of each instance.
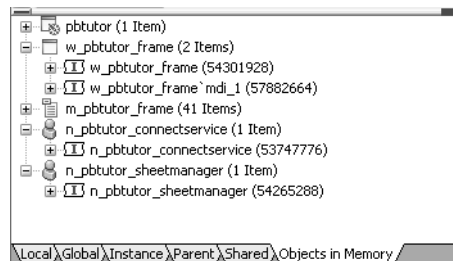
Using the Source view

The Source view displays the full text of a script. As you run or step through the application, the Source view is updated to show the current script with a yellow arrow indicating the next statement to be executed.

Multiple Source views
You can open more than one source view. If there are multiple source views open, only the first one opened is updated to show the current script when the context of an application changes.

Copying from the Source view
When text is selected in the Source view, you can select Copy from the pop-up menu in the Source view to copy the string to the clipboard. You can then paste the string into another dialog box to search for the string, insert a watch, or add a conditional breakpoint.

Changing the Source view
From the pop-up menu, you can navigate backward and forward through the scripts that have been opened so far, open ancestor and dependent scripts, and go to a specific line in the current script. There are several other ways to change the script from other views or from the menu bar.

❖ **To change the script displayed in a Source view:**

- Do one of the following:

    - Drag the name of a script to the Source view from the Call Stack, Source Browser, or Source History views

    - Select a line and then select Open Source from the pop-up menu in the Breakpoints, Source Browser, or Source History views

    - Select Edit>Select Script from the menu bar

❖ **To find a specified string in the Source view:**

1   Select Find from the pop-up menu, or select Edit>Find from the menu bar.

    The Find Text dialog box opens.

2   Type the string in the Find box and check the search options you want.

Using the Source Browser view

The Source Browser shows all the objects in your application in an expandable hierarchy. It provides a view of the structure of the application and a quick way to open any script in the Source view.

❖ **To open a script from the Source Browser:**

1   Double-click the object that the script belongs to or click the plus sign next to the object to expand it.

2   Do one of the following:

   • Double-click the script

   • Select the script and select Open Source from the pop-up menu

   • Drag the script onto a Source view

When you double-click or select Open Source, a new Source view opens if there was none open. If several Source views are open, the script displays in the view that was used last.

Using the Source History view

The Source History view lists all the scripts that have been opened in the current debugging session. Use the same techniques as in the Source Browser view to display a selected script in the Source view.

Source History limit
The Source History view shows up to 100 scripts and is *not* cleared at the end of each debugging run. It is cleared when you close the debugger, or you can clear the list from the pop-up menu.

Stepping through an application

When you have stopped execution at a breakpoint, you can use several commands to step through your application code and use the views to examine the effect of each statement. As you step through the code, the debugger views change to reflect the current context of your application and a yellow arrow cursor indicates the next statement to be executed.

Updating the Source view
When the context of your application changes to another script, the Source view is updated with the new context. If you have multiple Source views open, only the first one opened is updated.

Single-stepping
through an application

You can use either Step In or Step Over to step through an application one statement at a time. They have the same result except when the next statement contains a call to a function. Use Step In if you want to step into a function and examine the effects of each statement in the function. Use Step Over to execute the function as a single statement.

❖ **To step through an application entering functions:**

• Click the Step In button in the PainterBar, or select Debug>Step In from the menu bar.

❖ **To step through an application without entering functions:**

• Click the Step Over button in the PainterBar, or select Debug>Step Over from the menu bar.

Using shortcut keys
To make it easier to step through your code, the debugger has standard keyboard shortcuts for Step In, Step Out, Step Over, Run To Cursor, and Continue. If you prefer to use different shortcut key combinations, select Tools>Keyboard Shortcuts to define your own.

Stepping out of a
function

If you step into a function where you do not need to step into each statement, use Step Out to continue execution until the function returns.

❖ **To step out of a function:**

• Click the Step Out button in the PainterBar, or select Debug>Step Out from the menu bar.

Stepping through
multiple statements

As you step through an application, you might reach sections of code that you are not interested in examining closely. The code might contain a large loop, or it might be well-tested code that you are confident is free of errors. You can use Run To Cursor to select a statement further down in a script or in a subsequent script where you want execution to stop.

❖ **To step through multiple statements:**

1 Click on the line in the script where you want to resume single stepping.

2 Click the Run To Cursor button in the PainterBar, or select Debug>Run To Cursor from the menu bar.

PowerBuilder executes all intermediate statements and the yellow arrow cursor displays at the line where you set the cursor.

Bypassing statements

You can use Set Next Statement to bypass a section of code that contains a bug, or to test part of an application using specific values for variables. Execution continues from the statement where you place the cursor. Be cautious when you use Set Next Statement, because results may be unpredictable if, for example, you skip code that initializes a variable.

❖ **To set the next statement to be executed:**

1   Click on the line in the script where you want to continue execution.

2   Click the Set Next Statement button in the PainterBar, or select Debug>Set Next Statement from the menu bar.

3   If necessary, change the values of variables.

4   Continue execution using Continue, Step In, or Step Over.

    If you select Continue, PowerBuilder begins execution at the statement you specified and continues to the next breakpoint. If you select Step In or Step Over, PowerBuilder sets the next statement and displays the yellow arrow cursor at the line where you set the cursor.

Changing a variable's value

As you step through the application, you can change the values of variables that are in scope. You may want to do this to examine different flows through the application, to simulate a condition that is difficult to reach in normal testing, or if you are skipping code that sets a variable's value.

Limitations
You cannot change the values of enumerated variables, and you cannot change the value of any variable when you are debugging a remote component.

❖ **To change the value of a variable:**

1   Select the variable in the Variables view or the Watch view.

2   From the pop-up menu, select Edit Variable.

3   Type a value for the variable or select the Null check box and click OK.

    The value you enter must conform to the type of the variable. If the variable is a string, do not enclose the string in quotes. When you continue execution, the new value is used.

Fixing your code

If you find an error in a script or function during a debugging session, you must close the debugger before you fix it. After you have fixed the problem, you can reopen the debugger and run the application again in debug mode. The breakpoints and watchpoints set in your last session are still defined.

Debugging windows opened as local variables

One way to open a window is by declaring a local variable of type window and opening it through a string. For example:

```
window mywin
string named_window
named_window = sle_1.Text
Open(mywin, named_window)
```

The problem   Normally, you cannot debug windows opened this way after the script ends because the local variable (mywin in the preceding script) goes out of scope when the script ends.

The solution   If you want to debug windows opened this way, you can declare a global variable of type window and assign it the local variable. If, for example, GlobalWindow is a global window of type window, you could add the following line to the end of the preceding script:

```
GlobalWindow = mywin
```

You can look at and modify the opened window through the global variable. When you have finished debugging the window, you can remove the global variable and the statement assigning the local to the global.

Just-in-time debugging

If you are running your application in regular mode (using the Run button) and you notice that the application is behaving incorrectly, just-in-time debugging lets you switch to debug mode without terminating the application.

When you open the debugger while running an application, the application *does not* stop executing. The Source, Variables, Call Stack, and Objects in Memory views are all empty because the debugger does not have any context. To suspend execution and examine the context in a problem area, open an appropriate script and set breakpoints, then initiate the action that calls the script.

If just-in-time debugging is enabled and a system error occurs while an application is running in regular mode, the debugger opens automatically, showing the context where the error occurred.

You can also use the DebugBreak function to break into the debugger.

You must enable just-in-time debugging before you run your application to take advantage of this feature.

❖ **To enable just-in-time debugging:**

1 Select Tools>System Options.

2 Check the Just In Time Debugging check box and click OK.

❖ **To debug an application while running in regular mode:**

1 Enable just-in-time debugging.

2 Run the application.

3 Click the PowerBuilder button on the Windows Taskbar.

4 Click the Debug button in the dialog box that displays.

5 Open a script in the Source view and set breakpoints.

The application is suspended when it hits a breakpoint and the Source, Variable, Call Stack, and Objects in Memory views show the current context. You can now debug the application.

Running an application

When the application seems to be working correctly, you are ready to run it in regular mode. In regular mode, the application responds to user interaction and continues to run until the user exits the application or a runtime error occurs. You can rely on the default runtime error reporting by PowerBuilder or write a script that specifies your own error processing. You can also generate a diagnostic trace of your application's execution.

For how to analyze your application's logic and performance, see Chapter 33, "Tracing and Profiling Applications."

Running the application

❖ **To run an application:**

• Do one of the following:

• In the System Tree, highlight a target and select Run from the pop-up menu

- Click the Run or Select and Run button on the PowerBar

- Select Run>Run or Run>Select and Run from the menu bar

The Run button runs the last run or debugged target. The name of the current target is displayed in the Run button tool tip. The Select and Run button opens a dialog box that lets you select the target to run.

PowerBuilder becomes minimized and a button displays on the Taskbar. Your application executes.

❖ **To stop a running application:**

- End the application normally, or double-click the minimized PowerBuilder button or icon to open a response window from which you can terminate the application.

Handling errors at runtime

A serious error at runtime (such as attempting to access a window that has not been opened) will trigger the SystemError event in the Application object if you have not added exception handling code to take care of the error.

If there is no SystemError script

If you do not write a SystemError script to handle these errors, PowerBuilder displays a message box containing the following information:

- The number and text of the error message

- The line number, event, and object in which the error occurred

There is also an OK button that closes the message box and stops the application.

If there is a SystemError script

If there is a script for the SystemError event, PowerBuilder executes the script and does not display the message box. Whether or not you have added TRY/CATCH blocks to your code to trap errors, it is a good idea to build an application-level script for the SystemError event to trap and process any runtime errors that have not been handled, as described in "Using the Error object" next.

For more information about handling exceptions, see *Application Techniques*.

Using the Error object

In the script for the SystemError event, you can access the built-in Error object to determine which error occurred and where it occurred. The Error object contains the properties shown in Table 32-3.

Table 32-3: Properties of the Error object

Property	Data type	Description
Number	Integer	Identifies the PowerBuilder error.
Text	String	Contains the text of the error message.
WindowMenu	String	Contains the name of the window or menu in which the error occurred.
Object	String	Contains the name of the object in which the error occurred. If the error occurred in a window or menu, the Object property will be the same as the WindowMenu property
ObjectEvent	String	Contains the event for which the error occurred.
Line	Integer	Identifies the line in the script at which the error occurred.

Defining your own Error object
You can customize your own version of the Error object by defining a class
user object inherited from the built-in Error object. You can add properties and
define object-level functions for your Error object to allow for additional
processing. In the Application painter, you can then specify that you want to
use your user object inherited from Error as the global Error object in your
application. For more information, see "Building a standard class user object"
on page 365.

Runtime error
numbers

Table 32-4 lists the runtime error numbers returned in the Number property of
the Error object and the meaning of each number:

Table 32-4: PowerBuilder runtime errors

Number	Meaning
1	Divide by zero.
2	Null object reference.
3	Array boundary exceeded.
4	Enumerated value is out of range for function.
5	Negative value encountered in function.
6	Invalid DataWindow row/column specified.
7	Unresolvable external when linking reference.
8	Reference of array with null subscript.
9	DLL function not found in current application.
10	Unsupported argument type in DLL function.
11	Object file is out of date and must be converted to current version.
12	DataWindow column type does not match GetItem type.

Number	Meaning
13	Unresolved property reference.
14	Error opening DLL library for external function.
15	Error calling external function *name*.
16	Maximum string size exceeded.
17	DataWindow referenced in DataWindow object does not exist.
18	Function does not return value.
19	Cannot convert *name* in Any variable to *name*.
20	Database command not successfully prepared.
21	Bad runtime function reference.
22	Unknown object type.
23	Cannot assign object of type *name* to variable of type *name*.
24	Function call does not match its definition.
25	Double or Real expression has overflowed.
26	Field *name* assignment not supported.
27	Cannot take a negative to a noninteger power.
28	VBX Error: *name*.
29	Nonarray expected in ANY variable.
30	External object does not support data type *name*.
31	External object data type *name* not supported.
32	Name not found calling external object function *name*.
33	Invalid parameter type calling external object function *name*.
34	Incorrect number of parameters calling external object function *name*.
35	Error calling external object function *name*.
36	Name not found accessing external object property *name*.
37	Type mismatch accessing external object property *name*.
38	Incorrect number of subscripts accessing external object property *name*.
39	Error accessing external object property *name*.
40	Mismatched ANY data types in expression.
41	Illegal ANY data type in expression.
42	Specified argument type differs from required argument type at runtime in DLL function *name*.
43	Parent object does not exist.
44	Function has conflicting argument or return type in ancestor.
45	Internal table overflow; maximum number of objects exceeded.
46	Null object reference cannot be assigned or passed to a variable of this type.
47	Array expected in ANY variable.

Number	Meaning
48	Size mismatch in array-to-object conversion.
49	Type mismatch in array-to-object conversion.
50	Distributed Service Error: *name*.
51	Bad argument list for function/event: *name*.
52	Distributed Communications Error: *name*.
53	The server *name* could not be located. It was probably not started.
54	The server *name* is rejecting new messages. It is in the process of shutting down.
55	The request caused an abnormal termination. The connection has been closed.
56	A message was not fully transmitted.
57	This connection object is not connected to a server.
58	Object instance does not exist.
59	Invalid column range.
60	Invalid row range.
61	Invalid conversion of *number* dimensional array to object.
62	The server *name* is busy and not accepting new connections.
63	Function/event with no return value used in expression.
64	Object array expected on left side of assignment.
65	Dynamic function not found. Possible causes include: pass by value/reference mismatch.
66	Invalid subscript for array index operation.
67	Null object reference cannot be assigned or passed to an autoinstantiate.
68	Null object reference cannot be passed to external DLL function *name*.
69	Function *name* cannot be called from a secured runtime session.
70	External DLL function *name* cannot be called from a secured runtime session.
71	General protection fault occurred.
72	*name* failed with an operating system error code of *number*.
73	Reference parameters cannot be passed to an asynchronous shared/remote object method.
74	Reference parameters cannot be passed to a shared object method.
75	The server has forced the client to disconnect.
76	Passing null as a parameter to external function *name*.
77	Object passed to shared/remote object method is not a nonvisual user object.
78	Listening works only in the Enterprise version of PowerBuilder.
79	The argument to *name* must be an array.

Number	Meaning
80	The server has timed out the client connection.
81	Function argument file creator must be a four-character string.
82	Function argument file type must be a four-character string.
83	Attempt to invoke a function or event that is not accessible.
84	Wrong number of arguments passed to function/event call.
85	Error in reference argument passed in function/event call.
86	Ambiguous function/event reference.
87	The connection to the server has been lost.
88	Cannot ask for ClassDefinition Information on open painter: *name*.
89	5.0 style proxy objects are not supported. Copy the new style proxy that was generated at migration time.
90	Cannot assign array of type *name* to variable of type array of *name*.
91	Cannot convert *name* in Any variable to *name*. Possible cause: uninitialized value.
92	Required property *name* is missing.
93	CORBA User Exception: *exceptionname*.
94	CORBA System Exception: *exceptionname*.
95	CORBA Objects cannot be created locally.
96	Exception Thrown has not been handled.
97	Cannot save *name* because of a circular reference problem. Possible causes: 1 This object references another class, which in turn references this object. 2 Some other circular reference is pointing back to this object, causing a deadlock condition. Suggested actions: 1 Temporarily remove the circular reference from the referenced object. 2 Make your required changes to this object to refer to that object. 3 Add back the circular reference you removed in step 1. 4 Perform a full rebuild (recommended).
98	Obsolete object reference.
99	Error calling method of a PBNI object.
100	Error loading library containing a PBNI object.
101	Error unloading library containing a PBNI object.
102	Error creating a PBNI object.
103	Error destroying a PBNI object.

Number	Meaning
104	Error calling PowerBuilder system function *functionname*.
105	Executing a HALT statement in a server component is strictly forbidden.
106	Function is reserved or not yet implemented.
107	Argument is out of range.
108	Not enough memory to execute the operation.
109	Cannot assign a null value to array variables.

Some errors terminate the application immediately. They do not trigger the SystemError event.

SystemError event scripts

A typical script for the SystemError event includes a CHOOSE CASE control structure to handle specific errors. To stop the application, include a HALT statement in the SystemError script.

Caution
You can continue your application after a SystemError event, but doing so can cause unpredictable and undesirable effects. Where the application will resume depends on what caused the error. Typically, you are better off reporting the problem to the user, then stopping the application with HALT.

❖ **To test the SystemError event script:**

1 Assign values to the properties of the Error object with the PopulateError function.

2 Call the SignalError function to trigger the SystemError event.

The script for the SystemError event executes.

Tracing and Profiling Applications

About this chapter

This chapter describes how to generate trace information that you can use to improve your application's performance.

Contents

About tracing and profiling an application

You use tracing and profiling to debug and tune an application. When you run an application, you can generate an execution trace file. You use the trace file to create a profile of your application.

The profile shows you which functions and events were called by which other functions and events, how often they were called, when garbage collection occurred, when objects were created and destroyed, and how long each activity took to complete. This information helps you find errors in the application's logic and identify areas that you should rewrite to improve performance.

PBDebug tracing
You can also generate a simple text trace file without timer values by checking Enable PBDebug Tracing in the System Options dialog box.

For more about PBDebug, see "Generating a trace file without timing information" on page 920.

| When you can trace an application | You can create a trace file when you run an application in the PowerBuilder environment, and when you run an executable outside PowerBuilder. For machine-code executable files, the trace file is generated only if you check the Trace Information check box when you build the executable. |

When you run an application with tracing turned on, PowerBuilder records a timer value in a data file every time a specific activity occurs. You control when logging begins and ends and which activities are recorded.

Creating profiles

After you have generated a trace file, you can create several different profiles or views of the application by extracting different types of information from the trace file.

PowerBuilder provides three profiling tools that create profiles (views) of the application for you, but you can also create your own analysis tools.

Using profiling to tune an application

Examining the profiles generated by the profiling tools tells you where the application is spending the most time. You can also find routines that are being called too often, routines being called that you did not expect to call, or routines that are not being called at all. Follow these suggestions for tuning an application:

- The database connection process is often slow. Although you might not be able to speed this up, you might be able to enhance the user's perception of performance by moving the database connection process to a different place in your application.

- Use profiling to tune algorithms. Algorithmic fixes will yield greater performance enhancements than changing single lines of code.

- Optimizing an inefficient function is not as effective as removing unneeded calls to that function.

- Focus on optimizing the routines that are called most often.

- If you cannot speed up a routine, consider adding some user feedback, such as updating MicroHelp or displaying a progress bar.

Collecting trace information

There are three ways to collect trace information. You can use:

*   The Profiling tab on the System Options dialog box

*   A window similar to the Profiling tab

*   Trace objects and functions

Use the Profiling tab if you want to trace an entire application run in the development environment. For more information, see "Tracing an entire application in PowerBuilder" on page 895.

Use a window or trace objects and functions if you want to create a trace file for selected parts of the application or the entire application, either in the development environment or when running an executable file. See "Using a window" on page 895 and "Collecting trace information using PowerScript functions" on page 901.

Collection time
The timer values in the trace file exclude the time taken to collect the trace data. Because an application can be idle (while displaying a MessageBox, for example), percentage metrics are most meaningful when you control tracing programmatically, which can help minimize idle time. Percentages are less meaningful when you create a trace file for a complete application.

Whichever method you use, you can specify:

*   The name and location of the trace file and optional labels for blocks of trace data

*   The kind of timer used in the trace file

*   The activities you want recorded in the trace file

Trace file names and labels

The default name of the trace file is the name of the application with the extension *PBP*. The trace file is saved in the directory where the PBL or executable file resides and overwrites any existing file of the same name. If you run several different tests on the same application, you should change the trace file name for each test.

You can also associate a label with the trace data. If you are tracing several different parts of an application in a single test run, you can associate a different label with the trace data (the trace block) for each part of the application.

Timer kinds

There are three kinds of timer: clock, process, and thread. If your analysis does not require timing information, you can omit timing information from the trace file to improve performance.

If you do not specify a timer kind, the time at which each activity begins and ends is recorded using the clock timer, which measures an absolute time with reference to an external activity, such as the machine's start-up time. The clock timer measures time in microseconds. Depending on the speed of your machine's central processing unit, the clock timer can offer a resolution of less than one microsecond. A timer's resolution is the smallest unit of time the timer can measure.

You can also use process or thread timers, which measure time in microseconds with reference to when the process or thread being executed started. You should always use the thread timer for distributed applications. Both process and thread timers exclude the time taken by any other running processes or threads so that they give you a more accurate measurement of how long the process or thread is taking to execute, but both have a lower resolution than the clock timer.

Trace activities

You can choose to record in the trace file the time at which any of the following activities occurs. If you are using the System Options dialog box or a window, you select the check boxes for the activities you want. If you are using PowerScript functions to collect trace information, you use the TraceActivity enumerated type to identify the activity.

Table 33-1: Trace activities

Trace Activities check box	What is recorded	TraceActivity value
Routine Entry/Exit	Routine entry or exit	ActRoutine!
Routine Line Hits	Execution of any line in any routine	ActLine!
Embedded SQL	Use of an embedded SQL verb	ActESQL!
Object Creation/ Destruction	Object creation or destruction	ActObjectCreate!, ActObjectDestroy!
User Defined Activities	A user-defined activity that records an informational message	ActUser!
System Errors	A system error or warning	ActError!
Garbage Collection	Garbage collection	ActGarbageCollect!
Not available	Routine entry and exit, embedded SQL verbs, object creation and destruction, and garbage collection	ActProfile!
Not available	All except ActLine!	ActTrace!

When you begin and end tracing, an activity of type ActBegin! is automatically recorded in the trace file. User-defined activities, which you use to log informational messages to the trace file, are the only trace activities enabled by default.

Tracing an entire application in PowerBuilder

Use the Profiling tab on the System Options dialog box if you want to collect trace data for an entire application run in the PowerBuilder development environment.

❖ **To trace an entire application in PowerBuilder:**

1   Select Tools>System Options from the PowerBar and select the Profiling tab.

2   Specify a name for the trace file, select the trace options you want, and click OK.

    When you run the application, the activities you selected are logged in the trace file.

Using a window

You can create a window that is similar to the Profiling tab on the System Options dialog box and add it to any application that is under development, so that you can start and stop tracing when testing specific actions.

The *w_start_trace* window is available in the PowerBuilder Profiler sample in the General section of the PowerBuilder Samples and Utilities page at http://powerbuilder.codexchange.sybase.com. This sample also shows the code used to create the profiling tools described in "Analyzing trace information using profiling tools" on page 904.

The *w_start_trace* window lets you specify a trace file name, label, and timer kind, as well as which activities you want to trace:
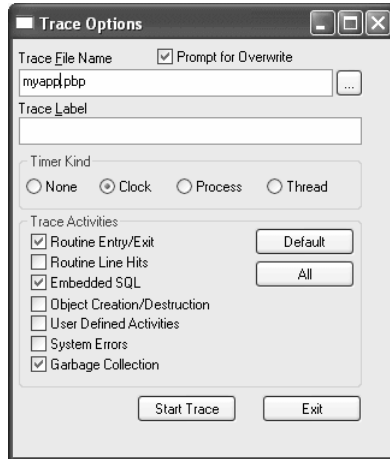
The following instance variables are defined for the window:

```
TimerKind itk_kind
string is_title = 'Trace Options '
string is_starttext
```

The open event for the window sets some defaults:

```
application lapp_current
lapp_current = getapplication()
itk_kind = Clock!
is_starttext = cb_startstop.text
sle_filename.text = classname(lapp_current)+'.pbp'
```

The following code shows the script for the Clicked event of the Start Trace button. The text for the button is set to Start Trace in the painter. When the user clicks Start Trace, the button label changes to Stop Trace. The Clicked event script checks the text on the button before either starting or stopping tracing. This script uses the functions described in "Collecting trace information using PowerScript functions" on page 901:

```
// instance variables:
// errorreturn le_errorreturn
integer li_key

// Check that the button label is Start Trace
// and a trace file name has been entered
if this.text = is_starttext then
```

```
                    if len(trim(sle_filename.text)) = 0 then
                       messagebox(parent.title, &
                         'Trace file name is required',information!)
                       sle_filename.setfocus()
                       return
                    end if

                    // If Prompt for overwrite is checked and the
                    // file exists, pop up a response window
                    if cbx_prompt.checked and &
                       fileexists(sle_filename.text) then
                       li_key = messagebox(parent.title, &
                          'OK to overwrite '+sle_filename.text, &
                          question!,okcancel!,1)
                       if li_key = 2 then return
                    end if

                    // Open the trace file
                    TraceOpen( sle_filename.text, itk_kind )

                    // Enable tracing for checked activities
                    // For each activity, check for errors and close
                    // the trace file if an error occurs
                    if cbx_embeddedsql.checked then
                       le_errorreturn = TraceEnableActivity( ActESql! )
                       if le_errorreturn <> Success! then
                          of_errmsg(le_errorreturn, &
                             'TraceEnableActivity( ActESql! )')
                          le_errorreturn = Traceclose()
                          if le_errorreturn <> Success! then
                             of_errmsg(le_errorreturn,'TraceClose')
                          end if
                          return
                       end if
                    end if

                    if cbx_routineentry.checked then
                       le_errorreturn =TraceEnableActivity(ActRoutine!)
                       if le_errorreturn <> Success! then
                          of_errmsg(le_errorreturn, &
                             'TraceEnableActivity( ActRoutine! )')
                          Traceclose()
                          if le_errorreturn <> Success! then
                             of_errmsg(le_errorreturn,'TraceClose')
                          end if
                          return
```

```
            end if
         end if

         if cbx_userdefined.checked then
            le_errorreturn = TraceEnableActivity( ActUser! )
            if le_errorreturn <> Success! then
               of_errmsg(le_errorreturn, &
                  'TraceEnableActivity( ActUser! )')
               Traceclose()
               if le_errorreturn <> Success! then
                  of_errmsg(le_errorreturn,'TraceClose')
               end if
               return
            end if
         end if

         if cbx_systemerrors.checked then
            le_errorreturn = TraceEnableActivity(ActError! )
            if le_errorreturn <> Success! then
               of_errmsg(le_errorreturn, &
                  'TraceEnableActivity( ActError! )')
               Traceclose()
               if le_errorreturn <> Success! then
                  of_errmsg(le_errorreturn,'TraceClose')
               end if
               return
            end if
         end if

         if cbx_routineline.checked then
            le_errorreturn = TraceEnableActivity( ActLine! )
            if le_errorreturn <> Success! then
               of_errmsg(le_errorreturn, &
                  ' TraceEnableActivity( ActLine! )')
               Traceclose()
               if le_errorreturn <> Success! then
                  of_errmsg(le_errorreturn,'TraceClose')
               end if
               return
            end if
         end if
         if cbx_objectcreate.checked then
            le_errorreturn = &
               TraceEnableActivity( ActObjectCreate! )
            if le_errorreturn <> Success! then
               of_errmsg(le_errorreturn, &
```

```
                  'TraceEnableActivity( ActObject! )')
            Traceclose()
            if le_errorreturn <> Success! then
               of_errmsg(le_errorreturn,'TraceClose')
            end if
            return
      end if
      le_errorreturn = &
         TraceEnableActivity( ActObjectDestroy! )
      if le_errorreturn <> Success! then
         of_errmsg(le_errorreturn, &
            'TraceEnableActivity(ActObjectDestroy!)')
         Traceclose()
         if le_errorreturn <> Success! then
            of_errmsg(le_errorreturn,'TraceClose')
         end if
         return
      end if
end if

if cbx_garbagecoll.checked then
   le_errorreturn = &
      TraceEnableActivity( ActGarbageCollect! )
   if le_errorreturn <> Success! then
      of_errmsg(le_errorreturn, &
       'TraceEnableActivity(ActGarbageCollect! )')
      Traceclose()
      if le_errorreturn <> Success! then
         of_errmsg(le_errorreturn,'TraceClose')
      end if
      return
   end if
end if

// Start tracing
le_errorreturn =TraceBegin( sle_tracelabel.text )
if le_errorreturn <> Success! then
   of_errmsg(le_errorreturn,'TraceBegin')
   return
end if

// Change the title of the window and the
// text of the Start Trace button
parent.title = is_title + '(Tracing)'
this.text = 'Stop &Tracing'
```

```
                        // If the button label is Stop Trace, stop tracing
                        // and close the trace file
                        else
                           le_errorreturn =TraceEnd()
                           if le_errorreturn <> Success! then
                              of_errmsg(le_errorreturn,'TraceEnd')
                              return
                           end if

                           le_errorreturn =TraceClose()
                           if le_errorreturn <> Success! then
                              of_errmsg(le_errorreturn,'TraceClose')
                           end if
                           this.text = is_starttext
                           parent.title = is_title
                        end if
```

of_errmsg function   The window uses two functions to handle error messages. The of_errmsg
                      function displays a message box:

```
                        // of_errmsg
                        Messagebox( this.title,'Error executing '+ as_msg + &
                           '. Error code : '+ of_converterror(ae_error) )
```

of_converterror   The of_converterror function converts the ErrorReturn parameter to a string:
function

```
                        // of_converterror: convert enumerated type
                        // ErrorReturn parameter to text.
                        String ls_result
                        choose case a_error
                           Case Success!
                              ls_result =  "Success!"
                           Case FileCloseError!
                              ls_result =  "FileCloseError!"
                           Case FileOpenError!
                              ls_result =  "FileOpenError!"
                           Case FileReadError!
                              ls_result =  "FileReadError!"
                           Case FileWriteError!
                              ls_result =  "FileWriteError!"
                           Case FileNotOpenError!
                              ls_result =  "FileNotOpenError!"
                           Case FileAlreadyOpenError!
                              ls_result =  "FileAlreadyOpenError!"
                           Case FileInvalidFormatError!
                              ls_result =  "FileInvalidFormatError!"
                           Case FileNotSetError!
                              ls_result =  "FileNotSetError!"
```

```
Case EventNotExistError!
   ls_result =  "EventNotExistError!"
Case EventWrongPrototypeError!
   ls_result =  "EventWrongPrototypeError!"
Case ModelNotExistsError!
   ls_result =  "ModelNotExistsError!"
Case ModelExistsError!
   ls_result =  "ModelExistsError!"
Case TraceStartedError!
   ls_result =  "TraceStartedError!"
Case TraceNotStartedError!
   ls_result =  "TraceNotStartedError!"
Case TraceNoMoreNodes!
   ls_result =  "TraceNoMoreNodes!"
Case TraceGeneralError!
   ls_result =  "TraceGeneralError!"
Case FeatureNotSupportedError!
   ls_result =  "FeatureNotSupportedError!"
Case else
   ls_result =  "Unknown Error Code"
end choose
return ls_result
```

Collecting trace information using PowerScript functions

You use the PowerScript system functions listed in Table 33-2 to collect
information in a trace file. Each of these functions returns a value of type
ErrorReturn, an enumerated datatype.

Table 33-2: PowerScript trace functions

Use this PowerScript function	To do this
TraceOpen	Open a named trace file and set the timer kind.
TraceEnableActivity	Enable logging of the specified activity.
TraceBegin	Start logging all enabled activities. You can pass an optional label for the trace block.
TraceError	Log a severity level and error message to the trace file.
TraceUser	Log a reference number and informational message to the trace file.
TraceEnd	Stop logging all enabled activities.
TraceDisableActivity	Disable logging of the specified activity.
TraceClose	Close the open trace file.

In general, you call the functions in the order shown in the table. That is, you must call TraceOpen before you call any other trace functions. You call TraceClose when you have finished tracing.

TraceEnableActivity and TraceDisableActivity can be called only when a trace file is open but tracing has not begun or has stopped—that is, before you call TraceBegin or after you call TraceEnd.

TraceUser and TraceError can be called only when the trace file is open and tracing is active—that is, after you call TraceBegin and before you call TraceEnd.

About TraceUser and
TraceError

You can use TraceUser to record specific events in the trace file, such as the beginning and end of a body of code. You can also record the execution of a statement you never expected to reach, such as the DEFAULT statement in a CHOOSE CASE block. TraceError works just like TraceUser, but you can use it to signal more severe problems.

Both TraceUser and TraceError take a number and text string as arguments. You can use a simple text string that states what activity occurred, or you can build a string that provides more diagnostic information by including some context, such as the current values of variables. Run the application with only ActUser! or ActError! tracing turned on and then use the Profiling Trace View to pinpoint problems quickly.

Example: trace data
collection

In this example, the user selects a timer kind from a drop-down list and enters a name for the trace file in a single-line edit box. Typically you would use the ErrorReturn return value from every trace call to return an error message if the call fails. For brevity, the example shows this only for the TraceOpen call.

Several trace activities are disabled for a second trace block. The activities that are *not* specifically disabled remain enabled until TraceClose is called.

```
ErrorReturn le_err
integer li_key
TimerKind ltk_kind

CHOOSE CASE ddlb_timerkind.Text
   CASE "None"
      ltk_kind = TimerNone!
   CASE "Clock"
      ltk_kind = Clock!
   CASE "Process"
      ltk_kind = Process!
   CASE "Thread"
      ltk_kind = Thread!
END CHOOSE
```

```
// Open the trace file and return an error message
// if the open fails
le_err = TraceOpen( sle_fileName.Text, ltk_kind )
IF le_err <> Success! THEN &
   of_errmsg(le_err, 'TraceOpen failed')
   RETURN
END IF

// Enable trace activities. Enabling ActLine!
// enables ActRoutine! implicitly
TraceEnableActivity(ActESQL!)
TraceEnableActivity(ActUser!)
TraceEnableActivity(ActError!)
TraceEnableActivity(ActLine!)
TraceEnableActivity(ActObjectCreate!)
TraceEnableActivity(ActObjectDestroy!)
TraceEnableActivity(ActGarbageCollect!)

TraceBegin("Trace_block_1")
// first block of code to be traced
// this block has the label Trace_block_1
…

TraceEnd()

// disable trace activities not needed for
// second block
TraceDisableActivity(ActLine! )
TraceDisableActivity(ActObjectCreate!)
TraceDisableActivity(ActObjectDestroy!)
TraceDisableActivity(ActGarbageCollect!)

TraceBegin("Trace_block_2")
// second block of code to be traced
…

TraceEnd()
TraceClose()
```

Analyzing trace information using profiling tools

After you have created a trace file, the easiest way to analyze it is to use the profiling tools provided on the Tool tab of the New dialog box. There are three tools:

- The Profiling Class View shows information about the objects that were used in the application

- The Profiling Routine View shows information about all the routines (functions and events) that were used in the application

- The Profiling Trace View shows the elapsed time taken by each activity in chronological order

Using the profiling tools as a model

Even if you want to develop your own analysis tools, using the profiling tools is a good way to learn about these models and about profiling in PowerBuilder. When you are ready to develop your own tools, see "Analyzing trace information programmatically" on page 910 for an overview of the approaches you can take.

Profiling Class View

The Class view uses a TreeView control to display statistics for PowerBuilder objects, their functions, and their events. It displays statistics only for those objects that were active while tracing was enabled. The Class view has three tabs:

- **Numbers**   Shows statistics only

- **Graph**   Shows statistics in a bar graph

- **Source**   Shows statistics and source code for those routines that originated in PowerScript source

For each object, the Class view shows all the routines called from each class with the number of times each routine was called (hit) as well as timing information for each call. The following illustration shows part of a Class view. Embedded SQL commands are shown as being called from a pseudo class called ESQL.
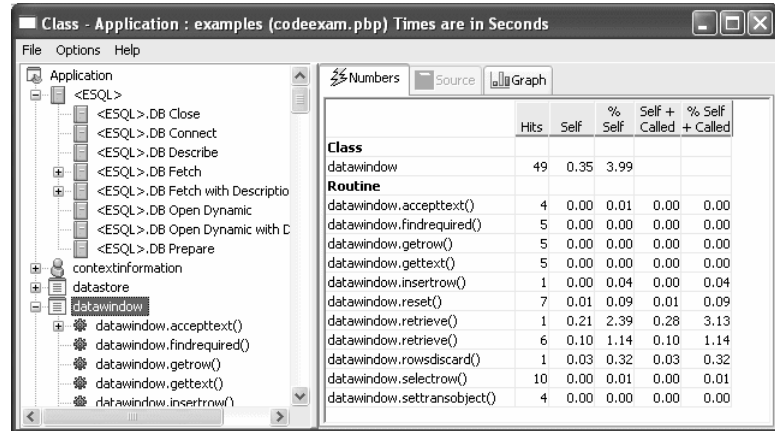
The Class view includes both PowerBuilder system-level objects (such as DataWindow and SystemFunction) and user-defined classes (such as windows and user objects). Each top-level node is a PowerBuilder class. As you expand the TreeView control, each node represents a routine and each subnode represents a called routine.

The Class view uses the call graph model to show cumulative statistics for objects, routines, and called routines. The information displayed on the right side of the display differs depending on the current node on the left.

Table 33-3: Statistics displayed in the Profiling Class View by node

Current node	Statistics displayed
Application	Statistics for each object
Object	Cumulative statistics for the object and detailed statistics for the object's routines
Routine	Cumulative statistics for the routine and detailed statistics for called routines

You can sort items on the right by clicking the heading.

Class view metrics   The Class view displays five metrics. The profiling tool accesses these metrics from instances of the ProfileCall and ProfileRoutine objects. The time scale you specified in the Preferences dialog box determines how times are displayed.

Table 33-4: Metrics in the Profiling Class View

Metric	What it means
Hits	The number of times a routine executed in a particular context.
Self	The time spent in the routine or line itself. If the routine or line was executed more than once, this is the total time spent in the routine or line itself; it does not include time spent in routines called by this routine.
%Self	Self as a percentage of the total time the calling routine was active.
Self+Called	The time spent in the routine or line and in routines or lines called from the routine or line. If the routine or line was executed more than once, this is the total time spent in the routine or line and in called routines or lines.
%Self+Called	Self+Called as a percentage of the total time that tracing was enabled.

About percentages
The percentages captured in the trace file are based on the total time tracing was enabled. Because an application can be idle (while displaying a MessageBox, for example), percentage metrics are most meaningful when you control tracing programmatically, which can help to minimize idle time. Percentages are least meaningful when you create a trace file for a complete application.

Profiling Routine View

The Routine view displays statistics for a routine, its calling routines, and its called routines. It uses multiple DataWindow objects to display information for a routine:

- **Called routines**   The top DataWindow lists functions and events called by the current routine.

- **Current routine**   The middle DataWindow and the DataWindow on the right highlight the current routine and show detailed statistics.

- **Calling routines**   The bottom DataWindow lists functions and events that call the routine displayed in the middle DataWindow.

The Routine view has two tabs:

- **Detail**   Shows statistics only

- **Source**   Shows statistics and source code for those routines that originated in PowerScript source

The Routine view uses the call graph model to show the call chain and cumulative statistics for routines and called routines.
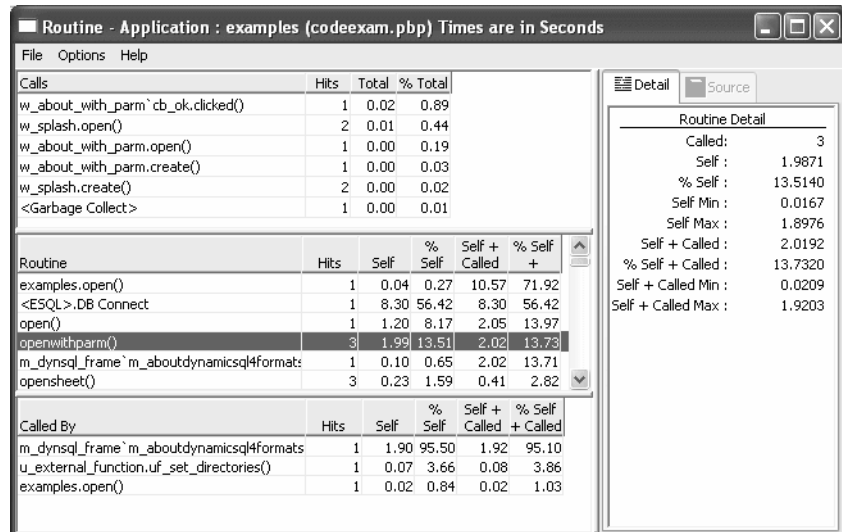
You can specify the current routine by clicking in the various DataWindows.

Table 33-5: Specifying the current routine in the Profiling Routine View

To do this	Click here
Establish a new current routine in the current routine DataWindow	On the routine. The profiling tool updates the top and bottom DataWindows with information on called and calling routines.
Select a calling routine as the new routine	On the routine in the top DataWindow. The profiling tool makes it the current routine in the middle DataWindow.
Select a called routine as the new routine	On the routine in the bottom DataWindow. The profiling tool makes it the current routine in the middle DataWindow.

You can sort items by clicking the column headings.

Routine view metrics

The Routine view displays nine metrics. The profiling tool accesses these metrics from instances of the ProfileCall and ProfileRoutine objects. The time scale you specified in the Preferences dialog box determines how times are displayed.

Table 33-6: Metrics in the Profiling Routine View

Metric	What it means
Hits (Called on Detail tab)	The number of times a routine executed in a particular context.
Self	The time spent in the routine or line itself. If the routine or line was executed more than once, this is the total time spent in the routine or line itself; it does not include time spent in routines called by this routine.
%Self	Self as a percentage of the total time the calling routine was active.
Self Min	The shortest time spent in the routine or line itself. If the routine or line was executed only once, this is the same as AbsoluteSelfTime.
Self Max	The longest time spent in the routine or line itself. If the routine or line was executed only once, this is the same as AbsoluteSelfTime.
Self+Called	The time spent in the routine or line and in routines or lines called from the routine or line. If the routine or line was executed more than once, this is the total time spent in the routine or line and in called routines or lines.
%Self+Called	Self+Called as a percentage of the total time that tracing was enabled.
Self+Called Min	The shortest time spent in the routine or line and in called routines or lines. If the routine or line was executed only once, this is the same as AbsoluteTotalTime.
Self+Called Max	The longest time spent in the routine or line and in called routines or lines. If the routine or line was executed only once, this is the same as AbsoluteTotalTime.

Profiling Trace View

The Trace view uses a TreeView control to display the events and functions in the trace file. The initial display shows top-level routines. Each node expands to show the sequence of routine execution. The fully expanded TreeView shows the complete sequence of executed instructions for the trace file.

The Trace view uses the trace tree model to show the sequence of execution. It includes statistics and (for those routines that originated in PowerScript source) source code.

You can use the Trace View Options section of the Preferences dialog box to control the display:

- **System routines**   This option controls whether the Trace view includes information for lines that execute PowerBuilder system routines.

- **Line information**   This option controls whether the Trace view includes line numbers.

The following screen shows a Trace view with several nodes expanded. The number to the right of each item is the execution time for that item.
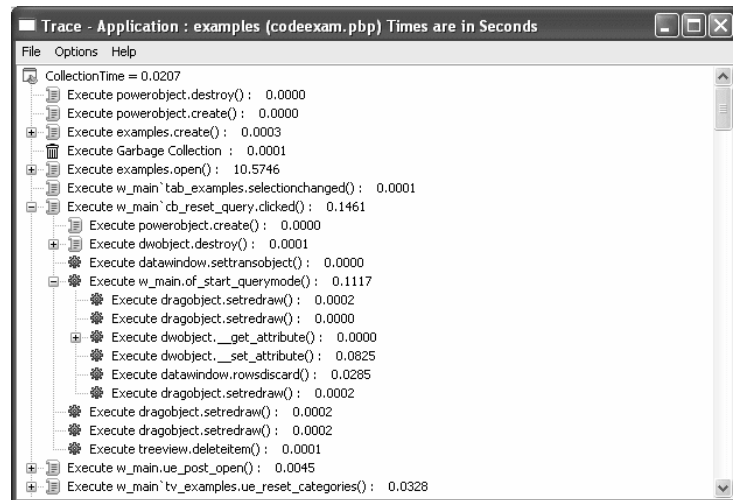
Trace view metrics   The Trace view displays two metrics. The profiling tool accesses these metrics from instances of the TraceTree and TraceTreeNode objects.

Table 33-7: Metrics in the Profiling Trace View

Entry	What it means
Routine or line number	The routine or line number that was executed.
Execution time	Total execution time for the Tree view entry. This is total time from the start of the entry to the end of the entry. For example, if you call the MessageBox function, this value reflects the elapsed time from when the message box was opened until the user provided some kind of response.

About preferences
The specifications you make in the Preferences dialog box control whether the Trace view displays system functions and line numbers.

Setting call aggregation preferences

You can control how the profiling tools display information using the Preferences dialog box. To open it, select Options>Preferences from any profiling view's menu bar.

In both Class and Routine views, you can choose how functions and events that are called more than once are displayed. Select the Aggregate Calls check box if you want the view to display a single line for each called function or event that represents a sum of all the times it was called. If you do not select the check box, the view displays statistics for each call on a separate line.

For example, if aggregation is enabled and a function calls another function five times, you see one entry with five hits; with no aggregation, you see five separate entries for the same function.

Internally, the profiling tool controls aggregation by using the AggregateDuplicateRoutineCalls boolean argument to the OutgoingCallList and IncomingCallList functions on the ProfileRoutine object.

Analyzing trace information programmatically

PowerBuilder provides three ways to analyze trace information using built-in system objects and functions:

• Analyze performance by building a call graph model

A call graph model contains information about all the routines in the trace file: how many times each routine was called, which routines called it and which routines it called, and the execution time taken by the routine itself and any routines it called.

• Analyze program structure and logical flow by building a trace tree model

A trace tree model contains information about all recorded activities in the trace file in chronological order, with the elapsed time for each activity.

• Access the data in the trace file directly

Trace objects and functions let you build your own model and analysis tools by giving you access to all the data in the trace file.

The profiling tools use the first two ways. The Class and Routine views are based on a call graph model, and the Trace view is based on a trace tree model.

Supporting files needed

To create a profile from a trace file, PowerBuilder must also access the PBL, PBD, or executable file used to create the trace file, and the PBL, PBD, or executable file must be in the same location as when the trace file was created.

Analyzing performance with a call graph model

You use the PowerScript functions and PowerBuilder objects listed in Table 33-8 to analyze the performance of an application.

Table 33-8: Functions for analyzing performance

Use this function	With this object	To do this
SetTraceFileName	Profiling	Set the name of the trace file to be analyzed.
BuildModel	Profiling	Build a call graph model based on the trace file. You can pass optional parameters that let you track the progress of the build.
RoutineList	Profiling and ProfileClass	Get a list of routines in the model or in a class.
ClassList	Profiling	Get a list of classes in the model.
SystemRoutine	Profiling	Get the name of the routine node that represents the root of the model.
IncomingCallList	ProfileRoutine	Get a list of routines that called a specific routine.
OutgoingCallList	ProfileRoutine and ProfileLine	Get a list of routines called by a specific routine or from a specific line.
LineList	ProfileRoutine	Get a list of lines in the routine in line order.
DestroyModel	Profiling	Destroy the current performance analysis model and all the objects associated with it.

Each of these functions returns a value of the enumerated datatype ErrorReturn. The objects contain information such as the number of times a line or routine was executed, and the amount of time spent in a line or routine and in any routines called from that line or routine.

Using the BuildModel function to build a call graph model

The call graph model that you create with the BuildModel function contains all the routines in the trace file and can take a long time to build. If you want to monitor the progress of the build or you want to be able to interrupt it while the model is being built, you can pass optional arguments to BuildModel.

BuildModel arguments

BuildModel takes three arguments: the name of an object of type PowerObject, the name of a user event, and a long value representing how often the user event should be triggered as a percentage of the build completed.

The user event returns a boolean value and has two arguments: the number of the current activity, and the total number of activities in the trace file.

Destroying existing models

Before you call BuildModel, you can call DestroyModel to clean up any objects remaining from an existing model.

Example: building a call graph model

In the following example, the user event argument to BuildModel is called ue_progress and is triggered each time five percent of the activities have been processed. The progress of the build is shown in a window called w_progress that has a cancel button.

```
Profiling lpro_model
lpro_model = CREATE Profiling
ib_cancel = FALSE
lpro_model.SetTraceFileName(is_fileName )

open(w_progress)
// call the of_init window function to initialize
// the w_progress window
w_progress.of_init(lpro_model.numberofactivities, &
   'Building Model', this, 'ue_cancel')

// Build the call graph model
lpro_model.BuildModel(this, 'ue_progress', 5)

// clicking the cancel button in w_progress
// sets ib_cancel to TRUE and
// returns FALSE to ue_progress
IF ib_cancel THEN &
   close(w_progress)
   RETURN -1
END IF
```

Extracting information from the call graph model

After you have built a call graph model of the application, you can extract detailed information from it.

For routines and lines, you can extract the timing information shown in Table 33-9 from the ProfileRoutine and ProfileLine objects.

Table 33-9: Timing information in the call graph model

Property	What it means
AbsoluteSelfTime	The time spent in the routine or line itself. If the routine or line was executed more than once, this is the total time spent in the routine or line itself.
MinSelfTime	The shortest time spent in the routine or line itself. If the routine or line was executed only once, this is the same as AbsoluteSelfTime.
MaxSelfTime	The longest time spent in the routine or line itself. If the routine or line was executed only once, this is the same as AbsoluteSelfTime.
AbsoluteTotalTime	The time spent in the routine or line and in routines or lines called from the routine or line. If the routine or line was executed more than once, this is the total time spent in the routine or line and in called routines or lines.
MinTotalTime	The shortest time spent in the routine or line and in called routines or lines. If the routine or line was executed only once, this is the same as AbsoluteTotalTime.
MaxTotalTime	The longest time spent in the routine or line and in called routines or lines. If the routine or line was executed only once, this is the same as AbsoluteTotalTime.
PercentSelfTime	AbsoluteSelfTime as a percentage of the total time tracing was active.
PercentTotalTime	AbsoluteTotalTime as a percentage of the total time tracing was active.

Example: extracting information from a call graph model

The following function extracts information from a call graph model about the routines called from a specific routine. You would use similar functions to extract information about the routines that called the given routine and about the routine itself.

The function takes a ProfileCall object and an index as arguments and returns a structure containing the number of times the called routine was executed and execution times for the called routine.

```
str_func_detail lstr_result
ProfileClass lproclass_class
```

```
ProfileRoutine lprort_routine

// get the name of the called routine
// from the calledroutine property of
// the ProfileCall object passed to the function
lprort_routine = a_pcall.Calledroutine
lstr_result.Name = ""
lproclass_class = a_pcall.Class
IF isValid(lproclass_class) THEN &
    lstr_result.Name += lproclass_class.Name + "."
lstr_result.name += a_pcall.Name

lstr_result.hits = a_pcall.HitCount
lstr_result.selfTime = a_pcall. &
    AbsoluteSelfTime * timeScale
lstr_result.totalTime = a_pcall. &
    AbsoluteTotalTime * timeScale
lstr_result.percentSelf = a_pcall.PercentSelfTime
lstr_result.percentTotal= a_pcall.PercentTotalTime
lstr_result.index = al_index

RETURN lstr_result
```

Analyzing structure and flow using a trace tree model

You use the PowerScript functions and PowerBuilder objects listed in Table 33-10 to build a nested trace tree model of an application.

Table 33-10: Functions for analyzing program structure and flow

Use this function	With this object	To do this
SetTraceFileName	TraceTree	Set the name of the trace file to be analyzed.
BuildModel	TraceTree	Build a trace tree model based on the trace file. You can pass optional parameters that let you track the progress of the build.
EntryList	TraceTree	Get a list of the top-level entries in the trace tree model.

Use this function	With this object	To do this
GetChildrenList	TraceTreeRoutine, TraceTreeObject, and TraceTreeGarbageCollect	Get a list of the children of the routine or object—that is, all the routines called directly by the routine, or the destructor called as a result of the object's deletion.
DestroyModel	TraceTree	Destroy the current trace tree model and all the objects associated with it.

Each of these functions returns a value of type ErrorReturn.

Each TraceTreeNode object returned by the EntryList and GetChildrenList functions represents a single node in the trace tree model and contains information about the parent of the node and the type of activity it represents.

Inherited objects

The following objects inherit from TraceTreeNode and contain additional information, including timer values:

> TraceTreeError
> TraceTreeESQL
> TraceTreeGarbageCollect
> TraceTreeLine
> TraceTreeObject
> TraceTreeRoutine
> TraceTreeUser

Using *BuildModel* to build a trace tree model

You use the same approach to building a trace tree model as you do to building a call graph model, except that you build a model of type TraceTree instead of type Profiling.

For example:

```
TraceTree ltct_treemodel
ltct_treemodel = CREATE TraceTree
ltct_treeModel.SetTraceFileName(is_fileName )

ltct_treeModel.BuildModel(this, 'ue_progress', 1)
```

For more about using BuildModel, see "Using the BuildModel function to build a call graph model" on page 912.

Extracting information from the trace tree model

To extract information from a tree model, you can use the EntryList function to create a list of top-level entries in the model and then loop through the list, extracting information about each node. For each node, determine its activity type using the TraceActivity enumerated datatype, and then use the appropriate TraceTree object to extract information.

Example: trace tree model

The following simple example extracts information from an existing trace tree model and stores it in a structure:

```
TraceTreeNode ltctn_list[], ltctn_node
long ll_index, ll_limit
string ls_line
str_node lstr_node

ltct_treemodel.EntryList(ltctn_list)
ll_limit = UpperBound(ltctn_list)
FOR ll_index = 1 to ll_limit
    ltctn_node = ltctn_list[ll_index]
    of_dumpnode(ltctn_node, lstr_node)
    // insert code to handle display of
    // the information in the structure here
    …
NEXT
```

The of_dumpnode function takes a TraceTreeNode object and a structure as arguments and populates the structure with information about each node. The following code shows part of the function:

```
string ls_exit, ls_label, ls_routinename
long ll_node_cnt
TraceTreeNode ltctn_list[]
errorreturn l_err

astr_node.Children = FALSE
astr_node.Label = ''
IF NOT isvalid(atctn_node) THEN RETURN
CHOOSE CASE atctn_node.ActivityType
    CASE ActRoutine!
      TraceTreeRoutine ltctrt_routin
      ltctrt_routine = atctn_node
      IF ltctrt_routine.Classname = '' THEN &
          ls_routinename = ltctrt_routine.ClassName + "."
      END IF
      ls_routinename += ltctrt_routine.Name
      ltctrt_routine.GetChildrenList(ltctn_list)
      ll_node_cnt = UpperBound(ltctn_list)
```

```
              ls_label = "Execute " + ls_routinename + ' :' + &
                  space(ii_offset) + String(l_timescale * &
                  (ltctrt_routine.ExitTimerValue - &
                  ltctrt_routine.EnterTimerValue), '0.000000')
              astr_node.Children = (ll_node_cnt > 0)
              astr_node.Label = ls_label
              astr_node.Time = ltctrt_routine.EnterTimerValue
              RETURN
          CASE ActLine!
              TraceTreeLine tctln_treeLine
              tctln_treeLine = atctn_node
              ls_label = LINEPREFIX + &
                  String(tctln_treeLine.LineNumber )
              astr_node.time = tctln_treeLine.Timervalue
              ...
              // CASE statements omitted
              ...
          CASE ELSE
              ls_label = "INVALID NODE"
          END CHOOSE

          astr_node.label = ls_label
          RETURN
```

Accessing trace data directly

You use the PowerScript functions and PowerBuilder objects listed in Table 33-11 to access the data in the trace file directly so that you can develop your own analysis tools.

Table 33-11: Functions for direct access to trace data

Use this function	With this object	To do this
Open	TraceFile	Opens the trace file to be analyzed.
NextActivity	TraceFile	Returns the next activity in the trace file. The value returned is of type TraceActivityNode.
Reset	TraceFile	Resets the next activity to the beginning of the trace file.
Close	TraceFile	Closes the open trace file.

With the exception of NextActivity, each of these functions returns a value of type ErrorReturn. Each TraceActivityNode object includes information about the category of the activity, the timer value when the activity occurred, and the activity type.

Timer values
The category of the activity is either TraceIn! or TraceOut! for activities that have separate beginning and ending points, such as routines, garbage collection, and tracing itself. Each such activity has two timer values associated with it: the time when it began and the time when it completed.

Activities that have only one associated timer value are in the category TraceAtomic!. ActLine!, ActUser!, and ActError! are all atomic activities.

Inherited objects
The following objects inherit from TraceActivityNode and contain data about the associated activity type:

TraceBeginEnd
TraceError
TraceESQL
TraceGarbageCollect
TraceLine
TraceObject
TraceRoutine
TraceUser

TraceTreeNode and TraceActivityNode objects
The objects that inherit from TraceActivityNode are analogous to those that inherit from TraceTreeNode, and you can use similar techniques when you write applications that use them.

For a list of activity types, see "Trace activities" on page 894.

Using the TraceFile object

To access the data in the trace file directly, you create a TraceFile object, open a trace file, and then use the NextActivity function to access each activity in the trace file sequentially. For each node, determine what activity type it is by examining the TraceActivity enumerated datatype, and then use the appropriate trace object to extract information.

Example: direct access to trace data

The following example creates a TraceFile object, opens a trace file called ltcf_file, and then uses a function called of_dumpActivityNode to report the appropriate information for each activity depending on its activity type.

```
string ls_fileName
TraceFile ltcf_file
TraceActivityNode ltcan_node
string ls_line

ls_fileName = sle_filename.Text
ltcf_file = CREATE TraceFile
ltcf_file.Open(ls_fileName)
ls_line = "CollectionTime = " + &
   String(Truncate(ltcf_file.CollectionTime, 6)) &
   + "~r~n" + "Number of Activities = " + &
   String(ltcf_file.NumberOfActivities) + "~r~n" + &
   "Time Stamp " + "Activity" + "~r~n"

mle_output.text = ls_line

ltcan_node = ltcf_file.NextActivity()
DO WHILE IsValid(ltcan_node)
   ls_line += of_dumpActivityNode(ltcan_node)
   ltcan_node = ltcf_file.NextActivity()
LOOP

mle_output.text = ls_line
ltcf_file.Close()
```

The following code shows part of of_dumpActivityNode:

```
string lstr_result

lstr_result = String(Truncate(atcan_node. &
   TimerValue, 6)) + " "
CHOOSE CASE atcan_node.ActivityType
   CASE ActRoutine!
      TraceRoutine ltcrt_routine
      ltcrt_routine = atcan_node
      IF ltcrt_routine.IsEvent THEN
        lstr_result += "Event: "
      ELSE
        lstr_result += "Function: "
      END IF
      lstr_result += ltcrt_routine.ClassName + "." + &
         ltcrt_routine.name + "(" + &
         ltcrt_routine.LibraryName + ") " &
```

```
                     + String(ltcrt_routine.ObjectId) + "~r~n"
         CASE ActLine!
            TraceLine ltcln_line
            ltcln_line = atcan_node
            lstr_result += "Line: " +      &
                String(ltcln_line.LineNumber) + "~r~n"
         CASE ActESQL!
            TraceESQL ltcsql_esql
            ltcsql_esql = atcan_node
            lstr_result += "ESQL: " + ltcsql_esql.Name &
                + "~r~n"

         // CASE statements and code omitted
         ...
         CASE ActBegin!
            IF atcan_node.Category = TraceIn! THEN
                lstr_result += "Begin Tracing~r~n"
            ELSE
                lstr_result += "End Tracing~r~n"
            END IF
         CASE ActGarbageCollect!
            lstr_result += "Garbage Collection~r~n"
         CASE else
            lstr_result += "Unknown Activity~r~n"
      END CHOOSE

      RETURN lstr_result
```

Generating a trace file without timing information

If you want to generate an activity log with no timing information in a text file, you can turn on PBDebug tracing in the System Options dialog box. The PBDebug trace file contains a log showing which object functions and instructions and system DLL functions were executed in chronological order.

❖ **To generate a simple trace file:**

1   Select Tools>System Options and check the Enable PBDebug Tracing check box.

2   Check the Prompt Before OverWriting PBDebug Output File box if you want to retain existing trace output when you run or debug the application.

3   (Optional) Specify a pathname for the PBDebug output file.

4   Run your application.

If you do not check the Prompt Before OverWriting PBDebug Output File box, PowerBuilder overwrites the existing trace file every time you run the application or click the Start button in the debugger. If you check the box, it displays a response window. You can choose to overwrite the file, append new trace output to the existing file, or cancel the run or debug session.

If you want to retain the trace file, save it with a different name before running the application, or specify a new file name in the System Options dialog box.

If you do not specify an output file path, PowerBuilder creates an output file in the same directory as the PowerBuilder executable file. The output file has the same name as the PowerBuilder executable with the extension *DBG*. If you do not have write permission to this directory, you must specify a value for the output file path.

Turning PBDebug off
Running your application with PBDebug on will slow down execution. Be sure to clear the Enable PBDebug Tracing check box on the System Options dialog box if you do not need this trace information.

For information on creating the same kind of diagnostic trace file when you run your compiled application outside PowerBuilder, see "Tracing execution" on page 933.

Creating Executables and Components

About this chapter

This chapter describes how to create an executable version of your target. It also provides an overview of how you use the PowerBuilder Project painter to build other kinds of components.

Contents

About building PowerBuilder targets

You can build many types of targets with PowerBuilder. For traditional client/server applications, you need to create an executable version of your target that you can deploy to users' machines. If you are building a distributed application with PowerBuilder, you typically build a client executable file and a server component that you can deploy to a transaction server. For some types of distributed applications, you need to build proxy objects.

Using the Project painter

You use the Project painter to create and maintain PowerBuilder projects that build all these different objects. The Project painter allows you to streamline the generation of the files your target needs and to rebuild easily when you make changes. There is a wizard to help you set up each project type.

Table 34-1 lists the types of projects you can create and what you can build using the project.

Table 34-1: PowerScript target project types

Project	What it builds
Application	An executable file and optional dynamic libraries.
Automation Server	An automation server (formerly called an OLE automation server) based on a custom class user object that can be used with COM-enabled clients.
COM/COM+ Component	A COM server based on one or more custom class user objects that can be deployed to COM+ and/or used with COM-enabled clients.
EAServer Component	One or more EAServer components based on custom class user objects that are deployed to EAServer and can be used with any EAServer client or component.
EAServer Proxy	One or more proxy objects (stubs) that can be used by a PowerBuilder client to access functions in EAServer components. The EAServer components can be built with PowerBuilder or another development tool.
EJB Client Proxy	One or more proxy objects (stubs) that can be used by a PowerBuilder client to access functions in an EJB component on an application server.
Web DW Container	An EAServer component that contains your DataWindow definitions and is deployed to EAServer. It uses the interface of the generic Web DataWindow server component.
Web Service Proxy	A proxy object that can be used by a PowerBuilder client to invoke a Web service defined in a WSDL (Web Services Description Language) file.

For how to create a new project, see "Creating a project" on page 925.

Building executable files

If you are building an executable file, there are two basic ways to package the application:

• As one standalone executable file that contains all the objects in the application

• As an executable file and one or more dynamic libraries that contain objects that are linked at runtime

Read the chapter on packaging your application for deployment in *Application Techniques* to get an understanding of the best way for you to package the application. Then follow the procedures in "Defining an executable application project" on page 926 to implement your strategy.

| Building other types of targets | For an overview of how you use the Project painter to build different types of components, see "Building components and proxies" on page 939. |

| Providing other resources | You might need to provide additional resources that your target uses, such as bitmaps and icons. There are two ways to provide resources: |

- Distribute them separately

- Include them in a PowerBuilder resource file (PBR) and build an executable, a dynamic library, or a component using the resource file

For more information, see "Distributing resources" on page 931.

| Building the workspace | You can build and deploy all the targets in your workspace using buttons on the PowerBar, pop-up menus in the System Tree, or a command line. For more information, see "Building workspaces" on page 26. |

Creating a project

You can create a new project when you create a new template application or a new EAServer, COM/COM+, or automation server component using Start or Object wizards. You can also create a project at any time from the Project page in the New dialog box.

The Project page has two kinds of icons: icons that open wizards that help you set up a project, and icons that open the Project painter. Wizard icons display next to the icon for the same project type. The following procedure describes how to create a new project from the Project page.

❖ **To create a new project object from the Project page:**

1 Select File>New or click the New button in the PowerBar to open the New dialog box.

2 Select the Project tab.

3 Select the wizard or project type you need and click OK.

If you select a wizard, complete the wizard screens to create a new project with most of its properties specified. Use the context-sensitive Help if you are not sure what to enter. You can open the Project painter now or later to modify the properties if necessary and to build the project.

If you do not select a wizard, the Project painter for the type of object you selected opens so that you can specify properties of the project object.

Once you have created a project, you can open it from the System Tree.

Projects can be modified only in the painter
Unlike most other PowerBuilder objects, a project object cannot be edited
in the Source editor.

Defining an executable application project

The Project painter for executable applications allows you to streamline the
generation of executable files and dynamic libraries. When you build a project
object, you specify the following components of your application:

- Executable file name

- Which of the libraries you want to distribute as dynamic libraries

- Which PowerBuilder resource files (if any) should be used to build the
  executable file and the dynamic libraries

- Which build options you want to use in your project

- Which code generation options you want to use

- Version information for your application

If you do not use the Template Application Target wizard to create a new
application project, you need to define the project using a Project wizard or by
setting project properties in the Project painter. After you have created a
project, you might need to update it later because your library list has changed
or you want to change your compilation options.

❖ **To define or modify an executable application project:**

1   Select the Application project icon on the Project tab in the New dialog
    box to create a new application project, or select File>Open to open an
    existing application project.

The Project painter workspace displays.
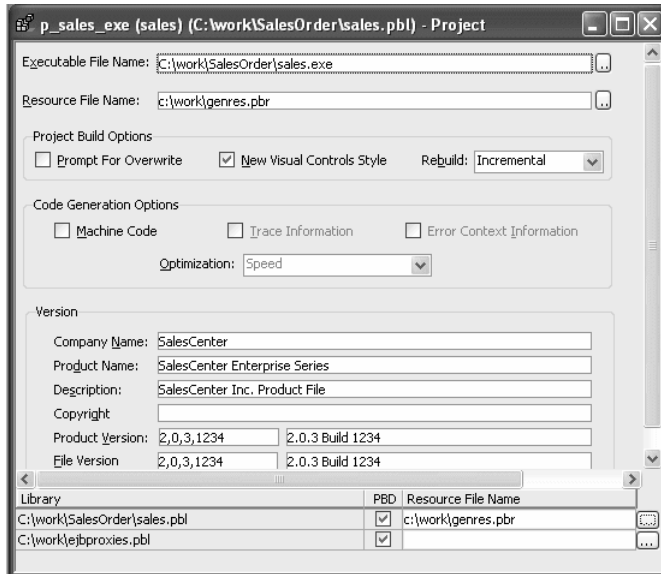
2   Specify or modify options as needed.

If you opened an existing project or a project created using the wizard, the options already selected display in the workspace. For information about each option, see "Executable application project options" next.

3   When you have finished defining the project object, save the object by selecting File>Save from the menu bar.

PowerBuilder saves the project as an independent object in the specified library. Like other objects, projects are displayed in the System Tree and the Library painter.

Executable application project options

Table 34-2 describes each of the options you can specify in the Project painter for executable applications. You can also specify most of these options in the Application Project wizard.

Table 34-2: Options for executable application projects

Option	What you specify
Executable File Name	Specify a name for the executable. The name must have the extension *EXE*. If you do not want the executable saved to your current directory, click the Browse (...) button next to the box to navigate to a different directory.

Option	What you specify
Resource File Name	(Optional) Specify a PowerBuilder resource file (PBR) for your executable if you dynamically reference resources (such as bitmaps and icons) in your scripts *and* you want the resources included in the executable file instead of having to distribute the resources separately. You can type the name of a resource file in the box or click the button next to the box to browse your directories for the resource file you want to include. For more about PBRs, see "Distributing resources" on page 931.
Prompt for Overwrite	Select this if you want PowerBuilder to prompt you before overwriting files. PowerBuilder overwrites any files it creates when building your application.
New Visual Style Controls	Select this to add a manifest file to the application that specifies the appearance of the controls as an application resource. When a user runs the application on Windows XP with the Windows XP style for controls set in the control panel, all PowerBuilder windows, DataWindow controls that mirror standard Windows controls, and other controls, display with the new style.
Rebuild	Specify either Full or Incremental to indicate whether you want PowerBuilder to regenerate all objects in the application libraries before it creates the executable and dynamic libraries. If you choose Incremental, PowerBuilder regenerates only objects that have changed, and objects that reference any objects that have changed, since the last time you built your application. As a precaution, regenerate all objects before rebuilding your project.
Machine Code	Select this if you want to generate compiled code instead of Pcode. For more information about compiled code and Pcode, see *Application Techniques*. Selecting Machine Code enables the other code generation options in the Project painter. They cannot be set in the wizard.
Trace Information	Select this if you want to create a trace file when you run your compiled code executable. You can use the trace file to troubleshoot or profile your application. For more information on obtaining trace information, see "Tracing execution" on page 933.
Error Context Information	Select this if you want PowerBuilder to display context information (such as object, event, and script line number) for runtime errors.
Optimization	Select an optimization level. You can build your application with no optimizations, or you can optimize for speed or space.

Option	What you specify
Version	Specify your own values for the Company Name, Product Name, Description, Copyright, Product Version, and File Version fields associated with the executable file and with machine-code DLLs. These values become part of the Version resource associated with the executable file, and most of them display on the Version tab page of the Properties dialog box for the file in Windows Explorer.
	The File and Product Version numeric fields, on the left in the painter, are used by Microsoft Installer to determine whether a file needs to be updated when a product is installed. They must have the format:
	N,N,N,N
	The four numbers can be used to represent the major version, minor version, point release, and build number of your product. They must all be present. If your file versioning system does not use all these components, you can replace the unused numbers with zeroes. For example, `3,1,333` causes an error message to display when you attempt to save or build the project, but `3,1,333,0` does not. The maximum value for any of the numbers in the string is 65535.
	The File and Product Version string fields, on the right in the painter, can have any format. These fields display in the executable file's Properties dialog box.
PBD or DLL	The label for this check box depends on whether you are building a Pcode or machine code executable. Select the check box to define a library as a dynamic library to be distributed with your application.
	If you are generating Pcode, you create PBD files. If you are generating machine code, you create DLL files. For more about dynamic libraries, see "Using dynamic libraries" on page 930.
Resource File Name	Specify a resource file for a dynamic library if it uses resources (such as bitmaps and icons) and you want the resources included in the dynamic library instead of having to distribute the resources separately. The file name cannot be specified in the wizard.

Location of temporary files

The machine code generation process puts temporary files in a temporary directory, such as the *TEMP* directory. You can specify a different location in the [PB] section of your PowerBuilder initialization file with the CODEGENTEMP variable. You might want to do this if you have limited space on your local system.

For example:

```
CODEGENTEMP=e:\pbtempdir
```

Using dynamic libraries

You can store the objects used in your PowerBuilder application in more than one library and, when you run the application, dynamically load any objects that are not contained in the application's executable file. This allows you to break the application into smaller units that are easier to manage and makes the executable file smaller. You do this by using dynamic libraries. If you compile using Pcode, PowerBuilder builds PowerBuilder dynamic libraries (PBD files). If you use machine code, PowerBuilder builds Dynamic Link Libraries (DLL files).

When you distribute your application to users, you distribute the executable, the dynamic libraries, and PowerBuilder runtime DLLs. For more information about deployment and a list of PowerBuilder runtime DLLs, see *Application Techniques*.

Dynamic library names

PowerBuilder dynamic libraries are given the name of the PBL with the extension *.pbd*. For example, the Pcode library built from *test.pbl* is named *test.pbd*.

Machine-code dynamic libraries are given the extension *.dll*. For example, the machine-code library built from *test.pbl* is named *test.dll*.

Reducing the size of dynamic libraries

When PowerBuilder builds a dynamic library, it copies the compiled versions of *all objects* from the source library (PBL file) into the dynamic library.

The easiest way to specify source libraries is simply to use your standard PowerBuilder libraries as source libraries. However, using this technique can make your dynamic libraries larger than they need to be, because they include all objects from the source library, not just the ones used in your application. You can create a PowerBuilder library that contains only the objects that you want in a dynamic library.

❖ **To create a source library to be used as a dynamic library:**

1   In the Library painter, place in one standard PowerBuilder library (a PBL file) all the objects that you want in the dynamic library.

     If you need to create a new library, select Entry>Library>Create from the menu bar, then drag or move the objects into the new library.

2   Make sure the application's library search path includes the new library.

Multiple dynamic libraries
You can use as many dynamic libraries as you want in an application. To do so, create a source library (PBL file) for each of them.

Specifying the dynamic libraries in your project	When you define your project, you tell PowerBuilder which of the libraries in the application's library search path will be dynamic by checking the PBD or DLL check box next to the library name in the Project painter.
Including additional resources for a dynamic library	When building a dynamic library, PowerBuilder does not inspect the objects; it simply copies the compiled form of the objects into the dynamic library. Therefore, if any of the objects in the library use resources (pictures, icons, and pointers)—*either specified in a painter or assigned dynamically in a script*—and you do not want to provide these resources separately, you must list the resources in a PowerBuilder resource (PBR) file. Doing so enables PowerBuilder to include the resources in the dynamic library when it builds it.

❖ **To reference additional resources:**

1   List the resources in a PBR file, as described in "Using PowerBuilder resource files" on page 932.

2   Use the Resource File Name box in the Project painter workspace to reference the PBR file in the dynamic library.

Distributing resources

You can choose to distribute your resources (pictures, pointers, and icons) separately or include them in your executable file or dynamic library.

Distributing resources separately

When a resource is referenced at runtime, if the resource has not been included in the executable file or in a dynamic library, PowerBuilder looks for it in the search path. You need to distribute resources with your application and make sure they get installed in the user's search path.

For example, assume you use two bitmap files as in the following script:

```
IF Balance < 0 THEN
   p_logo.PictureName = "frown.bmp"
ELSE
   p_logo.PictureName = "smile.bmp"
END IF
```

You can distribute the files *frown.bmp* and *smile.bmp* with your application. If the files are on the search path at runtime, the application can load them when they are needed.

The Windows search path is as follows:

1   The current directory

2   The Windows directory

3   The Windows system directory

4   All directories in the PATH environment variable

Using PowerBuilder resource files

Instead of distributing resources separately, you can create a PowerBuilder resource file (a PBR file) that lists all dynamically assigned resources.

A PBR file is an ASCII text file in which you list resource names (such as BMP, CUR, GIF, ICO, JPEG, RLE, and WMF files) and DataWindow objects. To create a PBR file, use a text editor. List the name of each resource, one resource on each line, then save the list as a file with the extension PBR.

Here is a sample PBR file:

```
ct_graph.ico
document.ico
codes.ico
button.bmp
next1.bmp
prior1.bmp
```

PowerBuilder compiles the listed resources into the executable file or a dynamic library file, so the resources are available directly at runtime.

Using DataWindow objects
If the objects in one PBL reference DataWindow objects, either statically or dynamically, that are in a different PBL, you must either specify a PowerBuilder resource file that includes the DataWindow objects, or define the library that includes them as a PBD or DLL that you distribute with your application. You cannot distribute them separately as you can image files.

For more information about creating and using PBR files, see the chapter on packaging your application for deployment in *Application Techniques*.

What happens at runtime

When a resource such as a bitmap is referenced at runtime, PowerBuilder first looks in the executable file for it. Failing that, it looks in the PBDs that are defined for the application. Failing that, it looks in directories in the search path for the file.

Tracing execution

You can trace execution of an executable file built with PowerBuilder. By tracing execution, you can troubleshoot your application if it does not behave the same way when run as an executable file as it does when run in the PowerBuilder development environment. You can also use the trace output to profile your application: for example, you can see how many times particular scripts and functions are being executed.

Two kinds of trace files

You can generate two kinds of trace files:

- **With timing information**   You collect trace information by adding code to the scripts in the application or adding a window that lets users turn tracing on and off. PowerBuilder generates a binary trace file that you analyze using a comprehensive set of objects and functions or the Profiling tools. For more information about tracing and profiling, see "About tracing and profiling an application" on page 891.

- **Without timing information**   You collect information by running the application with the /pbdebug command-line switch. PowerBuilder generates a text file that logs the creation and destruction of objects and the execution of scripts and functions.

Tracing execution using */pbdebug*

You generate PBDebug trace information for an executable file by invoking the executable with a command-line switch.

❖ **To generate PBDebug trace information:**

- Invoke the executable file using the /pbdebug command-line switch:

    EXEFILE /pbdebug

As the application executes, PowerBuilder records the trace output in a file called *exefile.pbg*, which is a text file that you can read in any editor. For information about PBDebug tracing in the development environment, see "Generating a trace file without timing information" on page 920.

Enabling tracing

If you are compiling machine code, you must enable tracing at compile time by selecting Trace Information in the Project painter Compile Options group. If you have not enabled tracing when you compile for machine code, no trace information is generated and the /pbdebug switch has no effect.

If you compile your project in Pcode, the compiler automatically adds the information needed to enable tracing.

Building an executable file and dynamic libraries

Once you have completed development and defined your project, you build the project to create the executable files and all specified dynamic libraries. You can build your project whenever you have made changes to the objects and want to test or deploy another version of your application.

This section describes building a single project in the Project painter. You can build all the targets in your workspace at any time using buttons on the PowerBar, pop-up menus in the System Tree, or a command line. For more information, see "Building workspaces" on page 26.

❖ **To build the application:**

1   Open the project you built in the Project painter.

2   Click the Build button in the PainterBar, or select Design>Build Project.

If the target's library list has changed

When you click Build, PowerBuilder checks your target's library list. If it has changed since you defined your project, PowerBuilder updates the Project painter workspace with the new library list. Make whatever changes you need in the workspace, then click Build again.

PowerBuilder builds the executable and all specified dynamic libraries.

The next two sections describe in detail how PowerBuilder builds the project and finds the objects used in the target.

When PowerBuilder has built the target, you can check which objects are included in the target. See "Listing the objects in a project" on page 938.

How PowerBuilder builds the project

When PowerBuilder builds your application project:

1   If you selected Rebuild: Full, PowerBuilder regenerates all the objects in the libraries.

2   If you selected Prompt for Overwrite, PowerBuilder displays a message box asking for confirmation before overwriting the executable file and each dynamic library.

3   To create the executable file you specified, PowerBuilder searches through your target and copies into the executable file the compiled versions of *referenced* objects from the libraries in the target's library search path that are not specified as dynamic libraries. For more details, see "How PowerBuilder searches for objects" next.

4   PowerBuilder creates a dynamic library for each of the libraries you specified for the target and maintains a list of these library files. PowerBuilder maintains the unqualified file names of the dynamic library files; it does not save the path name.

PowerBuilder does not copy objects that are not referenced in the application to the executable file, nor does it copy objects to the executable file from libraries you declared to be dynamic libraries. These objects are linked to the target at runtime and are not stored in the executable file.

What happens at runtime

When an object such as a window is referenced in the application, PowerBuilder first looks in the executable file for the object. If it does not find it there, it looks in the dynamic library files that are defined for the target. For example, if you specified that a dynamic library should be generated from *test.pbl*, PowerBuilder looks for *test.pbd* or *test.dll* at runtime. The dynamic library files must be in the search path. If PowerBuilder cannot find the object in any of the dynamic library files, it reports a runtime error.

How PowerBuilder searches for objects

When searching through the target, PowerBuilder does not find all the objects that are used in your target and copy them to the executable file. This section describes which objects it finds and copies and which it does not.

Which objects are copied to the executable file

PowerBuilder finds and copies the following objects to the executable file.

Objects that are directly referenced in scripts

PowerBuilder copies objects directly referenced in scripts to the executable file. For example:

- If a window script contains the following statement, w_continue is copied to the executable file:

      Open(w_continue)

- If a menu item script refers to the global function f_calc, f_calc is copied to the executable file:

      f_calc(EnteredValue)

- If a window uses a pop-up menu using the following statements, m_new is copied to the executable file:

      m_new   mymenu
      mymenu = create m_new
      mymenu.m_file.PopMenu(PointerX(), PointerY())

Objects that are referenced in painters

PowerBuilder copies objects referenced in painters to the executable file. For example:

- If a menu is associated with a window in the Window painter, the menu is copied to the executable file.

- If a DataWindow object is associated with a DataWindow control in the Window painter, the DataWindow object is copied to the executable file.

- If a window contains a custom user object that includes another user object, both user objects are copied.

- If a resource is assigned in a painter, it is copied to the executable file. For example, when you place a Picture control in a window in the Window painter, the bitmap file you associate with it is copied.

Which objects are not copied to the executable file

When creating the executable file, PowerBuilder can identify the associations you made in the painter, because those references are saved with the object's definition in the library, and direct references in scripts, because the compiler saves this information.

However, it cannot identify objects that are referenced dynamically through string variables. To do so, it would have to read through all the scripts and process all assignment statements to uncover all the referenced objects. The following examples show objects that are not copied to the executable file:

• If the DataWindow object d_emp is associated with a DataWindow control dynamically using the following statement, d_emp is not copied:

```
dw_info.DataObject = "d_emp"
```

• The bitmap files assigned dynamically in the following script are not copied:

```
IF Balance < 0 THEN
   p_logo.PictureName = "frown.bmp"
ELSE
   p_logo.PictureName = "smile.bmp"
END IF
```

• The reference to window w_go in a string variable in the following window script is *not* found by PowerBuilder when building the executable file, so w_go is not copied to the executable file:

```
window    mywin
string    winname = "w_go"
Open(mywin,winname)
```

Which objects are not copied to the dynamic libraries

When building a dynamic library, PowerBuilder does not inspect the objects; it simply copies the compiled form of the objects. Therefore, the DataWindow objects and resources (pictures, icons, and pointers) used by any of the objects in the library—*either specified in a painter or assigned dynamically in a script*—are not copied into the dynamic library.

For example, suppose *test_dw.pbl* contains DataWindow objects and *test_w.pbl* contains window objects that reference them, either statically or dynamically. If you build a dynamic library from *test_w.pbl*, you must either include the DataWindow objects in a PowerBuilder resource file that is referenced by *test_w.pbl*, or build a dynamic library from *test_dw.pbl*, as described in "How to include the objects that were not found" next.

How to include the objects that were not found

If you did not use any of the types of references described in the preceding sections, you do not need to do anything else to ensure that all objects get distributed: they are all built into the executable file. Otherwise, you have the following choices for how to include the objects that were not found.

Distributing graphic objects

For graphic objects such as icons and bitmaps, you have two choices:

- Distribute them separately

- Include them in a PowerBuilder resource file (PBR), then build an executable file or dynamic PowerBuilder library that uses the resource file

Distributing DataWindow objects

For DataWindow objects, you have two choices:

- Include them in a PBR, then build an executable file or dynamic PowerBuilder library that uses the resource file

- Build and distribute a dynamic library from the PBL that contains the DataWindow objects

Distributing other objects

All other objects, such as windows referenced only in string variables, must be included directly in a dynamic library.

Table 34-3 summarizes resource distribution possibilities.

Table 34-3: Summary: options for distributing resources

Distribution method	Graphic objects	DataWindow objects	Other objects
As a separate file	Yes	No	No
In an executable or dynamic library that references a PBR	Yes	Yes	No
Directly in a dynamic library	No	Yes	Yes

Listing the objects in a project

After you have built your project, you can display a list of objects in the project in a grid DataWindow object with three columns showing:

- The source library that contains the object

- The name of the object

- The type of the object

The report lists the objects that PowerBuilder placed in the executable file and the dynamic libraries it created when it built the project.

Because the report is a grid DataWindow object, you can resize and reorder columns just as you can in other grid DataWindow objects. You can also sort the rows and print the report using the Sort and Print buttons.

❖ **To list the objects in a project:**

1   Build your project.

2   Select Design>List Objects from the menu bar.

Building components and proxies

About the Project painter workspace

The Project painter workspace for executable applications is shown in "Defining an executable application project" on page 926. It has text boxes and radio buttons you use to specify the characteristics of your executable file and dynamic libraries.

The workspace for all other types of project objects is different. It is a read-only display that shows options you specify for the project object. If you used a wizard to create the project, it shows the options you selected in the wizard. If you did not use a wizard, you use Edit menu items or PainterBar buttons to select the objects the project will use and to specify project properties.

When you build the project, the Output window shows whether the build was successful and lists any errors encountered. For most projects, this information also displays in the Project workspace.

Defining and building
projects for
components and
proxies

You create new projects for components and proxies and open them in the
Project painter the same way you do application projects. Use the following
procedure to define and build projects for components and proxies.

❖ **To define and build a project object for components and proxies:**

1 Create a new project and open the Project painter.

For more information, see "Creating a project" on page 925.

2 Select Edit>Select Objects from the menu bar, select the objects you
require, and click OK.

The appearance of the Select Objects dialog box depends on the project
type. For some project types you can select multiple objects.

3 Select Edit>Properties from the menu bar, complete each Property page,
and click OK.

As you set or modify properties, your choices are reflected in the
workspace.

4 Select File>Save to save your choices.

5 Click the Build button in the PainterBar, or select Design>Build Project.

PowerBuilder builds the project. For some components, it also deploys the
generated objects.

Building and
deploying a
workspace

You can build and deploy a single project or all the projects in your workspace.
You can also build and deploy from a command-line. For more information,
see "Building workspaces" on page 26.

For more information
about building
components

Project	See the chapter on this topic	In this book
Automation Server	PowerBuilder automation server	*Application Techniques*
COM/COM+ Component	Building COM/COM+ components	
EAServer Component	Building EAServer components	
EAServer Proxy	Building EAServer clients	
EJB Client Proxy	Building EJB clients	
Web Services Proxy	Building Web service clients	
Web DW Container	Using the Web DataWindow	*DataWindow Programmer's Guide*

P A R T 8

Appendixes

Appendix A describes the extended attribute system
tables. Appendix B describes how to use OrcaScript for
automatic processing of builds and source control
operations.

The Extended Attribute System Tables

About this appendix

This appendix describes each column in the extended attribute system tables.

Contents

Topic	Page
About the extended attribute system tables	943
The extended attribute system tables	944
Edit style types for the PBCatEdt table	947

About the extended attribute system tables

PowerBuilder stores application-based information you provide for a database table (such as the text to use for labels and headings for the columns, validation rules, display formats, and edit styles) in system tables in your database. These system tables are called the extended attribute system tables. The tables contain all the information related to the extended attributes for the tables and columns in the database. The extended attributes are used in DataWindow objects.

The system tables

There are five extended attribute system tables.

Table A-1: List of extended attribute system tables

Table	Contains information about
PBCatTbl	Tables in the database
PBCatCol	Columns in the database
PBCatFmt	Display formats
PBCatVld	Validation rules
PBCatEdt	Edit styles

What to do with the tables

You can open and look at these tables in the Database painter just like other tables. You might want to create a report of the extended attribute information used in your database by building a DataWindow object whose data source is the extended attribute system tables.

Caution
You should not change the values in the extended attribute system tables. PowerBuilder maintains this information automatically whenever you change information for a table or column in the Database painter.

The extended attribute system tables

This section lists and describes all of the columns in each of the extended attribute system tables.

Table A-2: The PBCatTbl table

Column	Column name	Description
1	pbt_tnam	Table name
2	pbt_tid	Adaptive Server Enterprise Object ID of table (used for Adaptive Server Enterprise only)
3	pbt_ownr	Table owner
4	pbd_fhgt	Data font height, PowerBuilder units
5	pbd_fwgt	Data font stroke weight (400=Normal, 700=Bold)
6	pbd_fitl	Data font Italic (Y=Yes, N=No)
7	pbd_funl	Data font Underline (Y=Yes, N=No)
8	pbd_fchr	Data font character set (0=ANSI, 2=Symbol, 255=OEM)
9	pbd_fptc	Data font pitch and family (see note)
10	pbd_ffce	Data font typeface
11	pbh_fhgt	Headings font height, PowerBuilder units
12	pbh_fwgt	Headings font stroke weight (400=Normal, 700=Bold)
13	pbh_fitl	Headings font Italic (Y=Yes, N=No)
14	pbh_funl	Headings font Underline (Y=Yes, N=No)
15	pbh_fchr	Headings font character set (0=ANSI, 2=Symbol, 255=OEM)
16	pbh_fptc	Headings font pitch and family (see note)

Column	Column name	Description
17	pbh_ffce	Headings font typeface
18	pbl_fhgt	Labels font height, PowerBuilder units
19	pbl_fwgt	Labels font stroke weight (400=Normal, 700=Bold)
20	pbl_fitl	Labels font Italic (Y=Yes, N=No)
21	pbl_funl	Labels font Underline (Y=Yes, N=No)
22	pbl_fchr	Labels font character set (0=ANSI, 2=Symbol, 255=OEM)
23	pbl_fptc	Labels font pitch and family (see note)
24	pbl_ffce	Labels font typeface
25	pbt_cmnt	Table comments

About font pitch and family
Font pitch and family is a number obtained by adding together two constants:

Pitch: 0=Default, 1=Fixed, 2=Variable
Family: 0=No Preference, 16=Roman, 32=Swiss, 48=Modern, 64=Script, 80=Decorative

Table A-3: The PBCatCol table

Column	Column name	Description
1	pbc_tnam	Table name
2	pbc_tid	Adaptive Server Enterprise Object ID of table (used for Adaptive Server Enterprise only)
3	pbc_ownr	Table owner
4	pbc_cnam	Column name
5	pbc_cid	Adaptive Server Enterprise Column ID (used for Adaptive Server Enterprise only)
6	pbc_labl	Label
7	pbc_lpos	Label position (23=Left, 24=Right)
8	pbc_hdr	Heading
9	pbc_hpos	Heading position (23=Left, 24=Right, 25=Center)
10	pbc_jtfy	Justification (23=Left, 24=Right)
11	pbc_mask	Display format name
12	pbc_case	Case (26=Actual, 27=UPPER, 28=lower)
13	pbc_hght	Column height, PowerBuilder units
14	pbc_wdth	Column width, PowerBuilder units
15	pbc_ptrn	Validation rule name
16	pbc_bmap	Bitmap/picture (Y=Yes, N=No)

Column	Column name	Description
17	pbc_init	Initial value
18	pbc_cmnt	Column comments
19	pbc_edit	Edit style name
20	pbc_tag	(Reserved)

Table A-4: The PBCatFmt table

Column	Column name	Description
1	pbf_name	Display format name
2	pbf_frmt	Display format
3	pbf_type	Data type to which format applies
4	pbf_cntr	Concurrent-usage flag

Table A-5: The PBCatVld table

Column	Column name	Description
1	pbv_name	Validation rule name
2	pbv_vald	Validation rule
3	pbv_type	Data type to which validation rule applies
4	pbv_cntr	Concurrent-usage flag
5	pbv_msg	Validation error message

Table A-6: The PBCatEdt table

Column	Column name	Description
1	pbe_name	Edit style name
2	pbe_edit	Format string (edit style type dependent; see below)
3	pbe_type	Edit style type (see below)
4	pbe_cntr	Revision counter (increments each time edit style is altered)
5	pbe_seqn	Row sequence number for edit types requiring more than one row in PBCatEdt table
6	pbe_flag	Edit style flag (edit style type dependent; see below)
7	pbe_work	Extra field (edit style type dependent; see below)

Edit style types for the *PBCatEdt* table

Table A-7 shows the edit style types available for the PBCatEdt table.

Table A-7: Edit style types for the PBCatEdt table

Edit style type	pbe_type value (column 3)
CheckBox	85
RadioButton	86
DropDownListBox	87
DropDownDataWindow	88
Edit	89
Edit Mask	90

CheckBox edit style (code 85)

Table A-8 shows a sample row in the PBCatEdt table for a CheckBox edit style. Table A-9 shows the meaning of the values in Table A-8.

Table A-8: Sample row in PBCatEdt for a CheckBox edit style

Name	Edit	Type	Cntr	Seqn	Flag	Work
MyEdit	*Text*	85	1	1	*Flag*	
MyEdit	*OnValue*	85	1	2	0	
MyEdit	*OffValue*	85	1	3	0	
MyEdit	*ThirdValue*	85	1	4	0	

Table A-9: Values used in CheckBox edit style sample

Value	Meaning
Text	CheckBox text
OnValue	Data value for On state
OffValue	Data value for Off state
ThirdValue	Data value for Third state (this row exists only if 3 State is checked for the edit style—bit 30 of *Flag* is 1)

Value	Meaning
Flag	32-bit flag. Low-order four hex digits are generic edit type; high-order four are styles within the type. A 1 in any bit indicates the corresponding style is checked. A 0 in any bit indicates the corresponding style is unchecked.

> Bit 31: Left Text
> Bit 30: 3 State
> Bit 29: 3D
> Bit 28: Scale Box
> Bits 27 – 16 (3 hex digits): Not used (set to 0)
> Bits 15 – 4 (3 hex digits): Always 0 for CheckBox edit style
> Bit 3: Always 0 for CheckBox edit style
> Bit 2: Always 1 for CheckBox edit style
> Bit 1: Always 0 for CheckBox edit style
> Bit 0: Always 0 for CheckBox edit style

RadioButton edit style (code 86)

Table A-10 shows a sample row in the PBCatEdt table for a RadioButton edit style. Table A-11 shows the meaning of the values in Table A-10.

Table A-10: Sample row in PBCatEdt for a RadioButton edit style

Name	Edit	Type	Cntr	Seqn	Flag	Work
MyEdit	*Columns*	86	1	1	*Flag*	
MyEdit	*Display1*	86	1	2	0	
MyEdit	*Data1*	86	1	3	0	
MyEdit	*Display2*	86	1	4	0	
MyEdit	*Data2*	86	1	5	0	

Table A-11: Values used in RadioButton edit style sample

Value	Meaning
Columns	Character representation (in decimal) of number of columns (buttons) across.
Display1	Display value for first button.
Data1	Data value for first button.
Display2	Display value for second button.
Data2	Data value for second button. Display and data values are repeated in pairs for each radio button defined in the edit style.

Value	Meaning
Flag	32-bit flag. Low-order four hex digits are generic edit type; high-order four are styles within the type. A 1 in any bit indicates the corresponding style is checked. A 0 in any bit indicates the corresponding style is unchecked.
	Bit 31: Left Text Bit 30: 3D Bit 29: Scale Circles Bit 38: Not used (set to 0) Bits 27 – 16 (3 hex digits): Not used (set to 0) Bits 15 – 4 (3 hex digits): Always 0 for RadioButton edit style Bit 3: Always 1 for RadioButton edit style Bit 2: Always 0 for RadioButton edit style Bit 1: Always 0 for RadioButton edit style Bit 0: Always 0 for RadioButton edit style

DropDownListBox edit style (code 87)

Table A-12 shows a sample row in the PBCatEdt table for a DropDownListBox edit style. Table A-13 shows the meaning of the values in Table A-12.

Table A-12: Sample row in PBCatEdt for a DropDownListBox edit style

Name	Edit	Type	Cntr	Seqn	Flag	Work
MyEdit	*Limit*	87	1	1	*Flag*	*Key*
MyEdit	*Display1*	87	1	2	0	
MyEdit	*Data1*	87	1	3	0	
MyEdit	*Display2*	87	1	4	0	
MyEdit	*Data2*	87	1	5	0	

Table A-13: Values used in DropDownListBox edit style sample

Value	Meaning
Limit	Character representation (in decimal) of the *Limit* value.
Key	One-character accelerator key.
Display1	Display value for first entry in code table.
Data1	Data value for first entry in code table.
Display2	Display value for second entry in code table.
Data2	Data value for second entry in code table.
	Display and data values are repeated in pairs for each entry in the code table.

Value	Meaning
Flag	32-bit flag. Low-order four hex digits are generic edit type; high-order four are styles within the type. A 1 in any bit indicates the corresponding style is checked. A 0 in any bit indicates the corresponding style is unchecked. Bit 31: Sorted Bit 30: Allow editing Bit 29: Auto HScroll Bit 28: VScroll bar Bit 27: Always show list Bit 26: Always show arrow Bit 25: Uppercase Bit 24: Lowercase (if bits 25 and 24 are both 0, then case is Any) Bit 23: Empty string is NULL Bit 22: Required field Bit 21: Not used (set to 0) Bit 20: Not used (set to 0) Bits 19 – 16 (1 hex digit): Not used (set to 0) Bits 15 – 4 (3 hex digits): Always 0 for DropDownListBox edit style Bit 3: Always 0 for DropDownListBox edit style Bit 2: Always 0 for DropDownListBox edit style Bit 1: Always 1 for DropDownListBox edit style Bit 0: Always 0 for DropDownListBox edit style

DropDownDataWindow edit style (code 88)

Table A-14 shows a sample row in the PBCatEdt table for a DropDownDataWindow edit style. Table A-15 shows the meaning of the values in Table A-14.

Table A-14: Sample row in PBCatEdt for a DropDownDataWindow edit style

Name	Edit	Type	Cntr	Seqn	Flag	Work
MyEdit	*DataWin*	88	1	1	*Flag*	*Limit*
MyEdit	*DataCol*	88	1	2	0	*Key*
MyEdit	*DisplayCol*	88	1	3	0	*Width%*

Table A-15: Values used in DropDownDataWindow edit style sample

Value	Meaning
DataWin	Name of DataWindow object to use.
DataCol	Data column from DataWindow object.
DisplayCol	Display column from DataWindow object.

Value	Meaning
Limit	Character representation (in decimal) of *Limit* value.
Key	One-character accelerator key.
Width%	Width of the dropdown part of the DropDownDataWindow in %.
Flag	32-bit flag. Low-order four hex digits are generic edit type; high-order four are styles within the type. A 1 in any bit indicates the corresponding style is checked. A 0 in any bit indicates the corresponding style is unchecked. Bit 31: Allow editing Bit 30: Auto HScroll Bit 29: VScroll bar Bit 28: Always show list Bit 27: Uppercase Bit 26: Lowercase (if bits 27 and 26 are both 0, then case is Any) Bit 25: HScroll bar Bit 24: Split horizontal scroll bar Bit 23: Empty string is NULL Bit 22: Required field Bit 21: Always show arrow Bit 20: Not used (set to 0) Bits 19 – 16 (1 hex digit): Not used (set to 0) Bits 15 – 8 (2 hex digits): Always 0 for DropDownDataWindow edit style Bit 7: Always 0 for DropDownDataWindow edit style Bit 6: Always 0 for DropDownDataWindow edit style Bit 5: Always 0 for DropDownDataWindow edit style Bit 4: Always 1 for DropDownDataWindow edit style Bit 3 – 0 (1 hex digit): Always 0 for DropDownDataWindow edit style

Edit edit style (code 89)

Table A-16 shows a sample row in the PBCatEdt table for an Edit edit style. Table A-17 shows the meaning of the values in Table A-16.

About the example
This example shows an Edit edit style using a code table of display and data values. There is a pair of rows in PBCatEdt for each entry in the code table *only if* bit 23 of *Flag* is 1.

For information about code tables in edit styles, see Chapter 22, "Displaying and Validating Data."

Table A-16: Sample row in PBCatEdt for an Edit edit style

Name	Edit	Type	Cntr	Seqn	Flag	Work
MyEdit	*Limit*	89	1	1	*Flag*	*Key*
MyEdit	*Format*	89	1	2	0	*Focus*
MyEdit	*Display1*	89	1	3	0	
MyEdit	*Data1*	89	1	4	0	
MyEdit	*Display2*	89	1	5	0	
MyEdit	*Data2*	89	1	6	0	

Table A-17: Values used in Edit edit style sample

Value	Meaning
Limit	Character representation (in decimal) of *Limit* value.
Key	One-character accelerator key.
Format	Display format mask.
Focus	Character "1" if Show Focus Rectangle is checked. NULL otherwise.
Flag	32-bit flag. Low-order four hex digits are generic edit type; high-order four are styles within the type. A 1 in any bit indicates the corresponding style is checked. A 0 in any bit indicates the corresponding style is unchecked.

> Bit 31: Uppercase
> Bit 30: Lowercase (if Bits 31 and 30 are both 0, then case is Any)
> Bit 29: Auto selection
> Bit 28: Password
> Bit 27: Auto HScroll
> Bit 26: Auto VScroll
> Bit 25: HScroll bar
> Bit 24: VScroll bar
> Bit 23: Use code table
> Bit 22: Validate using code table
> Bit 21: Display only
> Bit 20: Empty string is NULL
> Bit 19: Required field
> Bit 18: Not used (set to 0)
> Bit 17: Not used (set to 0)
> Bit 16: Not used (set to 0)
> Bits 15 – 4 (3 hex digits): Always 0 for Edit edit style
> Bit 3: Always 0 for Edit edit style
> Bit 2: Always 0 for Edit edit style
> Bit 1: Always 0 for Edit edit style
> Bit 0: Always 1 for Edit edit style

Edit Mask edit style (code 90)

Table A-18 shows a sample row in the PBCatEdt table for an EditMask edit style. Table A-19 shows the meaning of the values in Table A-18.

About the example

This example shows an Edit Mask edit style using a code table of display and data values as part of a spin control. Rows 2 and beyond exist in PBCatEdt only if the edit mask is defined as a spin control (bit 29 of *Flag* is 1). Rows 3 and beyond exist only if the optional code table is populated.

For information about using an edit mask as a spin control, see Chapter 22, "Displaying and Validating Data."

Table A-18: Sample row in PBCatEdt for an EditMask edit style

Name	Edit	Type	Cntr	Seqn	Flag	Work
MyEdit	*Format*	90	1	1	*Flag*	*DtFcKy*
MyEdit	*Range*	90	1	2	0	*SpinInc*
MyEdit	*Display1*	90	1	3	0	
MyEdit	*Data1*	90	1	4	0	
MyEdit	*Display2*	90	1	5	0	
MyEdit	*Data2*	90	1	6	0	

Table A-19: Values used in EditMask edit style sample

Value	Meaning
Format	Display format mask.
DtFcKy	Concatenated string with 1-character data-type code, 1-character focus-rectangle code (0 or 1), and 1-character accelerator key.
	Data type codes:
	Format String = "0"
	Format Number = "1"
	Format Date = "2"
	Format Time = "3"
	Format DataTime= "4"
	Examples:
	"10x" means format is Number type, focus rectangle option is unchecked, accelerator key is "x" "31z" means format is Time type, focus rectangle option is checked, accelerator key is "z"

Value	Meaning
Range	Character representation (in decimal) of spin control range. The min value and max value are tab-delimited. Example: "1[tab]13" means min = 1, max = 13
SpinInc	Character representation (in decimal) of spin increment.
Display1	Display value for first entry in code table.
Data1	Data value for first entry in code table.
Display2	Display value for second entry in code table.
Data2	Data value for second entry in code table. Display and data values are repeated in pairs for each entry in the code table.

Flag	32-bit flag. Low-order four hex digits are generic edit type; high-order four are styles within the type. A 1 in any bit indicates the corresponding style is checked. A 0 in any bit indicates the corresponding style is unchecked.
	
	  Bit 31: Required
	  Bit 30: Autoskip
	  Bit 29: Spin control
	  Bit 28: Read only (code table option)
	  Bit 27: Use code table
	  Bit 26: Not used (set to 0)
	  Bit 25: Not used (set to 0)
	  Bit 24: Not used (set to 0)
	  Bit 23 – 16 (2 hex digits): Not used (set to 0)
	  Bit 15 – 8 (2 hex digits): Always 0 for Edit Mask edit style
	  Bit 7: Always 0 for Edit Mask edit style
	  Bit 6: Always 0 for Edit Mask edit style
	  Bit 5: Always 1 for Edit Mask edit style
	  Bit 4: Always 0 for Edit Mask edit style
	  Bits 3 – 0 (1 hex digit): Always 0 for Edit Mask edit style

The OrcaScript Language

About this appendix

This appendix describes the OrcaScript scripting language. OrcaScript allows you to perform source control operations and build PowerBuilder workspaces and executables without operator intervention. The full ORCA tool kit is available to Sybase partners only, but OrcaScript can be used by any PowerBuilder customer.

Contents

About OrcaScript

OrcaScript allows you to write batch scripts to process PowerBuilder applications and files without using the PowerBuilder development environment. You can use OrcaScript to get the latest version of a PowerScript target from source control, build the target PBLs, deploy components to EAServer, and compile PowerBuilder executable files—all without operator intervention.

Using OrcaScript with source control

The targets you obtain from source control using OrcaScript could be placed on a network build machine that is shared by PowerBuilder developers. This is especially advantageous for large shops with fixed working hours: the builds could be done nightly by running an OrcaScript batch file, and an up-to-date version of the targets and libraries would be available at the start of the next work day.

Developers could then use OrcaScript or operating system commands to copy the shared files directly to their local machines. Although developers would still connect directly to source control from their local workspaces, refreshing the targets in the workspaces would be much faster since compilation times for complex PowerScript targets would be greatly minimized.

Batch file order

If you include OrcaScript commands in a batch file, the file is read line by line. Each OrcaScript batch file must begin with a start session command and end with an end session command. You can save the batch file with any extension. You run the batch file by calling the OrcaScript executable on a command line and passing the batch file name as an argument:

```
OrcaScr105 myOrcaBat.dat
```

If you use relative directories in the OrcaScript batch file, create the batch file in the directory that is the required root directory at runtime. This must be in the same directory or in the path above a directory containing the files referenced by the batch file.

When you use relative directories, the OrcaScript batch file is portable for all users. However, users must make the directory where they copy the batch file the current directory (the one displayed in the DOS prompt) before invoking *OrcaScr105.exe*. The command to start the OrcaScript executable can also take the following parameters:

Parameter	Description	Example
/D	Sets variables that are valid in the batch file	`OrcaScr105 /D myVar1=value1 /D myVar2=value2 myOrca.dat`
/H or /?	Prints syntax help to screen	`OrcaScr105 /H`

Caution
You should not run an OrcaScript batch file if PowerBuilder is currently running on the same machine. If the PowerBuilder development environment is not shut down while OrcaScript is running, your PowerBuilder libraries can become corrupted. For this reason, casual use of OrcaScript is not recommended.

Error handling

Each line of an OrcaScript batch file either succeeds or fails. If a command fails, subsequent commands are not processed and the OrcaScript session is ended. An error message is printed to the command window.

Comments

A semicolon (;) indicates that the rest of the line is treated as a comment.

OrcaScript Commands

OrcaScript commands are not case sensitive. The generic command parameters can include only strings delimited by quotation marks, or predefined variables and constants without quotation marks. White space is used to separate multiple parameters for a single command. Any place a string is expected, a name that has been previously defined (set) in an OrcaScript command can be used.

In the OrcaScript command prototype syntax that follows, brackets indicate a parameter is optional. A pipe character inside angle brackets (< | >) indicates that a selection must be made from one of the values inside the angle brackets. As elsewhere in the PowerBuilder documentation, text in italic type indicates a variable.

For commands where a string variable is required by the command syntax but is not essential to the command function (such as *pbrName* for the build library command), you can use an empty string inside quotation marks for the string value. Most of the OrcaScript commands and parameters are self-explanatory. For usage notes and an example of an OrcaScript batch file for obtaining a PowerScript target from source control, see "Usage notes for OrcaScript commands and parameters" next.

OrcaScript commands

OrcaScript supports the following commands:

> start session
> end session
> set *name* = *value*
> set *name* += *value*
> set liblist *pbl_list* [*pbl_list* ...]
> set application *pblName applicationName*
> set exeinfo property <companyname | *productname | copyright | description>*
>   *propertyString*
> set exeinfo property <fileversion | fileversionnum | productversion |
>   productversionnum> *versionString*
> echo *value* [*value* ...]
> file copy *fromFile toFile* [*clobberAttribute*]
> regenerate *pblName entryName entryType*
> copy entry *pblName entryName entryType toPblName*
> build library *pblName pbrName* <pbd | 32>
> build executable *exeName iconName pbrName pbdflags* [machinecode]
>   [newvstylecontrols]
> build application <full | migrate | incremental >
> build project *pblName projectName* [*serverName serverPort*]
> create library *pblName pblComments*

scc get connect properties *workspaceName*
scc set connect property deletetempfiles <true|false>
scc set connect property provider *sccProvider*
scc set connect property userid *userID*
scc set connect property password *password*
scc set connect property logfile *logFileName*
scc set connect property project *projectPath*
scc set connect property localprojpath *localProjectPath*
scc set connect property auxproject *auxProjectPath*
scc set connect property logappend < true | false >
scc connect [offline]
scc set target *targetName* [*refreshType*][*refreshOption*][*refreshOption*]
scc get latest version *file_list* [*file_list* ...]
scc exclude liblist *pblName* [*pblName* ...]
scc refresh target <full | migrate | incremental >
scc close

Argument description

Arguments for OrcaScript commands are described in the table below:

Argument	Description
name	String you define for an OrcaScript session.
value	Value of a string that you set for the OrcaScript session.
pbl_list	String containing the list of PBLs for the session application. PBL names can be separated by semicolons in a single string, or separated by a blank space in multiple strings.
pblName	Name of a PBL for an OrcaScript action or for the OrcaScript session application.
applicationName	Name of the application for an OrcaScript action.
propertyString	String for setting the company or product name, copyright owner, or application description.
versionString	String for setting the product or file version numbers. The FileVersionNum and ProductVersionNum strings must consist of four integer values representing the major version number, minor version number, fix version number, and build number, with each value separated by a decimal point, for example "10.5.0.3012".
fromFile	File that you want to copy during an OrcaScript session.
toFile	File name for a file that you copy during an OrcaScript session.

Argument	Description
clobberAttribute	Determines whether the file copy command overwrites an existing file. If the destination file does not already exist, the file copy command creates the file regardless of the *clobberAttribute* value you select. Possible values are: • **Clobber (default)** File copy command overwrites an existing file marked read/write, but does not overwrite an existing file marked read-only • **NoClobber** File copy command does not overwrite an existing file even if it is marked read/write • **Clobber Always** File copy command overwrites an existing file even if it is marked read-only
entryName	Pointer to a string whose value is the name of the referenced object.
entryType	Value specifying the type of the referenced object. Values can be: application, datawindow, function, menu, query, struct, userobject, window, pipe, project, or proxy. Certain abbreviations (app, dw, fn, struct, uo, and win) are allowed as substitute values.
toPblName	Name of the PBL to which you copy an entry.
pbrName	Name of a resource file you want to include in a build.
pbd | 32	Select PBD to generate PowerBuilder dynamic libraries. Select 32 to generate platform-specific machine code. You must enter a full path for a PBL or PBR if you select 32 as the value of this argument in an OrcaScript build library command.
exeName	Name of the executable you want to build.
iconName	Name of an icon to use for an executable you build with OrcaScript.
pbdFlags	String composed of a series of Y and N values for each library in the library list. A value of "nnyy" indicates that there are four libraries in the library list, the last two being PBDs. Objects from PBLs are copied into the executable; objects from PBDs are not copied.
machinecode	Use to compile a project as machine code.
newvstylecontrols	Use Microsoft XP visual style for controls.
full | migrate | incremental	Build strategy for the session application.
projectName	Name of the project object you want to build and deploy.
serverName	Name of the server where you want to deploy a project.
serverPort	Port for the server where you want to deploy a project.

Argument	Description
pbdName	Name of a PBD you append to an EXE.
pblComments	Comments for a PBL you create in an OrcaScript session.

Arguments for source control commands

In addition to some of the arguments listed in the preceding table, OrcaScript source control commands use the following arguments:

Argument	Description
workspaceName	Name of the workspace to connect to source control. You must include the path to the workspace, although you can use a relative path.
sccProvider	Name of the source control provider.
userID	Name of the user registered to source control.
password	Password for the user ID.
logFileName	Name of a log file used to record SCC transactions.
projectPath	Path to the source control project.
localProjectPath	Local root directory for the project.
auxProjectPath	Contains any string that the SCC provider wants to associate with the project. It has a different meaning for every SCC vendor.
targetName	Name of the target for source control operations.
true | false	Boolean value for appending to the source control log file. If this command is not used but a log file is specified, the session value defaults to true.
offline	Keyword indicating that an actual SCC connection will not be required for this session. It is appropriate only when the ImportOnly refresh option is used on a subsequent `scc set target` command. When refreshing a target using ImportOnly, no communication with the SCC provider is required at runtime, so the job may be run offline.
refreshType	Value can be: • **refresh_all** Gets latest version of all objects from the SCC provider and refreshes all target libraries. Does not perform comparisons. • **outofdate** Performs comparisons and updates objects that are out of date. If no *refreshType* value is specified, the *refreshType* defaults to outofdate. **Combining values** You can combine compatible *refreshType* and *refreshOption* values (for example, outofdate and exclude_checkout) in the same string if the values are separated by a blank space.

Argument	Description
refreshOption	Value can be:
	• **importonly** Does not perform comparisons and does not refresh. Use to build targets if you refreshed the local path using the SCC provider's administration tool.
	• **exclude_checkout** Prevents objects that are currently checked out by the current user from being overwritten. Can be used with outofdate parameter in the same OrcaScript command.
file_list	String containing one or more resource file names (such as GIFs, HLPs, or PBRs) using relative or absolute path specification. The string should not include file names for objects contained in application PBLs. File names can be separated by semicolons in a single string, or separated by a blank space in multiple strings. The list of files must be on a single line even when file names are contained in multiple strings.

Usage notes for OrcaScript commands and parameters

Before calling any other ORCA functions, you need to open a session:

```
start session
```

You can start and end multiple OrcaScript sessions in the same batch file.

Copying files, objects, and properties

If you want to use OrcaScript simply to move objects among libraries, you do not need to set a library list or application. You can use the copy commands to copy files, objects, and properties. This example copies the d_labels DataWindow from the *source.pbl* library to the *destin.pbl* library:

```
copy entry "c:\\app\\source.pbl""d_labels" dw
"c:\\app\\destin.pbl"
```

Setting a library list and an application

If you want to use OrcaScript to build targets or deploy components (or set product and version information using the set exeinfo command) you must first set the library list and the current application. You can set the library list and current application only once in an OrcaScript session. To use another library list and application, end the OrcaScript session and start a new session. The following OrcaScript commands build target libraries and compile an executable file.

```
start session
set liblist
".\qadbtest\qadbtest.pbl;.\shared_obj\shared_obj.pbl;.\datatypes\datatype.
pbl;.\chgreqs\chgreqs.pbl"
set application ".\qadbtest\qadbtest.pbl" "qadbtest"
build library ".\shared_obj\shared_obj.pbl" "" pbd
build library ".\datatypes\datatype.pbl" "" pbd
build library ".\chgreqs\chgreqs.pbl" "" pbd
build executable  ".\qadbtest\qadbtest.exe" ".\emp.ico" ".\qadbtest.pbr"
"nyyy"
file copy ".\qadbtest\qadbtest.exe" ".\bin\qadbtest.exe"
file copy ".\chgreqs\chgreqs.pbd" ".\bin\chgreqs.pbd"
file copy ".\datatypes\datatype.pbd" ".\bin\datatype.pbd"
file copy ".\shared_obj\shared_obj.pbd" ".\bin\shared_obj.pbd"
end session
```

You can use relative paths when you generate PBDs with the "PBD" option, but the PBD always gets generated in the same directory as the PBL. To actually run the executable, you might have to move the PBDs to a "BIN" directory. The above example calls several file copy commands to accomplish this.

If you select 32 as the last argument in a build library command, you must use the full path for the PBL or PBR name included in that call.

Source control example

You can use OrcaScript source control commands instead of the commands to set the library list and application. The following is an example of an OrcaScript session that builds the same libraries as the previous example, but uses the target properties to set a library list and application:

```
start session
scc get connect properties "testbld\testbld.pbw"
scc connect
scc set target "c:\testbld\qadbtest\qadbtest.pbt" "outofdate
exclude_checkout"
scc refresh target "incremental"
build library ".\shared_obj\shared_obj.pbl" "" pbd
build library ".\datatypes\datatype.pbl" "" pbd
build library ".\chgreqs\chgreqs.pbl" "" pbd
build executable  ".\qadbtest\qadbtest.exe" ".\emp.ico" ".\qadbtest.pbr"
"nyyy"
scc close
end session
```

You can call the scc connect command only after getting connection properties, and you must call it before you set or refresh the source-controlled targets. You must call the scc close command before you end your OrcaScript session.

Shared library
example

If you have another target that shares libraries with a target that you already refreshed, you can use the OrcaScript exclude command to quickly reconstitute your target. The following example excludes the shared libraries *shared_obj.pbl*, *datatype.pbl*, and *chgreqs.pbl* that were refreshed in the previous example. It also demonstrates the use of variables for refresh options and build type. Set statements define variables that can be used throughout an OrcaScript session wherever the parser expects a string token.

```
start session
set refresh_flags = "outofdate"
set refresh_flags += "exclude_checkout"
set build_type = "incremental"
scc get connect properties "c:\testbld\testbld.pbw"
scc connect
scc set target ".\dbauto\dbauto.pbt" refresh_flags
scc exclude liblist ".\shared_obj\shared_obj.pbl"
".\datatypes\datatype.pbl" ".\chgreqs\chgreqs.pbl"
scc refresh target  build_type
build executable ".\dbauto\dbauto.exe" ".\emp.ico" "" "nyyy"
scc close
end session
```

Defining variables from the command line
Instead of defining variables in the OrcaScript session, you can define them from the command line when you call your script. If you saved the OrcaScript example in the previous script in a file named *MyExample.dat*, you could set the same variables by typing the following at a command line prompt:

```
 Orcascr105 /D refresh_flags="outofdate exclude_checkout"
 /D build_type="incremental"  MyExample.dat
```

SCC connection
properties

The SCC get connect properties command is an easy way to populate the Orca SCC connection structure with the source control properties of a local workspace. However, to create OrcaScript batch files that are portable from one workstation to another, the recommended technique is to set each property explicitly. Many of these properties are vendor specific. The best way to obtain correct values is to copy them directly from the SCC log file for your PowerBuilder workspace.

After you have obtained the values you need from the SCC log file, you can create portable batch files by setting the required connection properties and using relative directories and URLs for path information. The following example shows portable OrcaScript batch file commands for a workspace that connects to PBNative:

```
start session
scc set connect property provider "PB Native"
scc set connect property userid "Jane"
scc set connect property localprojpath ".\"
scc set connect property project "\\network_machine\PBNative_Archive\qadb"
scc set connect property logfile ".\MyPortableExample.log"
scc set connect property logappend "FALSE"
scc set connect property deletetempfiles "FALSE"
scc connect
; Perform refresh and build operations
scc close
end session
```

Using OrcaScript with source control targets offline

You can call scc connect offline to build source control targets offline. When you use this command, you must specify ImportOnly as a refresh option. If you also specify the Refresh_all option or the OutOfDate or Exclude_checkout refresh types, no connection is made to source control.

For the OutOfDate refresh type, the object source residing in the PBL is compared with the object source on the local project path. If these object sources are different, the object source on the local project path is imported and compiled.

For the Exclude_checkout refresh type, the workspace PBC file is used to determine current status. In order for the offline exclude_checkout processing to locate the PBC file, you must use the scc get connect properties *workspaceName* command at the beginning of the script. Objects marked as checked out to the current user in the PBC file will not be imported into the PBLs during target processing.

Applicable scc connect properties for offline processing
When scc connect offline is used, only the following connect properties apply:

scc set connect property localprojpath *localProjectPath*
scc set connect property logfile *logFileName*
scc set connect property logappend <true | false>

Setting refreshType
and refreshOption
values

When you set up a target with a source control connection, you can use
refreshType and *refreshOption* options in various combinations. You can even
combine these values in the same string if the values are separated by a blank
space. For example:

```
scc set target ".\dbauto\dbauto.pbt" "refresh_all
    importonly"
```

Refresh_all option   Refresh_all performs no comparisons and imports all
applicable objects. Refresh_all behavior varies depending on whether
ImportOnly is set. If ImportOnly is off, Refresh All issues an
SccGetLatestVersion call to obtain the tip revision of the PBT file specified in
the scc set target command. From the PBT file, it obtains the library list for the
target. It then calls SccGetLatestVersion to obtain the tip revision of the PBG
file associated with each PBL.

Each PBG file contains a list of objects registered under source control that
reside in the associated PBL. Refresh All then issues SccGetLatestVersion to
obtain the tip revision of each object and imports these objects into the PBL.

In offline processing, ImportOnly must be set to on. If you also set the
Refresh_all option, the PBT file that already exists on the local project path is
used to obtain the library list for the target. The PBG file that also exists on the
local project path is then read to obtain a list of objects associated with each
PBL. Refresh_all then processes the PBG lists, importing source entries
residing on the local project path into the appropriate PBL.

ImportOnly option   When ImportOnly is on, the expectation is that the user
has already populated the local project path with the desired revision of each
object. ImportOnly is used to build a target from a previous milestone, such as
a version label or a promotion model that does not represent the tip revision of
each object. Therefore, no SccGetLatestVersion calls are issued. The desired
revisions of the PBT, PBG, and object source files must already exist on the
local project path and they are used to import objects into the PBLs. You must
use this option if you are building a source controlled target while you are
offline.

OutOfDate option   OutOfDate processing behaves differently depending on
whether ImportOnly is set. When ImportOnly is off, OutOfDate issues an
SccGetLatestVersion call to obtain the tip revision of the PBT and PBG files. It
then compares each object in the target PBLs with the tip revision in the SCC
repository and imports the SCC source files into the PBLs for the objects that
are out of sync.

With ImportOnly turned on, OrcaScript never performs GetLatestVersion since the desired revision of all objects already exists on the local project path. In this case, OutOfDate processing compares source code in the PBL against object source on the local project path to decide which objects, if any, need to be reimported. Using ImportOnly with OutOfDate processing works the same whether you are online or offline.

Advantage of using OutOfDate with ImportOnly option

Combining the OutOfDate option with the ImportOnly option is particularly useful if you perform nightly builds of a project that has several promotion models defined. If the volume of changes is low, it may be more efficient to use OutOfDate processing rather than Refresh All. In one PowerBuilder workspace, you build the "development" view of the project that includes all development work in progress. In another workspace, you build the "maintenance" view of the project, which includes bug fixes waiting for QA verification. Elsewhere, you build a "production" view of the project containing only verified bug fixes.

Each PowerBuilder workspace connects to the same SCC project, but uses a different local project path. You use your vendor-specific SCC administration tool to synchronize the local project path with the desired revision of each object belonging to each promotion model. Then you launch OrcaScript to refresh the PBLs in each workspace. This results in a nightly rebuild of all three promotion models, which development team members can download each morning from a shared network drive.

Exclude_checkout option   The Exclude_checkout option excludes from the import list all objects that are currently checked out by the current user, no matter what other refresh options are used. When connected to SCC, this option requires an additional call to SccQueryInfo for each object in the target. Therefore, it is not recommended on a nightly build machine. However, it is highly recommended when a developer uses OrcaScript on his or her own workstation.

If you use Exclude_checkout processing while offline, the workspace PBC file is used to determine current status, so you must specify the set get connect properties workspaceName command. Objects marked as checked out to the current user in the PBC file will not be imported into the PBLs during target processing.

How the current user is determined for Exclude_checkout processing
For online SCC connections, Exclude_checkout calls scc connect property userid *userID* or the scc get connect properties *workspaceName* to determine the current user. The runtime processing makes actual SccQueryInfo calls to the SCC provider to determine check out status, so the information in the PBC file (from the prior SCC connection) is ignored. Objects checked out to the current user are not imported and replaced in the target library list.

For scc connect offline, the scc connect property userid command is completely ignored. OrcaScript must rely on information from the prior SCC connection. Each PBC file entry contains a bit flag that indicates "checked out to current user". This flag determines whether the object is imported or excluded. The current user at the time the PBC file was created is the user who last connected to this workspace through the PowerBuilder IDE on this workstation.

Build command failures	OrcaScript build commands for an executable or a library fail if the executable or library already exists in the build directory. To prevent an OrcaScript batch file containing these commands from failing, move or delete existing executables and libraries from the build directory before running the batch script.
Escape characters for string variables	OrcaScript, like PowerScript, uses the tilde (~) as an escape character. If you need to include a special character, such as a quotation mark, inside a string, you must place a tilde in front of it. A character in an OrcaScript batch file with a tilde in front of it is processed as a literal character.
Ending an OrcaScript session	You must close an OrcaScript session after you finish calling other OrcaScript commands. You close an OrcaScript session by calling:

```
end session
```

Property values are deleted during end-session processing. If an OrcaScript program starts numerous sessions, each individual session must contain statements to specify property values, such as those assigned in set exeinfo or scc set connect commands. However, variables that you set on a batch script command line using the /D parameter, or inside a batch file using the set *variable_name*="value" syntax, remain valid for the entire multisession program.

Index

PowerBuilder

K

J

X

Y

Z